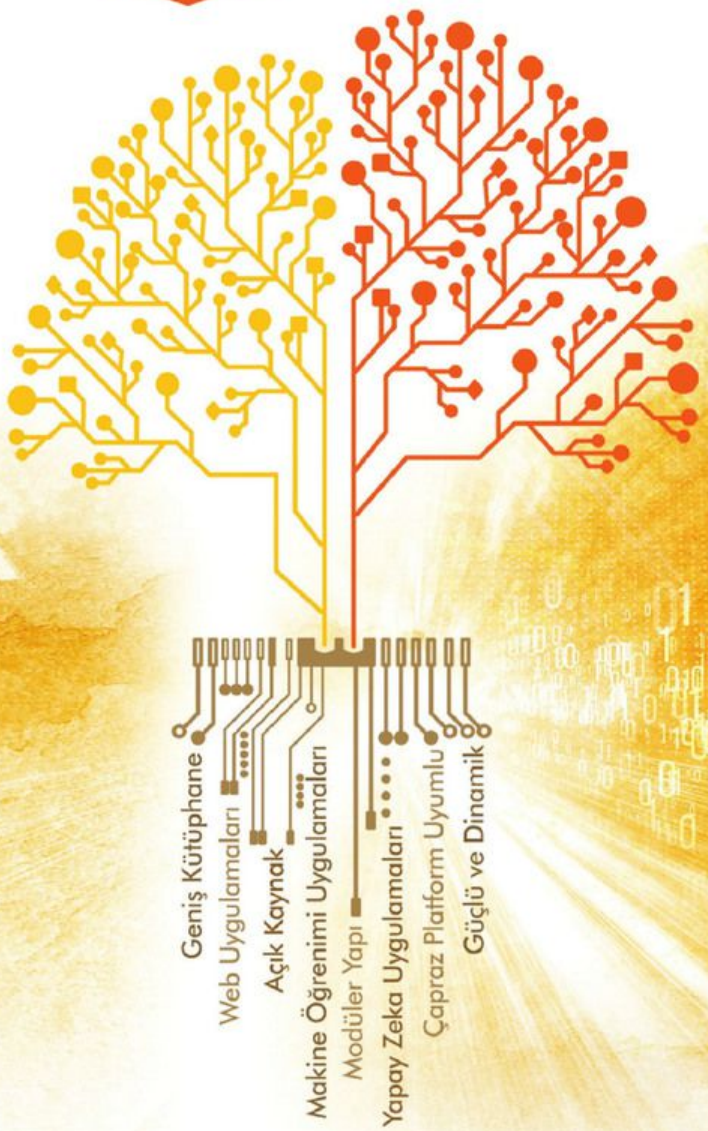
PYTHON EĞİTİMİ

HERKES İÇİN PYTHON PROGRAMLAMA DİLİ



Birleşmiş Milletler Eğitim, Bilim ve Kültür Kurumu
UNESCO Türkiye Milli Komisyonu

ÖĞRETMEN YETİŞTİRME VE GELİŞTİRME GENEL MÜDÜRLÜĞÜ

Birleşmiş Milletler Eğitim, Bilim ve Kültür Kurumu

UNESCO Türkiye Millî Komisyonu

Birleşmiş Milletler
Eğitim, Bilim ve Kültür
Kurumu

UNESCO
Türkiye
Millî Komisyonu

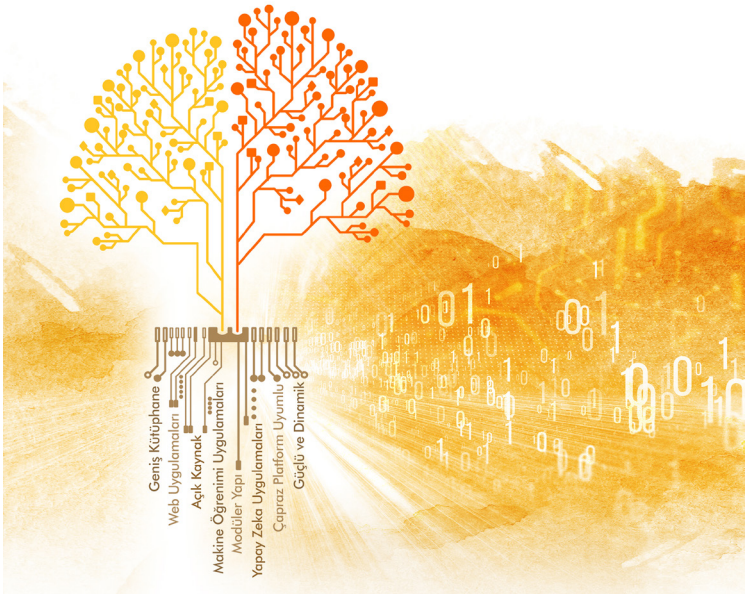
Millî Eğitim Bakanlığı
Öğretmen Yetiştirme ve Geliştirme Genel Müdürlüğü Adına
Doç. Dr. Adnan BOYACI
Genel Müdür

Editör
Sibel AKBIYIK
Daire Başkanı

Yazarlar
Dr. Murat ALTUN
Nihat BAL
Resul BÜTÜNER
Serhat Kağan ŞAHİN
Murat KOÇALİ

Grafik Tasarım
Serkan AKYÖRÜK

Kapak Tasarım
Meliha BAKA ÇAKMAKLI



ISBN: 978-975-11-5338-8

2020

Python Eğitimi

T.C. MİLLÎ EĞİTİM BAKANLIĞI | Öğretmen Yetiştirme ve Geliştirme Genel Müdürlüğü

Doç. Dr. Adnan BOYACI
ÖĞRETMEN YETİŞTİRME VE GELİŞTİRME GENEL MÜDÜRÜ

ÖNSÖZ

Değerli Meslektaşlarım,

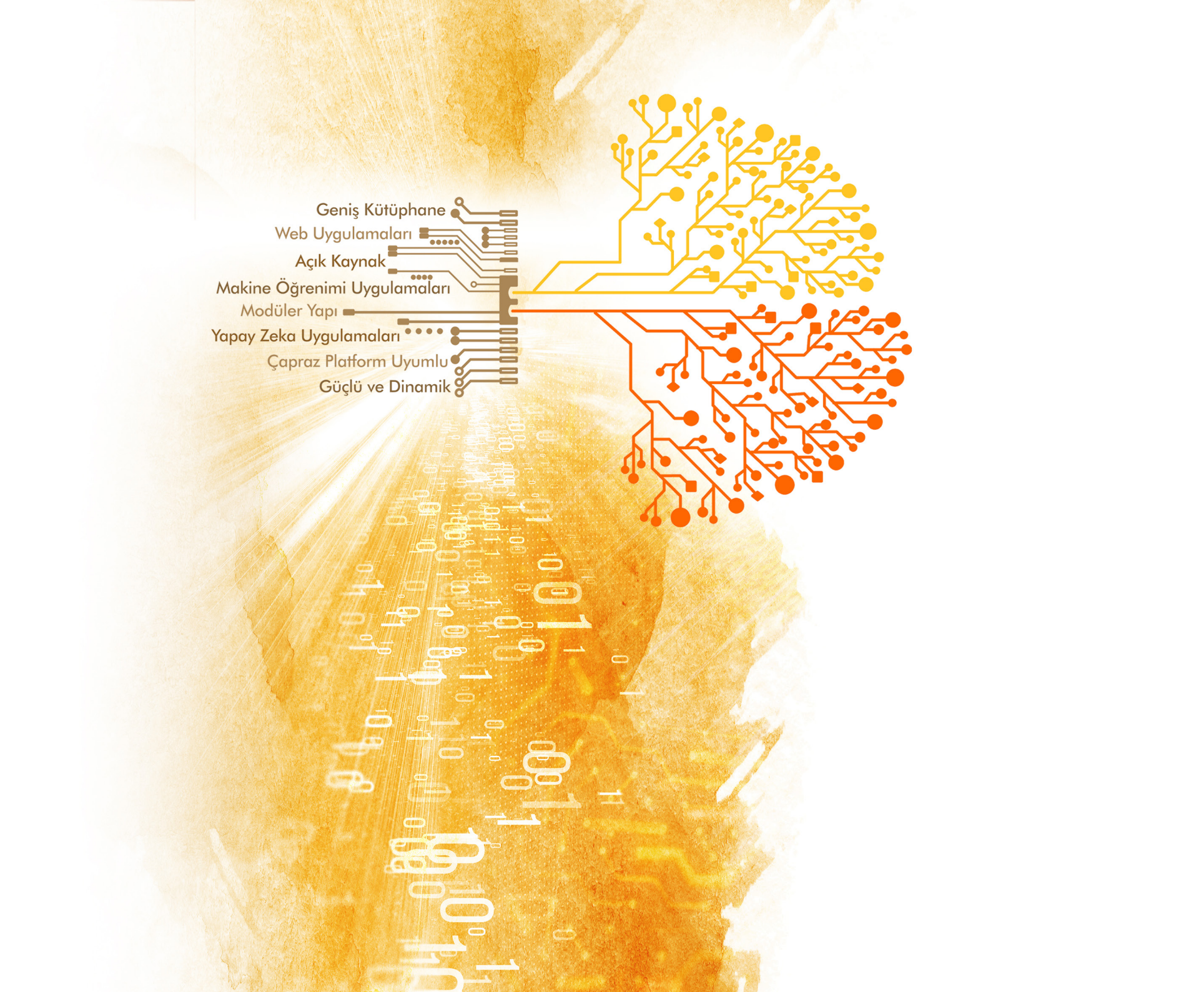
Millî Eğitim Bakanlığı Öğretmen Yetiştirme ve Geliştirme Genel Müdürlüğünün Python Eğitimi kitabı, programlama dili Python hakkında tüm öğretmenlerimizin bilgi sahibi olmalarına temel seviyede katkı sağlaması amacıyla hazırlanmıştır. Bu kitap, öğretmenlerimizin değişen çağın gereklerine ayak uydurmaları, mesleki yeterliliklerinin güçlendirilmesi ve geliştirilmesini hedeflemektedir.

Python Eğitimi kitabı, 16 modülden oluşmakta olup Python Programlama Dilinin indirme ve yükleme aşamalarından fonksiyonlarına, özelliklerinden modüler yapısına kadar pek çok konuda ayrıntılı bilgi içermektedir. Kitapta yer alan konular sıralı bir şekilde verilmiş konulardaki örnek kodlar da öğretmenlerimizin yararlanabilmeleri için çevrim içi ortamda sunulmuştur. Son yılların en popüler programlama dilleri arasında yer alan; veri bilimi, veri analizi ve yapay zekâ ile ilgili gelişmiş kütüphaneleri olan Python Programlama Dili, bu alanlarda ilerlemek isteyenler için de eşsiz bir araçtır. Geleceğin programlama dili olarak görülen Python, açık kaynak kodlu olması, esnek ve modüler yapısı nedeniyle çok kullanışlıdır. Bu özellikleriyle öğretmenlerimizin mesleki birçok konuda ihtiyaçlarını gidermeye ve donanımlarını geliştirmeye uygun bir programlama dilidir. Kitap sayesinde ulusal ve küresel ölçekte kabul gören Python Programlama Dili eğitimiyle öğretmenlerimizin dijital yeterliklerini geliştirmeyi umuyoruz.

Öğretmenlerimizin mesleki gelişimi için çalışan bilişim teknolojileri öğretmenlerine ve değerli meslektaşlarıma verdikleri katkı ve destekten dolayı teşekkür eder, bu çalışmanın öğretmenlerimiz ile öğrencilerimize faydalı olması temennisiyle sevgi ve saygılarımı sunarım.

Doç. Dr. Adnan BOYACI

ÖĞRETMEN YETİŞTİRME VE GELİŞTİRME GENEL MÜDÜRÜ


Geniş Kütüphane
Web Uygulamaları
Açık Kaynak
Makine Öğrenimi Uygulamaları
Modüler Yapı
Yapay Zeka Uygulamaları
Çapraz Platform Uyumlu
Güçlü ve Dinamik

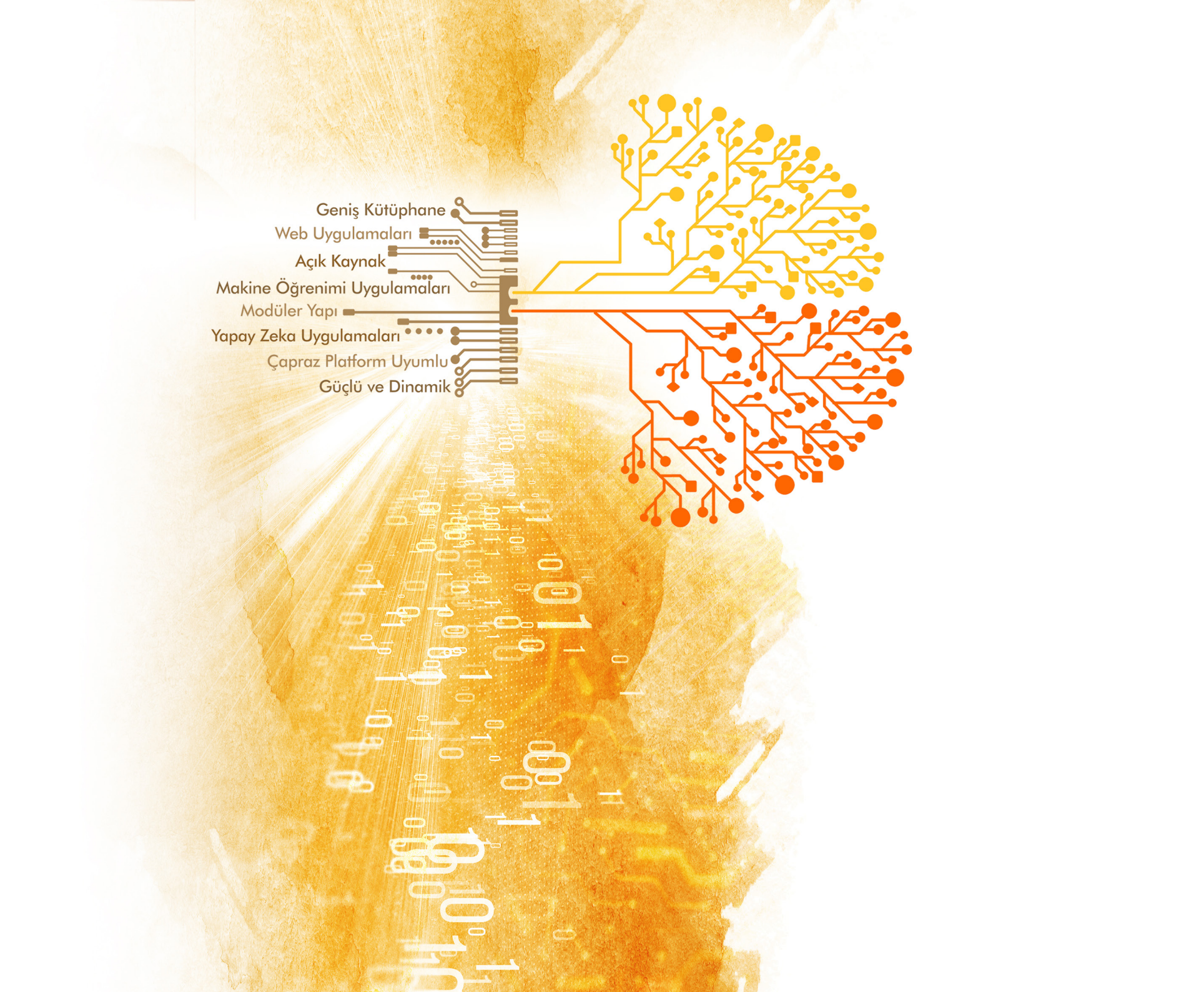
İÇİNDEKİLER

İÇİNDEKİLER

İÇİNDEKİLER

İÇİNDEKİLER

İÇİNDEKİLER


Geniş Kütüphane
Web Uygulamaları
Açık Kaynak
Makine Öğrenimi Uygulamaları
Modüler Yapı
Yapay Zeka Uygulamaları
Çapraz Platform Uyumlu
Güçlü ve Dinamik

MODÜL 1

PYTHON PROGRAMLAMA DİLİ

1.1. Bilgisayar Programları Nasıl Çalışır?

Bilgisayarlar **yazılım** ve **donanım** olmak üzere iki bileşenden oluşur. Donanımlar, bilgisayarların fiziksel parçalarıdır. Klavye, fare, ekran gibi bileşenler bilgisayarın donanım birimleridir. Yazılım ise bilgisayarda donanıma hayat veren ve bilgi işlemde kullanılan programlar, yordamlar, programlama dilleri ve belgelemelerin tümüdür. (TDK, 2020)

Bilgisayarlarda bulunan oyunlar, müzik programları, ofis programları hatta işletim sistemleri dâhil tüm yazılımlar bir programlama dili ile yazılmıştır.

Bilgisayarlar kendilerine verilen görevleri yerine getirirken bu işi doğası gereği değil, sahip oldukları işlemci komut setine ve işletim sisteminin yürütebileceği özellikte hazırlanmış yönergeleri yerine getirebilirler. Örnek olarak bilgisayardan bir gün içerisindeki ortalama sıcaklığı hesaplamasını istersek gerçekte burada kullanılan sıcaklık, günün kaç saat olduğu gibi veriler bilgisayar tarafından bilinmemektedir. Bu veri, bilgisayar açısından pek bir şey ifade etmeyeceği gibi bilgisayara yönergeler verilerek bir dizi işlem yaptırılabilir.

- Sıcaklığı temsil eden bir sayıyı kabul etmek,
- Günün kaç saat olduğunu temsil eden bir sayıyı (24) kabul etmek,
- Gün içindeki saatlik sıcaklıkları toplamak,
- Toplam sıcaklık değerini 24'e bölmek
- Sonucu görüntülemek.

Bu şekilde basit bir akış oluşturulabilir. Ancak bu şekilde bir işlem dizisi bilgisayar için anlamsız olabilir.

1.2. Doğal Diller ve Programlama Dilleri

Dil, düşünceleri ifade etmek ve kaydetmek için bir araçtır. Her toplumun kullandığı dil, birbirinden farklı olduğu gibi toplum içinde de dil bölgelere göre farklı biçimlerde kullanılabilir.

Bilgisayarın anladığı gerçek dil, 0 ve 1'lerden ibarettir. Bilgisayar bizim kendisine verdiğimiz komutları makine diline çevirerek icra eder.

Gerçekte bilgisayarları insanlar programlamaktadır. Programcıların direkt makine dilinde program yazmaları çok zahmetli ve uzun bir süreçtir. Ayrıca bilgisayar donanımındaki hızlı gelişim, yazılım geliştirme sürecinde daha pratik yöntemlerin gelişmesini gerekli kılmıştır. Bu gelişmeler yazılım geliştiren bireyler için programlama dillerinin ortaya çıkmasına neden olmuştur. Böylece yazılım geliştiricilere, kendi dillerine daha yakın ve hızlı geliştirme yapmaları için imkân tanınmıştır. Programlama dillerinde yazılan komutlar, direkt olarak bilgisayarın anlayabileceği komut yazma işlemlerini derleyici veya yorumlayıcılara bırakmışlardır.

C, Java, Python ve Pascal gibi programlama dilleri çıkıp, insan diline benzer yapıda yazılım geliştirme olanağı sağlayınca, yazılım geliştirme işi kolaylaşmış ve yazılımcı sayısında da artış görülmüştür. Python dili de İngilizce düşünen ve konuşan insanlar için neredeyse biriyle konuşuyormuşçasına yazılım geliştirme imkânı tanımıştır.

Makine dilinde 0 ve 1'lerin herhangi bir işleme tabi tutulmadan işlem yapılması, muhtemelen çok zor bir süreç olurdu. Makine dili; **merkezi işlem birimi (CPU)** tarafından anlaşılıp çalıştırılabilen komutlardan oluşan program yazma aracıdır. Makine dili üzerinden herhangi bir işlem ya da dönüştürmeye gerek kalmadan bilgisayar tarafından doğrudan anlaşılır. Bu dil, sadece 0 ve 1 sayılarından oluşan, ikili kodların anlamlı kombinasyonlarından meydana gelmektedir.

Makine dili insanlar tarafından oluşturulmuştur. Ancak 0 ve 1'lere bakarak bir çıkarımda bulunmak zordur. Bilgisayara verilen komutların bilgisayar tarafından anlaşılması için bir dizi işlemden geçmesi gerekmektedir:

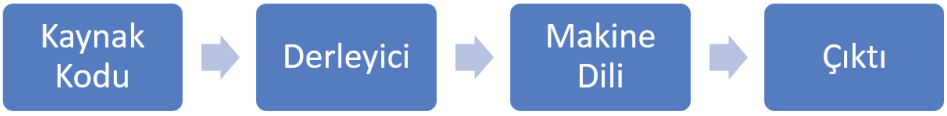
1. **Fetch** Bir sonraki görev hafızadan getirilir.
2. **Decode** Görevin ne demek istediği deşifre edilir.
3. **Execute** Görev yerine getirilir ve işlem gerçekleştirilir.
4. **Stop** Sonuç saklanır.

1.3. Derleyici ve Yorumlayıcılar

Makine dili, düşük seviyeli bir programlama dilidir ve bu programlama dili ile bilgisayara komutlar vermek, işlem yaptırmak zordur. Kullanıcıların kolaylıkla program yazmaları için yüksek seviyeli programlama dillerini kullanmaları gerekir. Yüksek seviyeli dil ile yazılmış olan programa **kaynak kodu** denir. Kaynak kodunun bilgisayar tarafından anlamlandırılabilmesi için makine koduna dönüştürülmesi gerekir. İşte, bu iş ya **derleyici (compiler)** ya da **yorumlayıcı** (interpreter) tarafından gerçekleştirilir.

Derleyici (compiler), belirli bir programlama dilinde yazılmış ifadeleri işleyen ve bunları bir bilgisayar işlemcisinin kullandığı makine diline veya "koda" dönüştüren özel bir programdır.

Derleyici, kaynak kodu derlemeden önce kodları kontrol eder. Olası yazım hataları veya dilin yapısına uygun olmayan kalıplar varsa kullanıcıya çıktı olarak bildirir. Bu sayede kullanıcı, **compile** işlemi esnasında hataları ayıklar.



Şekil 1.1: *Derleyici işlem adımları*

Yorumlayıcılar (interpreter), programı okuma ve yürütme yeteneğine sahip sistem yazılımı olarak adlandırılabilir. Yorumlayıcı programı satır satır yorumlar, bu işlem yukarıdan aşağı doğru yapılır ve kullanıcıya bildirilir. Daha alt satırlarda hata varsa bu hatalar bulunamaz çünkü satır satır işlem yapılmaktadır.

Şekil 1.2: *Yorumlayıcı işlem adımları*

1.4. Derleyici – Yorumlayıcı Farkı

Derleyici	Yorumlayıcı
■ Tüm programı bir bütün olarak makine diline çevirir. ■ Makine diline çevrilmiş bir programın çalışması daha hızlıdır. ■ Daha fazla bellek gerektirir. ■ Tüm hatalar belirlenerek bir seferde çıktı verir. Hata ayıklamak zordur. ■ C, C++ gibi diller derleyici kullanır. ■ Exe dosyaları derlenmiş uygulamaya örnek olarak verilebilir.	■ Programın bazen tek satırını bazen tek ifadesini alıp işler, satır satır işlem görür. ■ Daha yavaştır. ■ Hafızada fazla yer kaplamaz. ■ Gördüğü ilk hatada çalışmayı durdurur. Hata ayıklamak daha kolaydır. ■ Python, Ruby, Java dilleri yorumlayıcı kullanır.

1.5. Python Nedir?

Python, 90'lı yılların başında Amsterdam'da Guido Van Rossum tarafından geliştirilmeye başlanan bir programlama dilidir. Python yazılım geliştirme ve veri analizinde ön plana çıkmıştır. Python'un standart kütüphanesi; geliştirme araçları diğer birçok kütüphanesi açık kaynak kod olarak ücretsiz şekilde indirilebilmektedir.

Python nesne yönelimli, yorumlanabilen ve yüksek seviyeli bir programlama dilidir.

1.6. Python Sürümleri

Python 2 ve Python 3 olarak adlandırılan iki ana Python dağıtımı bulunmaktadır. Python 2, Python'un eski sürümüdür. En son 2011 yılında 2.72 sürümü yayınlanmış ve gelişimi durmuştur.

Kitap yazım sürecinde Python Vakfının kendi sitesi olarak https://www.python.org/ sitesinde son sürüm olarak 3.82 versiyonu yayınlanmaktadır. Ayrıca Vakıf düzenli aralıklarla güncelleme yapmaktadır.

Python 2 ile yazılan komut dosyaları Python 3 yorumlayıcısı tarafından çalıştırılamaz. Kodlardaki pek çok uyumsuzluk üzerinde düzenleme yapılması gerekmektedir

Python 3, sadece Python 2'den daha iyi bir versiyon değil, öncekine çok benzemesine rağmen tamamen farklı bir dildir.

Bu kitaptaki uygulamalar ve kodlar Python 3 diline göre hazırlanmıştır. Python 3.4, python 3.5, Python 3.6 ve Python 3.7 sürümlerinde yazılmış kodların tamamı Python 3.82 sürümünde çalışmaktadır.

1.7. Neden Python?

Python, popülerliğini hızlı bir şekilde yükseltmektedir. İlk defa programlamaya başlangıç yapanlar ya da farklı dillerde uzmanlaşanlar "Python öğrenmeli miyim?" sorusunu sormaktadır.

Uzun yıllar önce programlama öğrenmek isteyenler için bir programlama dilini öğrenmek veya aşina olmak şimdikinden çok daha zordu. Ancak zamanla insan diline yakın denilebilecek **yüksek seviyeli** programlama dilleri ortaya çıkmıştır.

Python kodlarını yorumlamak ve öğrenmek diğer dillere göre daha kolaydır. Diğer dillerde bulunan noktalama işareti zorunlulukları, parantezler veya kurallar programlamaya yeni başlayan kullanıcı için zaman zaman zorluklar çıkarmaktadır. Ancak Python'da bu tür zorunluluklar olmadığı gibi yapısı itibariyle diğer dillere göre daha sadedir. Ayrıca Python, yıllar içerisinde belirli bir olgunluğa, geliştirici topluluğuna ve öğretici dokümana sahip olmuştur. Python dili ile program geliştirirken karşılaşmanızın muhtemel olduğu birçok sorunun cevabı stackoverflow gibi sitelerde bulunmaktadır. Bu da Python'u öğrenirken hızlı ilerlemenize olanak sağlamaktadır. Python, web geliştirme uygulamalarında kullanılabileceği gibi, kullanıcı ara yüzüne sahip (GUI) PyQt gibi kütüphanelere de sahiptir.

Python'un veri bilimi, veri analizi ve yapay zekâ ile ilgili gelişmiş kütüphaneleri olduğu için bu alanda ilerlemek isteyen kişilerin en çok kullandığı programlama dillerinin başında Python gelmektedir.

1.8. Python'u İndirme ve Yükleme

Python ile yazılım geliştirmek için kullanabileceğiniz pek çok miktarda araç mevcuttur:

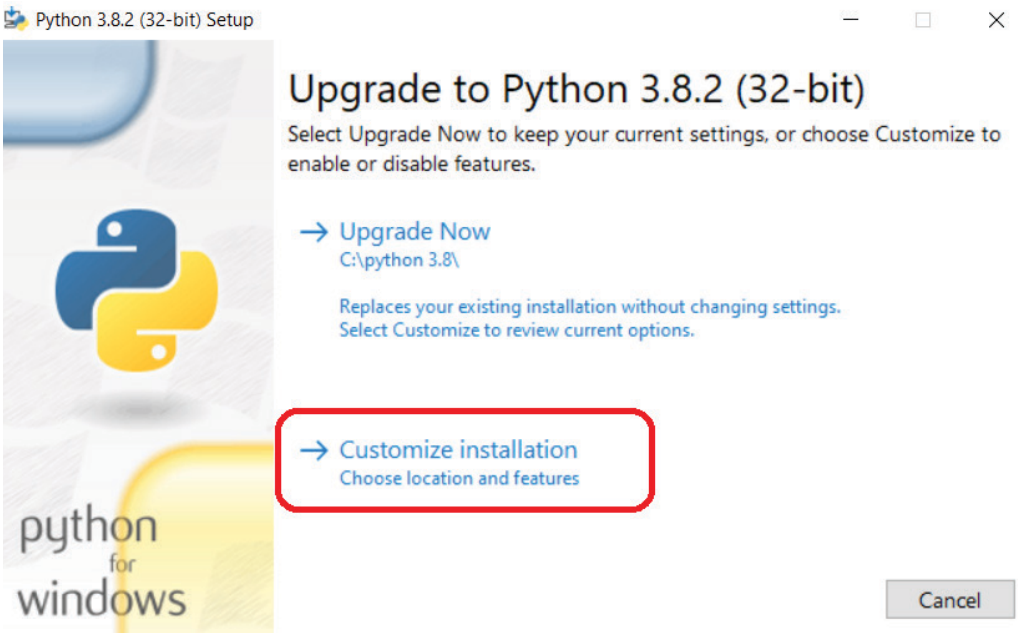
- anaconda
- jupyter notebook
- ipython
- orange
- spyder
- visual studio code
- pycharm

Kitapta Python'un resmî geliştirme ortamı kullanılacaktır. Bunun nedeni, kendi geliştirme ortamının (Idle) sade ve kolay olmasından dolayı ve güçlü özellikte bilgisayar gerektirmemesidir.

Python programını https://www.python.org/ resmî sitesinden en son sürümünü indirip kurulması gerekmektedir. Kitap yazım aşamasında son versiyon olarak 3.82 sürümü olduğu için bu sürüm üzerinden Python programlama dili anlatılacaktır. Python programı güncellendikçe yeni sürümler indirilip kurulabilir. Site üzerinde farklı işletim sistemleri için kurulum paketi bulunmaktadır. Kullanıcı, sistemine en uygun paketi indirerek kişisel bilgisayarına kurulumu gerçekleştirebilir.

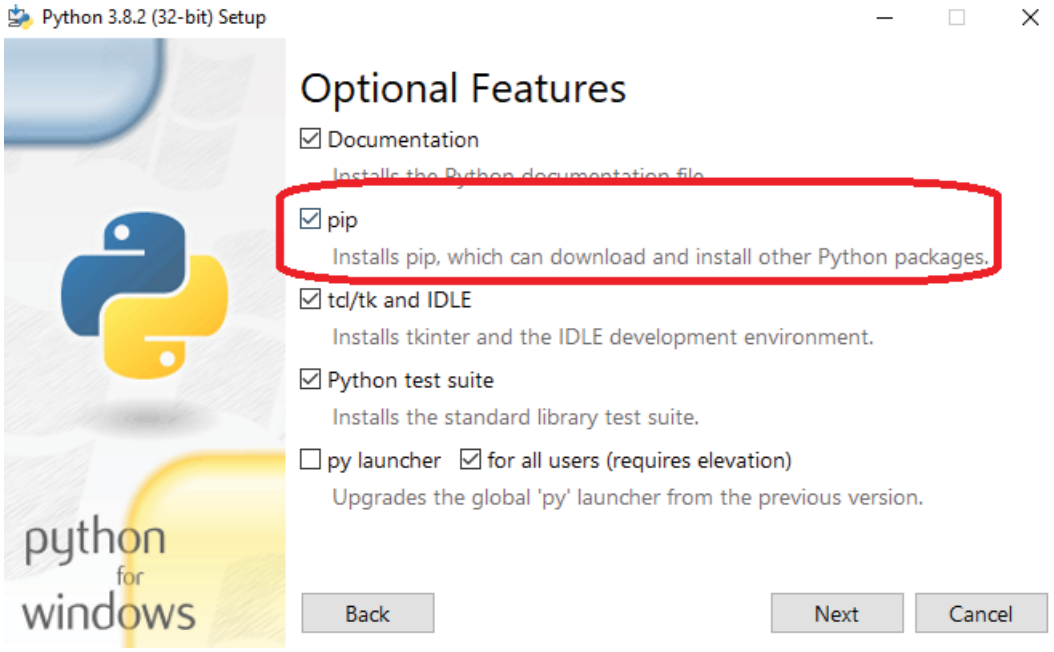
Şekil 1.3: Python resmi sitesi

Kurulum ekranında **Customize Installation** seçeneği seçilmelidir. Bu sayede Python'un kurulacağı klasör seçilebilir.



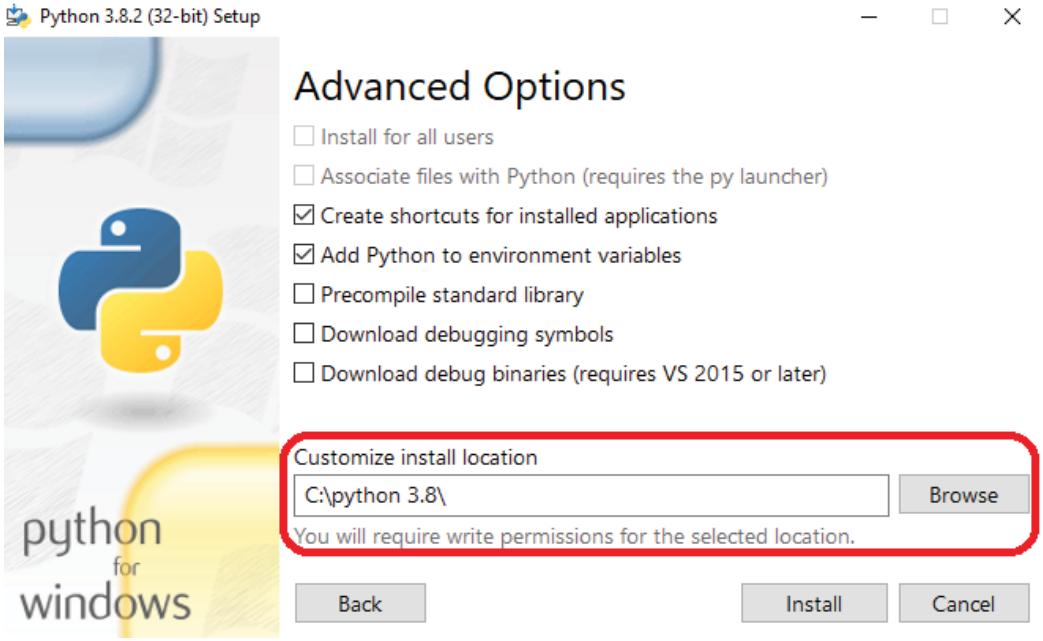
Şekil 1.4: *İlk kurulum ekranı*

Kurulum ekranının ikinci sayfasında **pip** (paket yükleyici) seçeneği mutlaka seçilmeli, bu sayede ileride Python'a yeni kütüphanelerin eklenmesi sağlanabilir.



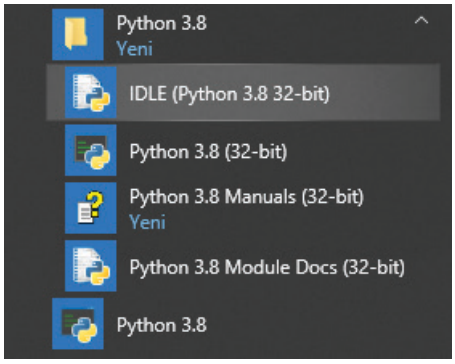
Şekil 1.5: *pip paket yükleyicisinin seçilmesi*

Kurulum ekranının 3. sayfasında, Python'un kurulacağı konum seçilmelidir. Eğer kurulum sihirbazı farklı bir konum gösteriyorsa değiştirilerek **c:\python 3.8** şeklinde erişilebilir bir konuma kaydedilmesinde fayda vardır.



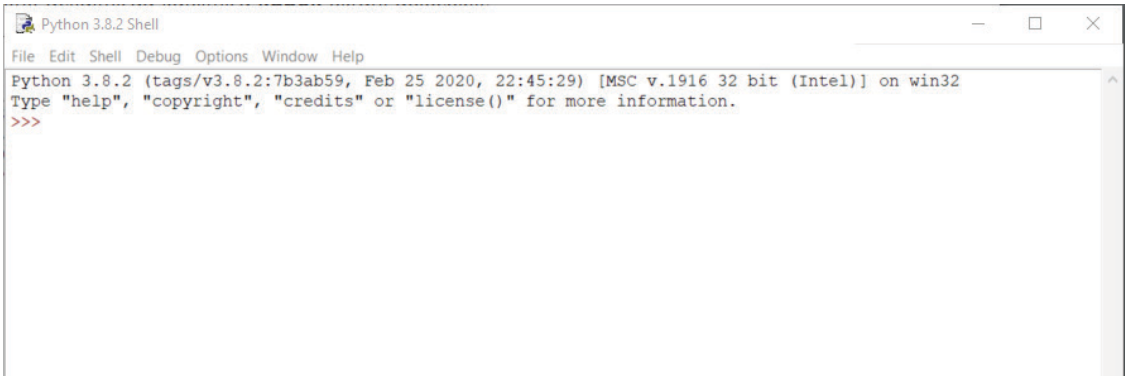
Şekil 1.6: *Kurulum klasörünün seçilmesi*

Kurulum tamamlandığında Python editörünü açmak için başlat menüsünden **IDLE** uygulaması tıklanarak Python editörü açılmalıdır.



Şekil 1.7: *Windows başlat menüsü Python uygulaması*

IDLE editörünü açtığımızda karşımıza **kabuk** ekranı gelecektir.



Şekil 1.8: *Etkileşimli kabuk*

Kabuk ekranı üzerinde çeşitli aritmetiksel ve mantıksal işlemler yapılabileceği gibi, basit ve birkaç satırlık kodlarımızı burada çalıştırarak uygulamanın vereceği olası çıktılar da gözlemlenebilir.

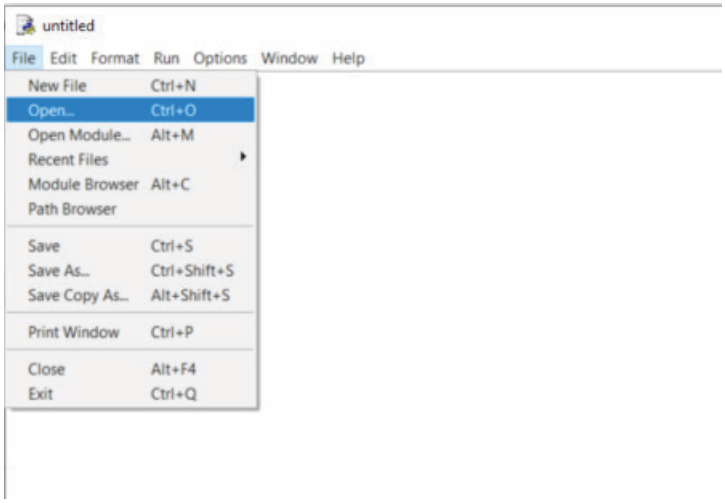
Yeni bir Python dosyası oluşturmak ve kod yazıp çalıştırmak için, **File** menüsünden **New File** seçeneği seçilmelidir.

Şekil 1.9: *Yeni Python dosyasının oluşturulması*

Açılan **Not Defteri** benzeri pencerede kodlar yazılarak çalıştırılabilir.

Şekil 1.10: *İlk kodlarımızı yazalım*

Yazılan Python kodlarının çalıştırılması için kaydedilmesi gereklidir. Yeni bir pencere açılarak kodlar yazılabileceği gibi daha önceden oluşturulan programlar da çağrılabilir.



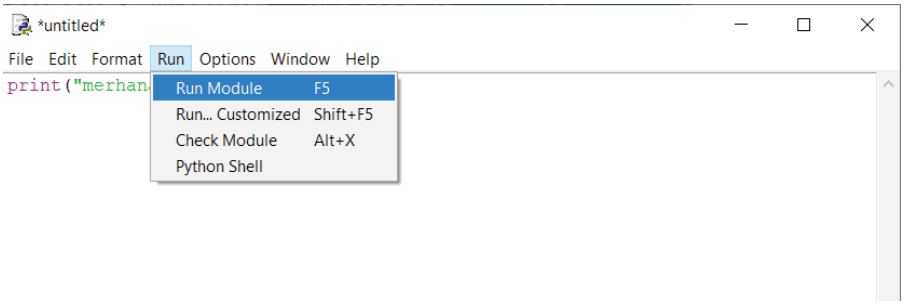
Şekil 1.11: *Daha önceden oluşturulan dosyaların çağrılması*

File menüsünde bulunan **Open** seçeneği ile daha önceden oluşturulmuş bir dosya yüklenebilir. Dosyayı kaydetmek için Save – Save As.. seçeneği seçilerek kaydedilecek dosyanın konumu seçilmeli, sonrasında dosyaya bir isim verilmelidir.

1.9. İlk Kodlarımızı Yazalım

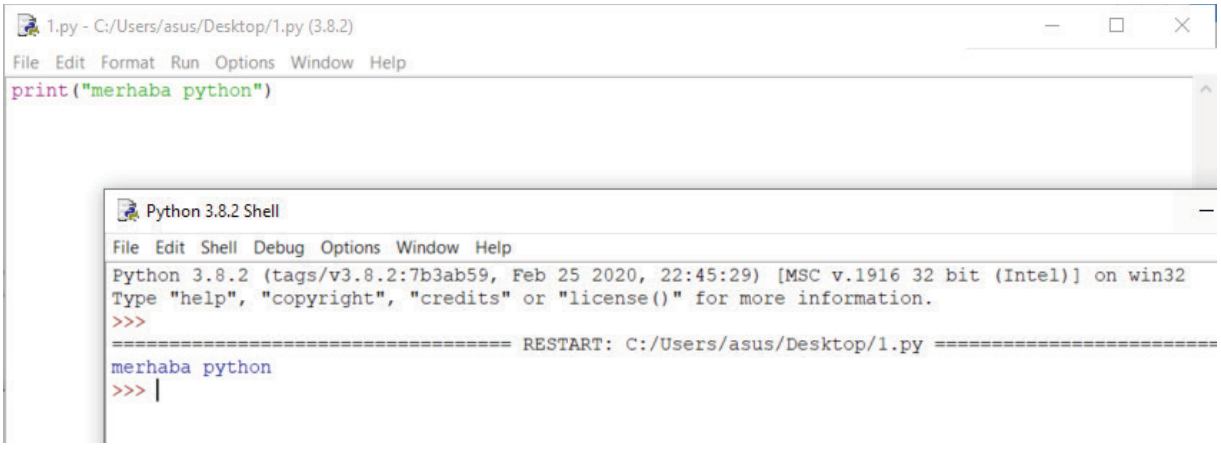
Bu aşamada ilk kodları yazılarak Python'un nasıl çalıştığı gözlemlenebilir.

Python editörü üzerinde **print("merhaba python")** şeklinde kodlar yazılmalıdır. Dosyayı çalıştırmak için **Run** menüsünde bulunan **Run Module** seçeneği seçilmeli ya da klavyeden **F5** tuşuna basılmalıdır. Eğer dosya kaydedilmemişse dosyanın kaydedilmesi için **Save – Save As** penceresi açılacak ve dosyayı kaydedebilmek için konum ve dosya adı girilmesi istenecektir.



Şekil 1.12: Uygulamanın çalıştırılması

Uygulamayı çalıştırıldığında **Etkileşimli Kabuk** penceresi açılacak ve yazılan kodların çıktısı etkileşimli kabuk üzerinde görüntülenecektir.

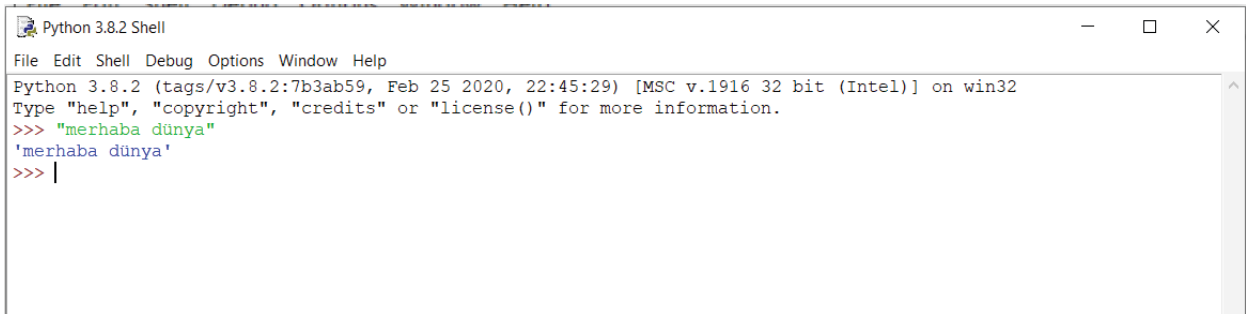


Şekil 1.13: Uygulamanın çıktısı

1.10. Etkileşimli Python Kabuğu

Etkileşimli kabuk kullanılarak, Python kodları kaydedilmeye gerek kalmadan, doğrudan konsol benzeri bir yazılım üzerinde denenebilir. **Etkileşimli kabuk (Interactive Shell)** yazılan komutun sonucunu çıktı olarak vererek kullanıcının dili öğrenmesini kolaylaştırır, uygulamaların denenerek çıktı alınmasına olanak sağlar.

Bunun yanında etkileşimli kabuk üzerinde çalışırken print fonksiyonu kullanmaya gerek kalmadan anlık olarak yapılan işlemlerin çıktısı görüntülenebilir. Kitabın ilk bölümlerinde yapılan uygulamalar, etkileşimli kabuk üzerinde yapılarak, Python programlama diline daha kolay bir şekilde adapte olunması sağlanacaktır.



Şekil 1.14: Etkileşimli Python kabuğu

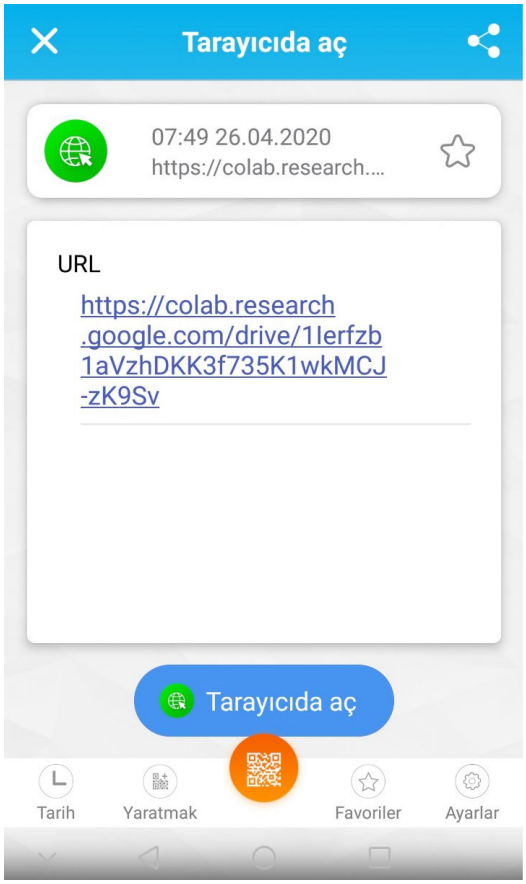
1.11. Google Colab Kullanımı ve Kitaptaki Uygulamalara Erişim

Kitaptaki kod örneklerine kare kod okutarak erişilebilir. Örnekler internet üzerinde sanal bir Python makinesi oluşturan (aslında Jupyter Notebook) Google Colab yardımıyla (bir Google hesabınız olduğunu varsayarak) kolay bir şekilde oluşturulabilir. Bilgisayar veya telefonda herhangi bir program kurulmadan Google Colab ile Python kodları yazılabilir. Google Colaboratory'e https://colab.research.google.com/ adresinden erişilebilir.

Örnek kodların nasıl çalıştırılacağı hakkında bilgi edinmek için aşağıdaki yönergeler izlenmelidir.

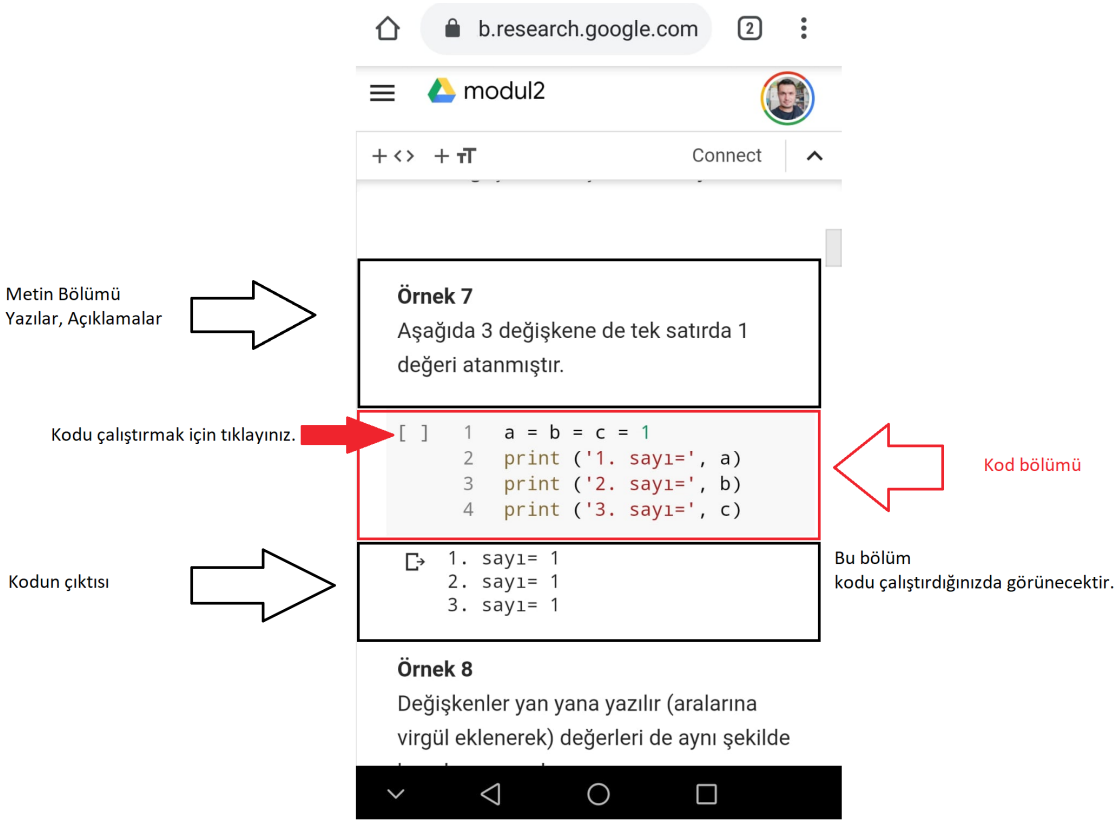
1. Öncelikle bir Google hesabı gerekmektedir. Google hesabı yoksa veya kodlar yerel bilgisayarda çalıştırmak istenirse (py uzantılı dosyalar) sayfanın altında bulunan kare kod kullanılmalıdır.

2. Akıllı cihazınızdan **kare kod okuma uygulaması** açılmalıdır.
3. Bölümler için ayrı ayrı oluşturulmuş kare kodlardan istenilen bir tanesi okutulabilir. **Linki aç, tarayıcıda aç** tuşuna basılmalıdır.



Şekil 1.15: *Mobil cihazdan karekod okuyucu ile kodun okutulması*

4. Telefonda Google oturumu zaten açık olacağı için otomatik olarak Google Colab üzerinden örnek kodlara erişilebilir.

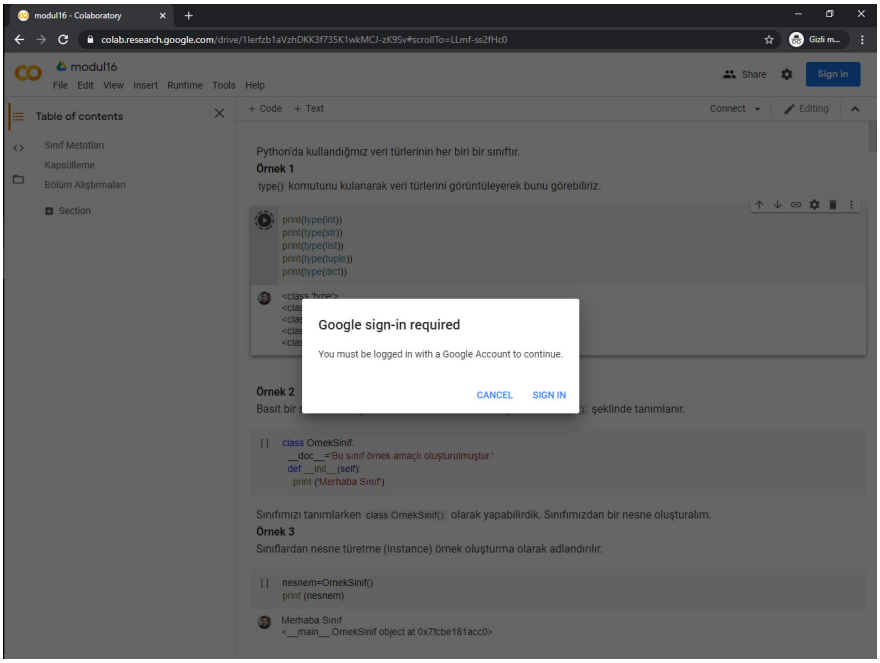


Şekil 1.16: *Mobil cihazdan Google Colab'a erişim*

Herhangi bir kodu çalıştırmak için kod bloğuna gelerek [ ] tuşuna basılması yeterlidir.

Şekil 1.17: *Butona basılınca kodun çıktısı görüntülenecektir.*

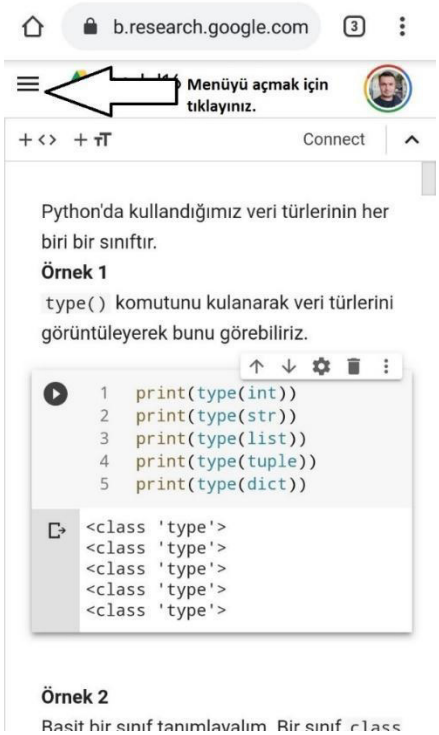
5. **Kodu çalıştır**'a tıklandığında aşağıdaki gibi bir hata mesaj alınır. **SIGN IN**'e basılarak Google hesabına giriş yapılmalıdır.



Şekil 1.18: *Google hesabına giriş ekranı*

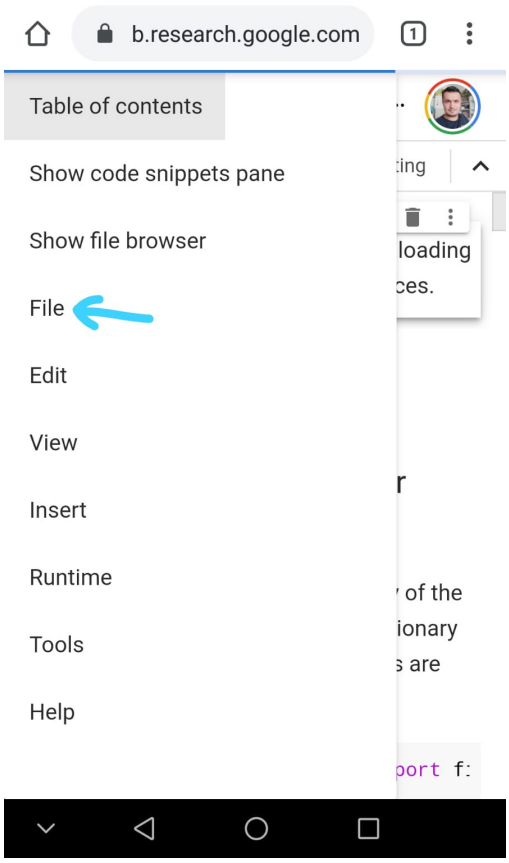
6. Program kodunu ilk defa çalıştırırken güvenlikle ilgili bir uyarı mesajı gelirse mesaj okunduktan sonra **Run Anyway** tuşuna basılmalıdır.

7. İstendiği takdirde kodlar düzenlenebilir, notlar eklenebilir ve değişiklik yapılabilir. Kodlar bilgisayara indirilebilir ve/veya Google Drive'a kaydedilerek yine Google Colab üzerinden çalışmaya bilgisayardan devam edilebilir. Bunun için telefonda Google Colab menüsüne erişilmelidir. Sayfanın sol başında bulunan menü işaretine tıklanmalıdır.



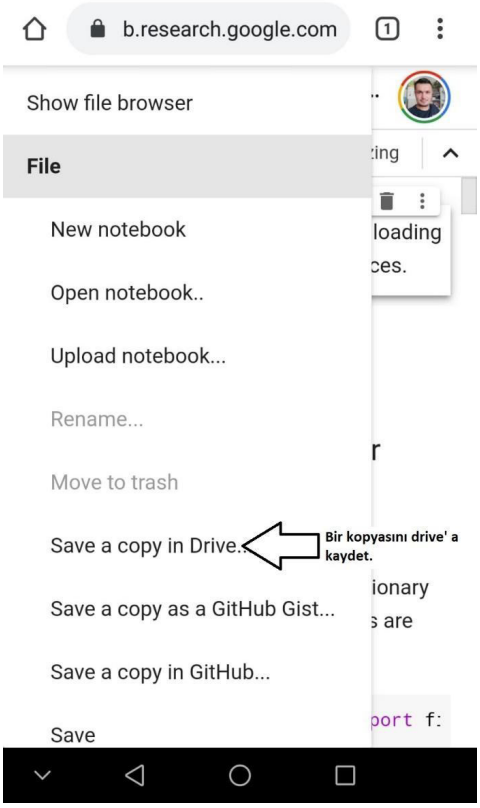
Şekil 1.19: Google hesabına geçiş

8. Daha sonra **File** menüsü tıklanmalıdır.



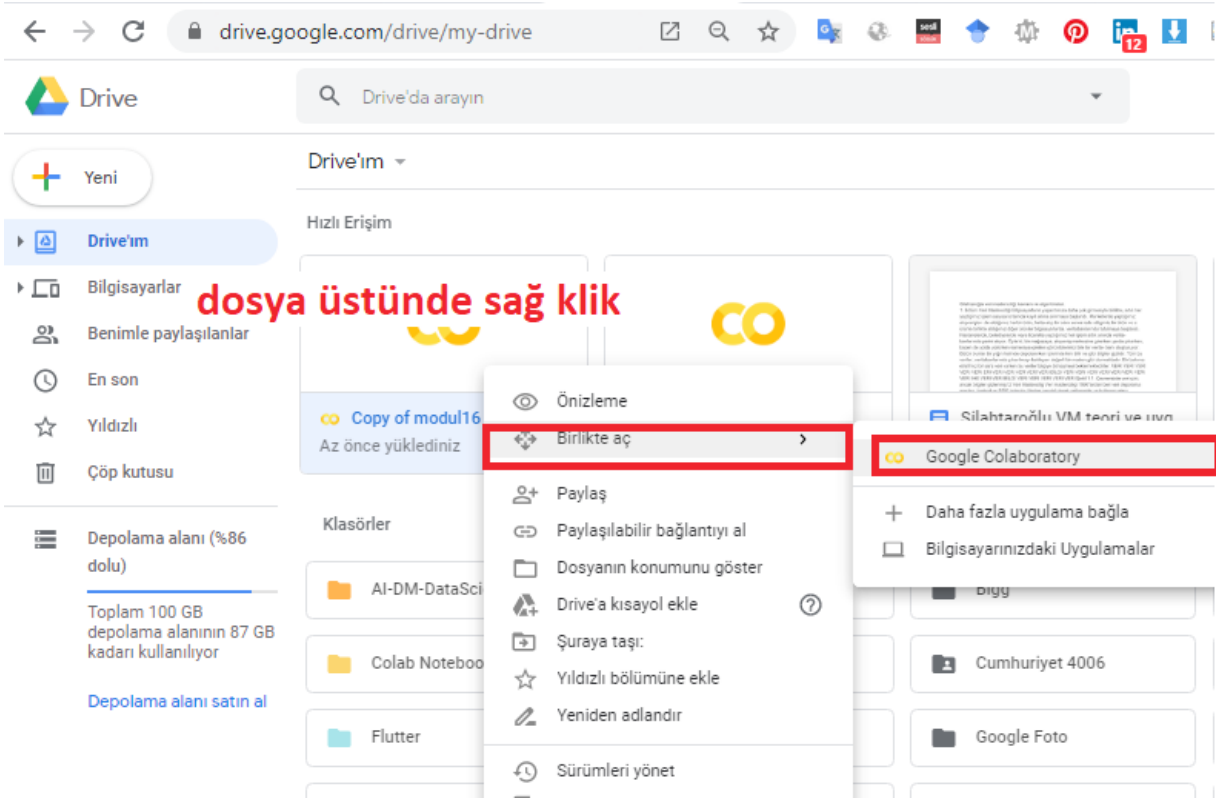
Şekil 1.20: Google hesabına geçiş adımları

9. **Save a copy in Drive**'a basılarak, dosyanın bir kopyası kullanıcının drive hesabı içinde oluşturulup, oluşturulan kopya açılarak çalışma, kullanıcının kendi dosyası üzerinden devam edebilmesini sağlar.



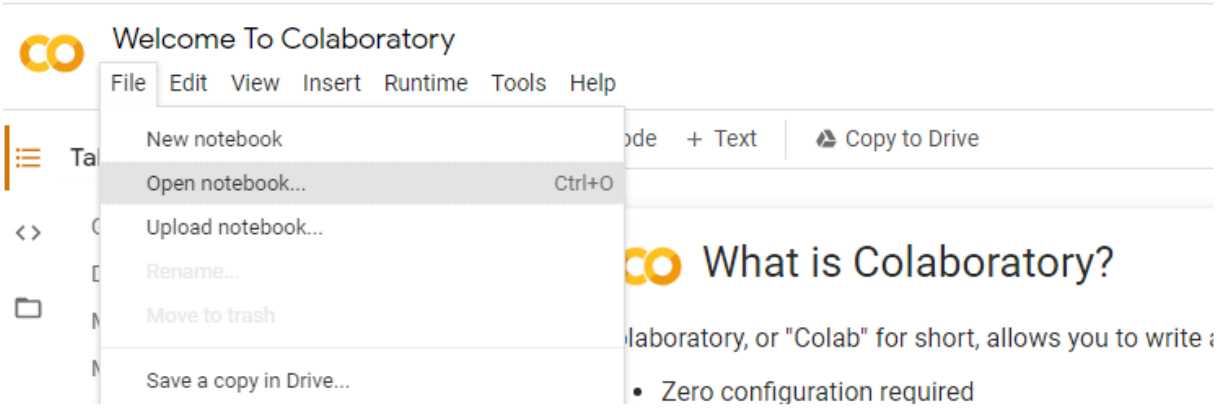
Şekil 1.21: Uygulama kodlarının kendi hesabımıza aktarılması

10. Bu aşamadan sonra bilgisayardan Drive üzerinden, dosyaya sağ tıklanarak birlikte aç Google Colab seçilip çalışmaya devam edilebilir.

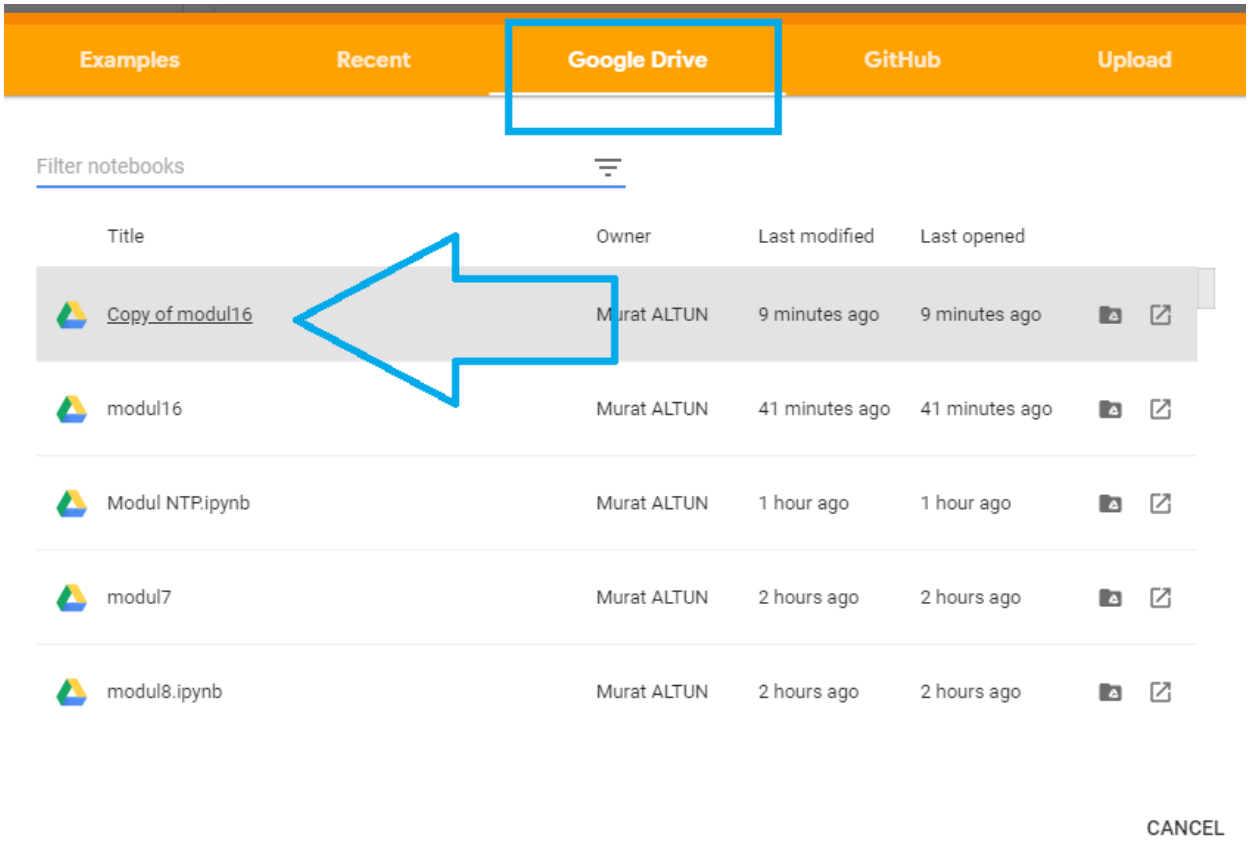


Şekil 1.22: Kodların Google Colab ile açılması

11. Başka bir yöntem ise https://colab.research.google.com/ adresine **File** menüsü altında **open notebook** seçildikten sonra Google Drive seçilerek ve ilgili dosyaya tıklanmalıdır.

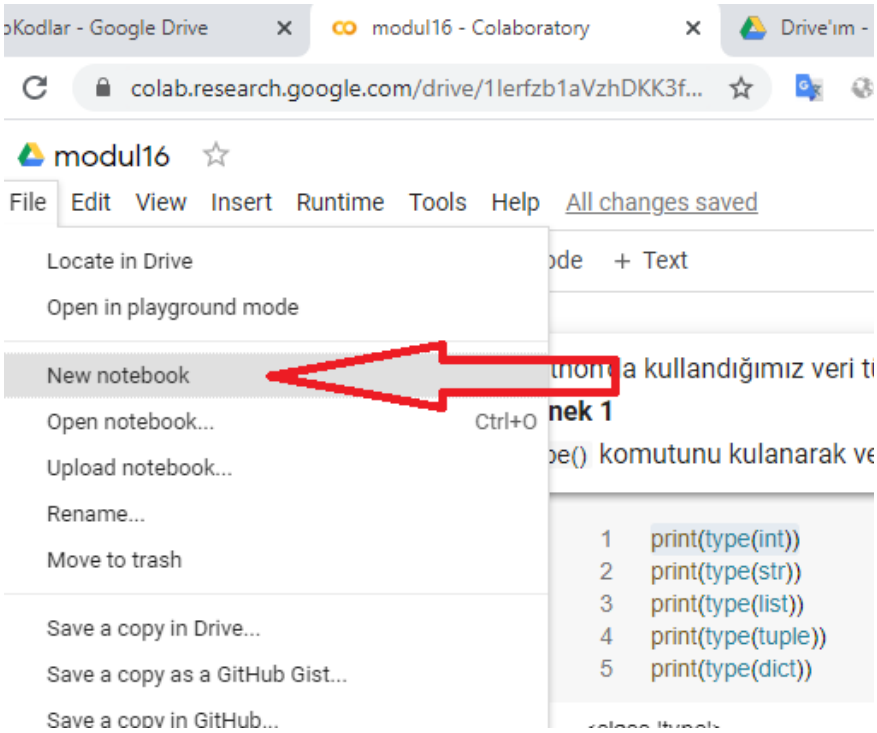


Şekil 1.23: *Google Colab açmak için alternatif yöntem*



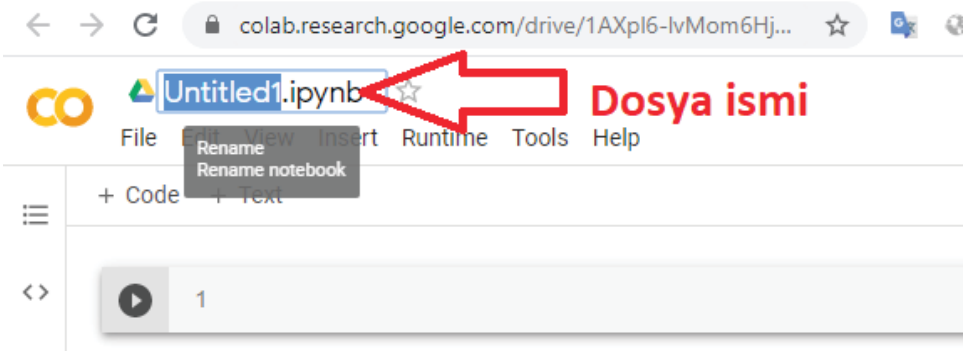
Şekil 1.24: *Uygulama kodlarının kullanıcının Google hesabına*

12. Tüm bu süreçlerden bağımsız olarak https://colab.research.google.com/ adresine yeni bir notebook oluşturularak, Python kodları kolayca denenebilir.



Şekil 1.25: *Boş bir notebook dosyasının oluşturulması*

13. Yeni notebook açtıktan sonra ismine tıklanarak değiştirilebilir. Değişiklikler otomatik olarak kaydedilmektedir. Notebook daha sonra kaydedilen isimle, Google Drive üzerinde Colab Notebooks klasörü içinden erişilebilir.



Şekil 1.26: *Notebook dosyasının isminin değiştirilmesi*

Sorular ve ayrıntılı yardım için:

Şekil 1.27: *Yardım için karekodu okutun.*

MODÜL 2

YORUM SATIRLARI, DEĞİŞKENLER, VERİ TİPLERİ VE OPERATÖRLER

Şekil 2.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

2.1. Yorum Satırlarını Kullanma

Yorum satırları, Python yorumlayıcısı tarafından dikkate alınmayan ve yorumlanmayan ifadelerdir. Python'da yorum satırları genel olarak aşağıdaki işlemler için kullanılır:

- Bir hatırlatma eklemek
- Program veya kodlarla ilgili bir açıklama yapmak
- Kullanılmayan bir kod satırını pasif hale getirmek
- Süsleme yapmak

Bu tür açıklama satırları kodun başkaları tarafından daha iyi anlaşılmasını sağlar. Python'da tek satırlık açıklama için "**#**" işareti kullanılır. "#" işareti kullandığınızda o satırdaki metin kod olarak işlenmez.

Örnek 1

Aşağıdaki yorum satırları Python tarafından dikkate alınmamaktadır. Bu nedenle sadece "print()" komutu çalışır.

```
#Bu kod ekrana yazı yazılmasını sağlamaktadır.
print ("Konu: Yorum satırlarını kullanma")
#Her satırın başına # işareti eklenerek
#alt alta yorum satırları oluşturulabilir.
Konu: Yorum satırlarını kullanma
```

Örnek 2

Yorum satırlarını kod satırının devamında aynı satırda kullanabilirsiniz. Bu kullanımda önce kod gelir, devamında yorum satırı "#" işareti ile başlar (öndeki kod çalışır), daha sonra satır sonuna kadar yorum satırı olarak dikkate alınmaz.

```
print (2+3) # Bu kod satırı ekrana 2 sayının toplamını yazar.
5
```

Her satırın başına "#" işareti ekleyerek alt alta yorum satırları oluşturabilirsiniz. Yorum satırlarında özel karakterler etkisizdir.

Örnek 3

Birden fazla yorum satırı kullanılacaksa yorumlar üçlü tek tırnak veya üçlü çift tırnak blokları arasına yazılır.

```
'''Python'da birden fazla açıklama satırı kullanmak için üçlü tek tırnak veya çift
tırnak kullanılır. Açıklama satırını bitirmek için aynı işaretler kullanılır.'''
'Python'da birden fazla\naçıklama satırı\nkullanmak için üçlü tek tırnak veya çift\n
tırnak kullanılır. Açıklama satırını bitirmek için \naynı şekilde kullanılır.'
```

Konsol çıktısında açıklamalar birden fazla satırda olduğu için satır sonlarına \n kaçış dizisi karakterini konsol otomatik olarak eklemiştir. Aynı yorum satırını """yorum satırları""" üçlü çift tırnak kullanarak da yapılabilir. Yorum satırları içinde kaçış dizisi karakterlerinin de çalışmadığı unutulmamalıdır.

Örnek 4

Yorum satırlarını Python yorumlayıcı dikkate almaz. Ancak aşağıdaki kod satırları yorum satırı olarak değerlendirilmez.

```
#!/usr/bin/env python3 veya #!c:/Python/python.exe
# -*- coding: utf-8 -*-
"#!/usr/bin/env python3" kodu Python 3 yorumlayıcısının Linux için dosya konumunu
belirtir.
"#!c:/Python/python.exe" kodu Python 3 yorumlayıcısının Windows için dosya konumunu
belirtir.
"# -*- coding: utf-8 -*- " Kullanacağınız karakter kodlamasını belirtmek için
kullanılır. utf-8 Türkçe alfabeyi de destekleyen bir karakter kodlama sistemidir.
```

Örnek 5

Yorum satırları süsleme amacıyla da kullanılabilir. Bazı program dosyalarında aşağıdaki gibi süslü açıklamalar, etiketler görülebilir.

```
'''
#########################################################################
#*********************************************************************##
#                        Python Öğreniyorum                           ##
#                             Python 3                                ##
#*********************************************************************##
#########################################################################
'''
```

2.2. Değişkenler

Kod yazarken sadece sabit değerler üzerinden işlemler yapılmaz. Kullanıcıdan veya başka kaynaklardan veri alınması gerekir. Örneğin, kullanıcıya ismiyle "merhaba" diyeceğimiz bir kod yazmak istersek her kullanıcıdan ismini girdi olarak almamız gerekir. İşlem yapabilmek için bu girdiler (değerler) bellekte tutulmalıdır. Bu girdileri depolamak amacıyla bellekte belirli bir yer ayrılması gerekir. Bir değişken tanımlandığında yorumlayıcı, veri türüne bağlı olarak bellekte yer ayırır ve ayrılan bölümde hangi türden verilerin saklanabileceğini belirler. Değişkenler bir isim, sayı veya farklı türdeki bir veri için bellekte ayrılan bu yeri temsil eder. Değişkenlere farklı veri tiplerinde değerler atanabilir. Değer atama ile (bir değişkeni bir değere eşleyerek) tam sayı, ondalık sayı, dizi veya karakter dizisi türünde değerler değişkenlerde tutulabilir. Python, bu konuda çok esnektir. Python'da değişkenlerinin veri tiplerini açıkça bildirmeye gerek yoktur. Aynı değişken farklı veri tiplerinde değerler alabilir. Aynı değişkene önce sayı, sonra bir metin daha sonra başka türde bir değer atanabilir. Bir değişkene değer atandığında veri tipi otomatik olarak tanımlanır. Eşittir operatörü "=" değişkenlere değer atamak için kullanılır. "=" Operatörünün solunda değişkenin adı ve "=" operatörünün sağında ise bu değişkene atanacak değer yer alır.

Örnek 6

Değişken oluşturma ve değer atamaya ilişkin örnekler:

```
#sayi1 değişkenine 5 sayısı atandı.
sayi1=5
print('Değişkenin içindeki sayı: ', sayi1)
sayi1=10
print('Değişkenin içindeki sayı: ', sayi1, 'oldu')
sayi1='Murat'
print ('Değişkenin içindeki değer: ', sayi1, 'oldu')
sayi1=10.5
print ('Değişkenin içindeki sayı: ', sayi1, 'oldu')
Değişkenin içindeki sayı:  5
Değişkenin içindeki sayı:  10 oldu
Değişkenin içindeki değer:  Murat oldu
Değişkenin içindeki sayı:  10.5 oldu
```

Örnekte aynı değişkene hem karakter dizisi (string) hem de sayısal değerler atanmıştır.

Bir değişkene değer atanırken birden fazla yöntem kullanılabilir. Örnekte buna ilişkin kodlar verilmiştir.

Örnek 7

Aşağıda 3 değişkene de tek satırda 1 değeri atanmıştır.

```
a = b = c = 1
print ('1. sayı=', a)
print ('2. sayı=', b)
print ('3. sayı=', c)
1. sayı= 1
2. sayı= 1
3. sayı= 1
```

Örnek 8

Değişkenler aralarına virgül eklenerek yan yana yazılır. Değerleri de aynı sıralama ile karşılarına yazılır.

```
adi, soyadi, yasi='Canan', 'DAĞDEVİREN', 34
print ("Adı=", adi)
print ("Soyadı=", soyadi,)
print ("Yaşı=", yasi)
Adı= Canan
Soyadı= DAĞDEVİREN
Yaşı= 34
```

Örnek 9

Değişkenlere değer atamak için başka bir yöntem aralarına noktalı virgül ";" ekleyerek değişken - değer ikilileri şeklinde yazmaktır.

```
adi='Aziz'; soyadi='SANCAR'; yasi=73
print ("Adı=", adi)
print ("Soyadı=", soyadi,)
print ("Yaşı=", yasi)
Adı= Aziz
Soyadı= SANCAR
Yaşı= 73
```

Örnek 10

Değer atanmayan ve/veya tanımlanmamış bir değişken kullanılırsa Python hata verir.

```
print (yenisayi)
#Python değişkenleri değer atandığında tanımlandığı için hata mesajı alırsınız.
# yenisayi değişkeni tanımlanmamış olduğu için hata mesajı alınır.
NameError                 Traceback (most recent call last)
<ipython-input-1-9ff08615337f> in <module>()
----> 1 print (yenisayi)

NameError: name 'yenisayi' is not defined
```

Değer atamadan tanımlamak için yenisayi=int( ) kodu kullanılabilir. Bu durumda değişkene ilk değer olarak "0" sıfır atanır.

```
yenisayi=int( )
print(yenisayi)
0
```

Örnek 11

Değişkenler veri tiplerine göre kullanılmazsa Python hata verir.

```
sayi1=1
metin1='deneme'
print(sayi1+metin1) #Bir sayı ile bir metin, kelime toplanamaz.
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-e585c633881d> in <module>()
      1 sayi1=1
      2 metin1='deneme'
----> 3 print(sayi1+metin1) #Bir sayı ile bir metin, kelime toplanamaz.

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

2.2.1. Değişken Adlandırmada Kurallar

Değişken adı verilirken uyulması gereken bazı kurallar ve kurallar kadar katı olmasa da yararlı kullanım önerileri vardır.

Değişken adlandırma kuralları:

- Değişkenler bir harf (a - z, A - Z) veya alt çizgi (_) ile başlamalıdır. Bunların dışında sayı veya başka bir karakter ile de başlayamaz.
- Değişken adında rakam, alt çizgi(_), büyük veya küçük harf olabilir.
- Değişken adları herhangi bir uzunlukta olabilir.

Python, değişken adlandırmada Türkçe karakterlerin (ç , ğ, ı, ö, ş ve ü) kullanımına izin vermektedir. Ek olarak Python programlama dilinde ayrılmış sözcükler kullanılamaz. Bu özel sözcüklerin listesini görmek için aşağıdaki kod kullanılabilir.

```
import keyword
keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass',
'raise', 'return', 'try', 'while', 'with', 'yield']
```

Büyük harf ve küçük harf kullanılarak tanımlanan değişkenlerin adı aynı olsa bile farklı değişkenler olduğu unutulmamalıdır.

Örnek 12

Python dilinde uygun değişken adı örnekleri:

```
#Uygun değişken isimleri
sayi1=1
Sayi1=2
```

Sayi1' ekrana yazdırılırsa çıktı ne olur?

```
print (sayi1)
print(Sayi1)
#Büyük harf ve küçük harf kullanarak tanımlanan değişkenlerin adı aynı olsa bile
farklı değişkenler olduğunu unutulmamalıdır.
sayı1=3
#Python  değişken adlandırmada Türkçe karakter kullanımına izin vermektedir.
print(sayı1)
1
2
3
```

Örnek 13

Python'da hatalı değişken adı kullanımı örneği:

```
1sayi=5 #Hatalı değişken adı.
print (1.sayi)
  File "<ipython-input-52-a0b35430cdb5>", line 1
    1sayi=5 #Hatalı değişken adı.
       ^
SyntaxError: invalid syntax
```

2.2.2. Değişken Adlandırma için Standartlar

Değişken adlandırma için bazı standartlar vardır. Bu standartlar değişken adının ve içeriğinin anlaşılmasına yardımcı olarak programcıların daha kolay çalışmasını sağlar. Değişken adı, onun içeriği hakkında bilgi verirse kodun anlaşılması kolaylaşır. Değişken adlarının yazımında bazı standart kullanımlar vardır. Birden fazla kelimenin kullanılacağı değişken adlarında kelimelerin ilk harfi büyük olabilir. Camel standardında (başka standartlar da bulunmaktadır) değişken adlarının görünüşü deve hörgücüne benzetilmektedir. Değişkenin adına küçük harfle başlanır ve sonraki her kelime büyük harfle başlar.

Örnek 14

```
adi= "Elif"
soyadi="Altun"
dogumYili=1981
universiteMezunuMu=True
universiteyeBasladigiYil=1999
mezuniyetNotu=2.00
print ('Adı: ', adi)
print ('Soyadı: ', soyadi)
print('Üniversite Mezunu mu? ', universiteMezunuMu)
print('Üniversiteye Başladığı Yıl: ', universiteyeBasladigiYil)
print('Mezuniyet Notu: ', mezuniyetNotu)
Adı:  Elif
Soyadı:  Altun
Üniversite Mezunu mu?  True
Üniversiteye Başladığı Yıl:  1999
Mezuniyet Notu:  2.0
```

2.3. Veri Tipleri

Veri tiplerini anlayabilmek için aşağıdaki kod satırı incelenebilir. Aşağıda bir sayı ile bir karakter dizisi üzerinde operatörleri kullanarak işlemler yapılmıştır.

Örnek 15

Hatalı veri tipi kullanımına örnek: "sayi2" değişkeninin tek tırnak içinde verildiğine ve bir karakter dizisi olduğuna dikkat edilmelidir.

```
sayi1=5
sayi2='3'
print (sayi1+sayi2)
TypeError                          Traceback (most recent call last)
<ipython-input-55-294ee141ba94> in <module>()
      1 sayi1=5
      2 sayi2='3'
----> 3 print (sayi1+sayi2)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Kod çalıştırıldığında bir hata mesajı alınır. Bir aritmetik operatörü kullanılırken bir sayı ile bir karakter dizisi (ikinci değişkenin adı sizi yanıltmasın) toplamaya çalışıldığı için Python hata verir. Veri tipleri, değişkenler üzerinde yapılabilecek işlemleri belirler.

Örnek 16

Operatörler konusunda * operatörü ile ilgili bir ayrıntıdan bahsedilmiştir.

```
sayi1=5
sayi2='3'
#sayi2 değişkenin tek tırnak içinde verildiğinde bir karakter dizisi olduğuna dikkat
ediniz.
print (sayi1*sayi2)
#Sizce nasıl bir sonuç çıkar?
33333
```

Örnekte görüldüğü gibi kod "çarpma" işlemi yapamamıştır. Çünkü ortada iki sayısal değer yoktur. İkinci örnekte 3 sayısını bir karakter olarak 5 defa yazmıştır.

Python'da değişkenlere değer atanırken veri tipleri belirtilmez. Python, atanan değere göre veri türünü kendisi belirler. Ancak programlama yaparken veri tiplerini bilmek ve ona göre kullanmak gerekir.

2.3.1. Python'da Sık Kullanılan Veri Tipleri

Python programlama dilinde sıkça kullanılan veri tipleri aşağıdaki tabloda sunulmuştur:

Veri Tipi	Sınıfı	Açıklama
Integer	int	Tam sayı. (Örnek: 3, 5, 369963)
Float	float	Ondalıklı sayı. (Örnek: 10.45)
Complex	complex	Karmaşık sayılar A + Bj şeklinde kullanılır. (Örnek: 4+5j)
Karakter dizisi (String)	str	Karakter dizilerini (metinleri) göstermek için kullanılır. Çift tırnak veya tek tırnak içinde gösterilir. "Merhaba Dünya"
Boolean	bool	Sadece True veya False değeri alır. int(True)=1 iken int(False)=0 dır.
Liste	list	Farklı veri türleri içerebilir. listem=['Çınar', 24, 'Mühendis', True]
Demet (tuple)	tuple	Farklı veri türleri içerebilir. demet1=('Çınar', 24, 'Mühendis', True)
Sözlük (dictionary)	dict	Farklı veri türleri içerebilir. sozluk={'adi': 'Çınar','yasi'=24, 'meslekUnvani':'Mühendis', 'askerlikDurumu': True}

2.3.2. Sayısal (int, float ve complex) Veri Tipleri

"int" veri tipi Python'da tam sayıların tutulduğu veri tipidir: 3, 5, 198763 gibi değerleri tutar. En çok kullanılan veri tiplerinden biridir.

"float" veri tipi ondalıklı sayıların tutulduğu veri tipidir: 0.5, 234678.67 gibi değerleri tutar.

"complex" veri tipi ise karmaşık sayıların tutulduğu veri tipidir. A+Bj tipinde veriler tutulur: 4+5j gibi değerleri tutar.

Örnek 17

Python'da bir değişkene değer atandığında veri tipleri atanan değere göre otomatik olarak belirlenir.

```
piSayisi=3.14
#float tipinde bir veri
print ('pi sayısı=', piSayisi)
rCm=2
#integer tipinde veri
alan=3.14*2**2
print ('Alan=', alan)
#sonuç float tipinde
print('Yarıçapı 2 olan dairenin alanı ', alan, ' cm 2 dir' )
karmasikSayi=4+5j
print('Bir karmaşık sayı=', karmasikSayi+3j)
pi sayısı= 3.14
Alan= 12.56
Yarıçapı 2 olan dairenin alanı  12.56  cm 2 dir
Bir karmaşık sayı= (4+8j)
```

2.3.3. Karakter Dizisi (string) Veri Tipi

Karakter dizisi, kullanıcıdan alınan değerlerin metin formatında tutulduğu veri tipleridir. Python karakter dizisi oldukça kullanışlı işlevlere sahiptir.

Örnek 18

Bir karakter dizisi ekrana yazdırılabilir, başka bir karakter dizisiyle birleştirilebilir.

"len( )" metodu bir karakter dizisinin uzunluğunu vermektedir.

```
metin1 = 'Merhaba '
metin2 = 'Mars'
print (metin1)            # karakter dizisinin tamamını yazar
print (metin1 * 2)        # karakter dizisini 2 defa yazar
print (metin1 + metin2)   # iki karakter dizisini birleştirir
print ('metin1  adlı değişkendeki değerin uzunluğu:', len(metin1))
#Boşluğun da bir karakter olduğunu gözden kaçırmayınız.
Merhaba
Merhaba Merhaba
Merhaba Mars
metin1 adlı değişkendeki değerin uzunluğu: 8
```

2.3.3.1. Karakter Dizilerinde (string) Dilimleme İşlemleri

Bir karakter dizisinin içindeki karakterlere tek tek veya belirli bir aralıkta erişilebilir. Köşeli parantez içinde [ ] tek bir sayı verildiğinde bu karakter dizisinin indisini ifade eder. İndis numarası "0" dan başlayarak karakterin metindeki kaçıncı sırada yer aldığını gösterir. metin[0] ifadesi metin değişkenindeki 1. karakteri verir. Belirli bir aralıktaki karakteri alırken "[başlangıç indisi:bitiş indisi]" şeklinde ifade edilir. metin[0:5] metin değişkeninde indisi 0, 1, 2, 3 ve 4 olan karakterleri dilimler. Bitiş indisi dilimlemeye dahil edilmez. İndislerin "0" dan başladığı unutulmamalıdır.

Başlangıç indisi verilmeyen karakter dizisinde örnek: metin[:7] başlangıç indisi otomatik olarak sıfır (0) olur. Örnek 0, 1, 2, 3, 4, 5 ve 6. karakterlerden oluşan bir metin verir. Bitiş indisi değeri verilmezse başlangıç indisinden başlanarak son karakter dâhil dilimleme işlemi yapılır.

Negatif İndis Sayıları: Karakter dizisine sondan başlandığını ifade eder. metin[-1] karakter dizisinin en sonundaki karakteri verir. metin [-2] ise sondan ikinci karakteri verir.

Karakter dizilerinde indisler ritmik atlanarak dilimleme yapılabilir. Bunun için [başlangıç indisi: bitiş indisi: ritmik artış miktarı] şeklinde kullanılır. "metin[0:8:2]" : 0'dan başlayarak 0, 2, 4 ve 6 indis numaralı karakterleri dilimler.

Örnek 19

Karakter dizilerinin kullanımı ile ilgili örnekler:

```
metin='Merhaba Mars'
print (metin[0])    # ifadenin ilk karakterini yazar.
print (metin[4:7])  # ifadenin 5, 6 ve 7. karakterlerini yazar.
print (metin[8::])  # 9. karakterden sonuncu karaktere kadar yazar.
print (metin[-2])   # karakter dizisinin en sondan ikinci karakterini yazar.
print (metin [:7])  # indisi 0' dan 7'ye kadar olan (7 dahil değil) karakterleri yazar.
print (metin[8:])   # başlangıç indisinden sonra tüm karakterleri yazar.
print(metin[0:8:2]) # 0, 2, 4 ve 6 indis numaralı karakterleri dilimler.
M
aba
Mars
r
Merhaba
Mars
Mraa
```

Liste, demet ve sözlükler kapsamlı konular olduğu için ayrı bölümlerde verilmiştir.

2.4. type( ) Metodu Kullanımı

Python, her ne kadar veri tiplerini otomatik olarak verse de bir değişkenin veri tipini kontrol etmek ve kullanım amacına göre değiştirmek gerekebilir. Bir değişkenin veri tipini öğrenmek için "type( )" komutu kullanılır.

Örnek 20

Bir değişkenin veri tipi type( ) komutuyla kontrol edilir. type (degiskenAdi) şeklinde yazılır.

```
sayi1=5
sayi2=10.556
metin1="1"
#metin1 değişkenine tırnak içinde verilen sayının string tipinde bir değişken
olduğuna dikkat ediniz.
sayi3=4+5j
askerlikYaptiMi=True
#True doğru, 1, evet anlamındadır.
print ("1. değişkenin veri tipi: ", type(sayi1))
print ("2. değişkenin veri tipi: ", type(sayi2))
print ("3. değişkenin veri tipi: ", type(metin1))
print ("4. değişkenin veri tipi: ", type(sayi3))
print ("5. değişkenin veri tipi: ", type(askerlikYaptiMi))
listem=['Çınar', 24, 'Mühendis', True]
print ("6. değişkenin veri tipi: ", type(listem))
demet1=('Çınar', 24, 'Mühendis', True)
print ("7. değişkenin veri tipi: ", type(demet1))
sozluk={'adi': 'Çınar','yasi': 24, 'meslekUnvani':'Mühendis', 'askerlikDurumu': True}
print ("8. değişkenin veri tipi: ", type(sozluk))
1. değişkenin veri tipi:  <class 'int'>
2. değişkenin veri tipi:  <class 'float'>
3. değişkenin veri tipi:  <class 'str'>
4. değişkenin veri tipi:  <class 'complex'>
5. değişkenin veri tipi:  <class 'bool'>
6. değişkenin veri tipi:  <class 'list'>
7. değişkenin veri tipi:  <class 'tuple'>
8. değişkenin veri tipi:  <class 'dict'>
```

2.5. Veri Tiplerini Dönüştürmek

Tam sayılarla işlem yapıldığında "integer", kesirli sayılarla işlem yapıldığında "float" veri tipi kullanılmaktadır. Değerler üzerinde işlem yaparken (örneğin "input" ile kullanıcıdan veri alırken) içinde sadece rakamlar bulunan "string" ifadeyi sayısal veri tipine dönüştürmek, bazen de bunun tersini yapmak gerekebilir. Bunun için bazı fonksiyonlar bulunmaktadır. Veri tipini dönüştürmek için kullanılan temel fonksiyonlar şunlardır:

int( ) : Veri tipini integer'a çevirir.

float() : Veri tipini float'a çevirir.

str() : Veri tipini karakter dizisine çevirir.

"integer" tipinde bir sayıyla "float" tipinde bir sayı çarpıldığında sonuç "float" tipinde olacağı için Python bu veri tipini otomatik olarak belirler.

Örnek 21

Aşağıdaki örnekte iki sayının da tırnak içinde verilmiş olduğuna dikkat ediniz.

```
metin1='5'
metin2='3'
print (metin1+metin2)
53
```

Yukarıda kullanılan değerler tırnak içinde verildiğinden karakter dizisi veri tipindedir. Kod sonuç olarak iki karakteri yan yana yazacaktır.

Örnek 22

İki değişkenin veri tipini dönüştürünüz. Veri tipi dönüştürüldüğünde karakter dizisi sayıya çevrilmiş olacaktır. Böylece iki sayısal değer üzerinde toplama işlemi yapılabilir.

```
print(int(metin1)+int(metin2))
8
```

Karakter dizisi olan bir değer sayısal bir ifadeye dönüştürüldüğünde bu ifadenin sadece sayısal karakterler içermesi gerekir. “A” harfi veya metinsel bir ifade int(“A”) kullanılarak sayıya çevrilemez.

Veri tipi, kullanılan değerler ile uyumlu olmalıdır. Veri tipi dönüşümünde bazı değerlerde kayıplar olabilir.

Örnek 23

Aşağıdaki örnekte pi değerinin “float” ve “integer” veri tiplerinde kullanıldığında çıkan sonuca dikkat ediniz.

```
piDegeri=3.14
print ('Veri Tipi: ',type(piDegeri))
#3.14 değerini verdiğimizde piDegeri float veri tipi olarak tanımlanacaktır.
yariCap=5
daireninAlani=((piDegeri*2)*yariCap)
print('Dairenin Alanı (float)=', daireninAlani)
piDegeriInt=(int(piDegeri))
# int(piDegeri*2) ifadesi float değeri int veri tipine dönüştürüldü.
print('Int veri tipine dönüştürülen piDegeri: ', piDegeriInt)
daireninAlani=((piDegeriInt*2)*yariCap)
print('Dairenin Alanı (int)=', daireninAlani)
Veri Tipi:  <class 'float'>
Dairenin Alanı (float)= 31.400000000000002
Int veri tipine dönüştürülen piDegeri:  3
Dairenin Alanı (int)= 30
```

Örnek 24

Boolean veri tipini sayısal veri tipine ve string veri tipine dönüştürebilirsiniz. Bu işlemin tersini de yapabilirsiniz.

```
askerlikYaptiMi=True
print('Askerlik yaptı mı?', askerlikYaptiMi)
askerlikYaptiMiInt=int(askerlikYaptiMi) #integer tipine dönüştürüldü.
print('Askerlik yaptı mı?', askerlikYaptiMiInt)
askerlikYaptiMiStr=str(askerlikYaptiMi) #string tipine dönüştürüldü.
print('Askerlik yaptı mı?', askerlikYaptiMiStr)
#Çıktı olarak True verir ancak bu boolean tipinde değildir.
Askerlik yaptı mı? True
Askerlik yaptı mı? 1
Askerlik yaptı mı? True
```

2.6. Operatörler

Python'da aritmetik, mantıksal işlemler ve ilişkisel karşılaştırmalar gibi işlemler yapmak için operatör (işleç) adı verilen semboller ve özel sözcükler kullanılır. Bu bölümde Python'da kullanılan temel operatörler yer almaktadır.

2.6.1. Aritmetik operatörler

Toplama, çıkarma, çarpma ve bölme gibi işlemler başta olmak üzere aritmetik işlemleri yapmak için kullanılan operatörlerdir.

İşaret	İşlem	Örnek	Sonuç
+	Toplama	5+3	8
-	Çıkarma	5-3	2
*	Çarpma	5*3	15
/	Bölme	5/3	1.6666666666666667
**	Kuvvet	5**3	125
//	Tam sayı bölme	5//3	1
%	Mod	5%3	2

Toplama Operatörü

Bu operatör iki veya daha fazla sayısal değeri toplamak için kullanılır.

Örnek 25

Sayısal değerler "integer" , "float" veya "complex" tipinde olabilir.

```
print (5+3)
print (5+3+3)
sayi1=10
print (sayi1+5)
#Aynı şekilde değişken kullanarak da yapabiliriz.
sayi2=10.34
print (sayi1+sayi2+5.5) #farklı veri tiplerindeki sayıları ve değişkenleri de
kullanabiliriz
8
11
15
25.84
```

Örnek 26

Toplama operatörünün karakter dizilerinde farklı bir kullanımı vardır: İki veya daha fazla karakter dizisini birleştirmek için kullanılabilir.

```
print ('Merhaba ' + 'Mars')
metin1='Merhaba ' + 'Mars' + ' nasılsın?'
print(metin1)
Merhaba Mars
Merhaba Mars nasılsın?
```

Çıkarma Operatörü

Bu operatör, sayıların farkını bulmak için kullanılır.

Örnek 27

```
print(5-3)
sayi1=5
sayi2=3
print(sayi1-sayi2)
print (3-4-5)
2
2
-6
```

Çarpma Operatörü

Çarpma operatörü iki veya daha fazla sayıyı çarpmak için kullanılır.

Örnek 28

```
print(5*3)
print(5*3*2)
sayi1=5
sayi2=3
print(sayi1*sayi2)
15
30
15
```

Çarpma operatörü, toplama operatöründe olduğu gibi karakter dizilerinde farklı bir amaç için kullanılmaktadır.

Örnek 29

Bir karakter dizisini istediğiniz kadar yazdırmak için çarpma operatörünü kullanabilirsiniz.

```
print (20*'BA')
BABABABABABABABABABABABABABABABABABABABA
```

Bölme Operatörü

Bu operatör, iki sayıyı bölmek için kullanılır.

Örnek 30

```
print (5/3)
0.8333333333333334
```

Bölme operatörünü ikiden fazla sayıyı bölmek için de kullanabilirsiniz.

Örnek 31

```
print(18/3/2)
3.0
```

Bu tür durumlarda Python işlemleri soldan başlayarak sırayla yapar. Önce 18/3 işlemini yapar ve sonucu 2'ye böler.

Kuvvet Alma Operatörü

Bir sayının kuvvetini almak için kullanılır. Bir sayının kuvveti o sayının kendisiyle kuvvet kadar çarpılmasıyla bulunur. 5**3=5*5*5

Örnek 32

5'in 3. kuvvetinin bulunması:

```
print(5**3)
125
```

Kuvvet alma operatörü bir sayının kökünü almak için "1/kuvvet" olarak kullanılabilir.

Örnek 33

49'un karekökünü almak için 1/2 kuvveti alınır.

```
print(49**(1/2))
7.0
```

Tam Bölüm Operatörü

İki sayının birbirine tam bölüm sonucunu verir. Bölüm sonucu ondalıklı sayı ise ondalıklı kısmını almaz.

Örnek 34

```
print (121.00//3)
40.0
```

Mod Alma Operatörü

Bir sayının diğer bir sayıya bölümünden kalanını verir.

Örnek 35

```
print (5%3) #Sayının 3 ile bölümünden kalan.
print (9%2) #Kalan 0 ise sayı çifttir.
2
1
```

2.6.2. İlişkisel Operatörler

İlişkisel operatörler, değerler arasındaki ilişkiyi kontrol ederek sonucu "boolean" bir değer olarak (True, False) döndürür. "True" değeri şartın, ilişkinin veya koşulun sağlandığı anlamına gelirken, "False" değeri ise ilişkinin sağlanmadığı anlamına gelir.

İşaret	İşlem	Örnek	Sonuç
==	Eşit mi?	5==3	False
!=	Farklı mı?	5!=3	True
>	Büyüktür?	5>3	True
<	Küçüktür?	5<3	False
>=	Büyük veya eşittir?	3>=3	True
<=	Küçük veya eşittir?	5<=3	False
is	Değerler eşit mi?	'Elif' **is** 'elif'	False
is not	Değerler farklı mı?	'Elif' **is not** 'elif'	True
in	İçeriyor mu?	'bil' **in** 'bilişim'	True
not in	İçermiyor mu?	'bil' **not in** 'bilişim'	False

Eşittir Operatörü

"==" operatörü iki değerin birbirine eşit olup olmadığını anlamak için kullanılır. İki değer birbirine eşitse "True", eşit değilse "False" değeri verir.

Örnek 36

```
print(5==3)
#değişkenlere atadığınız değerleri de aynı şekilde kontrol edebilirsiniz.
sayi1=5
sayi2=3
print(sayi1==sayi2)
print(sayi1==5)
False
False
True
```

Örnek 37

"==" Operatörü karakter dizilerinde de değerlerin eşitliğini kontrol etmek için kullanılır.

```
print('Emre'=='emre')
# Küçük büyük harf duyarlılığına (case sensetive) dikkat ediniz.
metin1='Emre'
metin2='emre'
print(metin1==metin2)
print(metin1=='Emre')
False
False
True
```

Eşit Değildir Operatörü

"!=" operatörü iki değerin birbirinden farklı olup olmadığını anlamak için kullanılır. "==" operatörünün tersine değerler birbirine eşitse "False", eşit değilse "True" değerini döndürür.

Örnek 38

```
print(5!=3)
sayi1=5
sayi2=3
print(sayi1!=sayi2)
print(sayi1!=5)
# Çıktıların == operatörünün tersi olduğuna dikkat ediniz.
True
True
False
```

Örnek 39

"!=" operatörü karakter dizilerinde de değerlerin farklılığını kontrol etmek için kullanılır.

```
print('Emre'!='emre')
# Küçük büyük harf duyarlılığına (case sensetive) dikkat ediniz.
metin1='Emre'
metin2='emre'
print(metin1!=metin2)
print(metin1!='Emre')
True
True
False
```

Büyüktür Operatörü

Büyüktür “>” operatörü iki değeri karşılaştırmak için kullanılır. 1. sayı 2. sayıdan büyükse “True” değilse “False” değerini döndürür.

Örnek 40

```
sayi1=6.06
sayi2=6.07
print(sayi1>sayi2)
False
```

“sayi2” değişkenine yeni bir değer atandığını gözden kaçırmayınız.

```
sayi2=6
print (sayi1>sayi2)
sayi2=6.06
print(sayi1>sayi2)
True
False
```

Küçüktür Operatörü

Küçüktür “<” operatörü iki değeri karşılaştırmak için kullanılır. 1. sayı 2. sayıdan küçükse “True” değilse “False” değerini döndürür. Büyüktür operatörünün tersi işlevini görür.

Örnek 41

```
sayi1=6.06
sayi2=6.07
print(sayi1<sayi2)
True
```

Programımıza devam edelim. “sayi2” değişkenine yeni bir değer atadığımızı gözden kaçırmayın.

```
sayi2=6
print (sayi1<sayi2)
sayi2=6.06
print(sayi1<sayi2)
True
False
False
```

Karakter dizileri, alfabetik olarak sıralandığında sonra gelen ifade daha büyük olarak değerlendirilir.

```
print ('z'>'a')
print ('a'<'z')
True
True
```

Büyük Eşittir (>=) ve Küçük Eşittir Operatörleri

Büyük eşittir “>=” operatörü iki değeri karşılaştırmak için kullanılır. Birinci değer ikinciden büyükse veya ikinciye eşitse “True” değilse “False” değerini döndürür. Küçük eşittir “<=” operatörü ise birinci değer ikinci değerden küçükse veya ikinci değere eşitse “True” değilse “False” değerini döndürür.

Örnek 42

Büyük Eşittir (>=) ve Küçük Eşittir (<=) Operatörleri kullanımı:

```
sayi1=6.06
sayi2=6.06
print(sayi1>=sayi2)
print (sayi1<=sayi2)
#sayılar eşit olduğu için iki operatör de True değeri döndürür.
sayi2=6.07
print(sayi1>=sayi2)
print(sayi1<=sayi2)
#1.sayı 2. sayıdan küçük olduğu için sadece <= operatörü True değerini döndürür.
sayi2=6.05
print(sayi1>=sayi2)
print(sayi1<=sayi2)
#sayi2 değişkeni 6.05 olduğu ve sayi1 sayi2 den büyük olacağı için sadece >=
operatörü True değeri döndürür.
True
True
False
True
True
False
```

"is" ve "is not" Operatörleri

"is operatörü" ve "==" operatörü benzer işleve sahiptir ve iki değerin eşit olup olmadığını kontrol etmek için kullanılır. Değerler eşitse "True" değilse "False" değerini döndürür.

"is not" operatörü ise is operatörünün tersi işlev görür. "is" not operatörü değerler farklı ise True değerini değerler aynıysa "False" değerini döndürür.

NOT

"==" operatörü değerlerin eşitliğini kontrol ederken "is" aynı zamanda her iki değerin aynı nesneyi referans gösterip göstermediğini kontrol eder.

Örnek 43

```
sayi1=5
print (sayi1  is 5)
print (sayi1 is not 5) #is operatörünün tersini verir.
True
False
```

"is" operatörü karakter dizisinde de kullanılabilir.

```
print ('elif' is 'Elif')
#Yukarıdaki kod için büyük harf küçük harf duyarlılığını hatırlayınız.
adi='Elif'
print (adi is 'Elif')
print (adi is not 'Elif') #is operatörünün tersini verir.
False
True
False
```

"in" ve "not in" Operatörleri

"in" operatörü bir karakter dizisinin başka bir karakter dizisinde yer alıp almadığını kontrol etmek için kullanılır. Karakter dizisi, diğer karakter dizisi içinde yer alıyorsa "True" değeri, yer almıyorsa "False" değeri döndürür. "not in" operatörü ise içinde yer almıyorsa "True" yer alıyorsa "False" değeri döndürür.

Örnek 44

```
# in operatörü kullanımı
print ('Bil' in 'Bilişim')
# 2. karakter dizisi içinde 1. karakter dizisi var mı?
print ('Bil' not in 'Bilişim')
# in operatörünün tersi sonuç verir.
True
False
```

NOT

"in operatörünün" for döngüsünde işlevsel bir kullanımı vardır. "Döngüler" konusunda bu kullanımına yer verilmiştir.

2.6.3. Atama Operatörleri

Atama operatörleri değişkene değer atamak için kullanılır. Değişkeni başka bir değerle işleme alarak sonucun yine aynı değişkene atanmasını sağlar.

İşaret	İşlem	Örnek	Sonuç
+=	Artırarak atama	sayi1+=3	8
-=	Eksilterek atama	sayi1-=3	2
*=	Çarparak atama	sayi1*=3	15
/=	Bölerek atama	sayi1/=3	1.6666666666666667
* *=	Kuvvet alarak atama	sayi1**=3	125
//=	Tam sayı bölerek atama	sayi1//=3	1
%=	Mod alarak atama	sayi1%=3	2

NOT

sayi1=5 olarak atanmıştır.

Artırarak Atama Operatörü

Bir değişkenin üstüne sayısal değerleri toplayarak sonucun aynı değişkene atanmasını sağlar.

Örnek 45

Aşağıdaki kod ”sayi1=sayi1+3” koduyla aynı işlevi görür.

```
sayi1=5
sayi1+=3
print (sayi1)
metin1='Merhaba '
metin1+='Mars' #metin1=metin1+"Mars" koduyla aynı işlevi görür.
print(metin1)
8
Merhaba Mars
```

Eksilterek Atama Operatörü

Bir değişkenden bir sayısal değeri eksilterek sonucun aynı değişkene atanmasını sağlar.

Örnek 46

Aşağıdaki kod “sayi1=sayi1-3” koduyla aynı işlevi görür.

```
sayi1=5
sayi1-=3
print (sayi1)
2
```

Çarparak Atama Operatörü

Bir değişkeni bir sayısal değer ile çarparak sonucun aynı değişkene atanmasını sağlar.

Örnek 47

Aşağıdaki kod "sayi1=sayi1*3" koduyla aynı işlevi görür.

```
sayi1=5
sayi1*=
print (sayi1)
#Aynı şekilde karakter dizilerinde de kullanabilirsiniz.
metin1='Merhaba '
metin1*=3 #metin1=metin1*3 koduyla aynı işlevi görür.
print(metin1)
15
Merhaba Merhaba Merhaba
```

Bölerek Atama Operatörü

Bir değişkeni bir sayısal değere bölerek sonucun aynı değişkene atanmasını sağlar.

Örnek 48

Aşağıdaki kod "sayi1=sayi1/3" koduyla aynı işlevi görür.

```
sayi1=5
sayi1/=3
print(sayi1)
1.6666666666666667
```

Kuvvet Alarak Atama Operatörü

Bir değişkenin kuvvetini alarak sonucun aynı değişkene atanmasını sağlar.

Örnek 49

Aşağıdaki kod "sayi1=sayi1**3" koduyla aynı işlevi görür.

```
sayi1=5
sayi1**=3
print (sayi1)
125
```

Tam Sayı Bölerek Atama Operatörü

Bir değişken sayıya bölündüğünde sonucun (ondalık kısmını atarak) aynı değişkene atanmasını sağlar.

Örnek 50

Aşağıdaki "kod sayi1=sayi1//3" koduyla aynı işlevi görür.

```
sayi1=5
sayi1//=3
print (sayi1)
1
```

Mod Alarak Atama Operatörü

Bir değişkenin bir sayıya bölümünden kalanın aynı değişkene atanmasını sağlar.

Örnek 51

Aşağıdaki kod "sayi1=sayi1%3" koduyla aynı işlevi görür.

```
sayi1=5
sayi1%=3
print (sayi1)
2
```

2.6.4. Mantıksal Operatörler

İfadeleri mantıksal olarak bağlamak için kullanılan: “or”, “and” ve “not” operatörleridir.

“or” Operatörü

“or” operatörü “veya” anlamındadır. Belirtilen koşullardan birinin sağlanması durumunda “True” değeri döndürür. Bir sayı 6’dan küçük **veya** 10’dan büyükse koşulunu düşünelim. Sayımız 5 ise 6’dan küçük olduğu için bu şartı sağlayacaktır. Sayımız 6, 7, 8, 9 veya 10 olursa şartların her ikisini de sağlamadığı için “False” değeri döndürülür.

Örnek 52

```
sayi1=5
#or operatörü kullanımı
print (sayi1<6 or sayi1>10)
adi='Elif'
print (adi=='Elif' or adi=='Emre')
#Adı Elif veya Emre ise True değerini döndürür.
meslekUnvani='Mühendis'
print (meslekUnvani=='Öğretmen' or meslek=='Doktor')
#Meslek ünvanı Öğretmen veya Doktor olmadığı için False döndürür.
print (meslekUnvani=='Öğretmen' or meslekUnvani=='Doktor' or
meslekUnvani=='Mühendis')
#Meslek Ünvanı Öğretmen veya Doktor veya Mühendis'ten biri ise True değerini
döndürür.
# İkiden fazla koşul için de kullanılabilir.
True
True
False
True
```

"and" Operatörü

Bu operatör "ve" anlamındadır. Belirtilen koşulların hepsinin sağlanması durumunda "True" değerini döndürür. Bir öğrencinin ders puanı 50'den büyük ve 60'tan küçükse koşulunu düşünüldüğünde öğrencinin puanı 50 ise her iki şartı da sağlayacaktır ve "True" değerini döndürecektir.

Örnek 53

```
ogrenciDersPuani=50
print (ogrenciDersPuani>50 and ogrenciDersPuani<60)
adi='Emre'
yasi=24
print (adi=='Emre' and yasi>=20)
#Adı Emre ve yaşı en az 20 ise True değerini döndürür.
meslekUnvani='Mühendis'
askerlikDurumu='Yaptı'
isTecrubeYil=2
print (meslekUnvani=='Mühendis' and askerlikDurumu=='Yaptı')
print (meslekUnvani=='Mühendis' and askerlikDurumu=='Yaptı' and isTecrubeYil>=3)
False
True
True
False
```

Meslek unvanı Mühendis ise ve askerliğini yapmışsa "True" değerini döndürür.

Meslek ünvanı Mühendis ise, askerliğini yapmışsa ve en az 3 yıl iş tecrübesi varsa "True" değerini döndürür.

"not" Operatörü

Bu operatör "değil" anlamındadır. Belirtilen koşulun tersi doğruysa "True" değeri verir. Bir öğrencinin puanı 45'ten küçük değilse ifadesi düşünüldüğünde öğrencinin puanı 50 ise "True" değerini döndürür.

Örnek 54

```
ogrenciDersPuani=50
print(not(ogrenciDersPuani<45))
print (ogrenciDersPuani>45) #Yukarıdaki ifade ile aynı işlevi görür.
True
True
```

2.6.5. Operatörlerde Öncelik Sırası

Farklı operatörler birlikte kullanılabilir. Operatörler birlikte kullanılırken hangi işlemin önce yapılacağına dair bir sıralama vardır.

- Parantez içindeki işlemler her zaman öncelikli olarak yapılır.
- Çarpma ve bölme işlemleri toplama ve çıkarma işlemine göre önce yapılır.
- Aynı derecedeki operatörlerde işlem sırası önceliği soldan sağa doğrudur.
- İşlemlerde öncelik sırasını belirtmek için en iyi yöntem operatörleri parantez içinde kullanmaktır.

Örnek 55

```
print((3+5)*2) #Bu işlemin sonucunu tahmin ediniz.
#Öncelikle parantez içi yapıldığında 8*2=16
print (3+5*2) #Bu işlemin sonucu kaçtır?
#Öncelikle çarpma işlemi yapıldığından 3+10=13
print (3**2*2)
print (6*7/7)
print (6*3/2+8/2*3)
16
13
18
6.0
21.0
```

Bu bölümde temel operatörler verilmiştir. Python'da bu operatörlerin yanında başka operatörlerde yer almaktadır. Ayrıca kullandığınız Python sürümüne göre farklı operatörler kullanılabilir.

2.7. Bölüm Sonu Örnekleri

1. Aşağıdaki kod çalıştırıldığında ekrana "True" veya "False" değerlerinden hangisi yazılır?

```
sayi1=8*5/2
print (sayi1>20)
```

2. Aşağıdaki kod çalıştırıldığında ekrana "True" veya "False" değerlerinden hangisi yazılır?

```
vizePuani=80
finalPuani=70
sonuc=(0.4*vizePuani)+(finalPuani*0.6)
print (sonuc>=60)
```

3. Aşağıdaki değişken adlandırmalarından hangisi yanlıştır?

```
_sinav_puani1=80
sinavPuani=90
1sinavpuani=100
SınavPuanı=70
```

4. Aşağıdaki kod çalıştırıldığında ekrana hangi değişken tipi yazılır?

```
plaka='06'
print(type(plaka))
```

5. Aşağıdaki kod çalıştırıldığında ekrana ne yazılır?

```
#sehir='Ankara'
sehir='İstanbul'
print ('Şehir: ', sehir)
```

6. Aşağıdaki kod çalıştırıldığında ekrana ne yazılır?

```
sehir='Ankara'
sehir='İstanbul'
print (sehir!='istanbul') #operatöre dikkat
#Büyük harf küçük harf duyarlılığına dikkat
print (sehir=='İstanbul')
```

7. Veliye öğrencisinin devamsızlığını bildiren mesaj metni yazan kodları yazınız. Veli adını soyadını, mesajın içeriğini, devamsızlık süresini ayrı değişkenlere atayarak aşağıdaki gibi bir mesajda yazdırın. Sn..... .....öğrencinizin toplam devamsızlığı .... gündür. Python Lisesi

8. Tüketiciye su faturasını bildiren mesaj metni oluşturan kodları yazınız.

Kullanıcıya fatura tutarını belirten mesajda kullanılacak değişkenler: Hitap şekli, abone numarası, tüketim dönemi, tüketim tutarı*1.

Sayın ...nolu abonemiz ..... dönemi faturanız ..... TL'dir. Python Belediyesi

Cevaplar

1.
```
False
```

2.
```
True
```

3. "1sinavpuani" yanlıştır.
```
File "<ipython-input-35-36471b2a21d1>", line 3
1sinavpuani=100
^
SyntaxError: invalid syntax
```

4.
```
Karakter dizisi
<class 'str'>
```

5.
```
Şehir: İstanbul
```

6.
```
True
True
```

7.
```
onMesaj="Sayın "
veliAdi='Murat KARA'
mesajMetni =" öğrencinizin toplam devamsızlığı: "
devamsizlik='19.5'
veliMesaj=(onMesaj+veliAdi+mesajMetni+devamsizlik+' gün Python lisesi ')
print(veliMesaj)
Sayın Murat KARA öğrencinizin toplam devamsızlığı : 19.5 gün Python lisesi
```

8.
```
onMesaj="Sayın "
mesajSonu=" Mayıs 2020 dönemi faturanız: "
aboneNo="29101923"
tuketim=20
tuketimTutari=int(tuketim)*1.0
mesaj=onMesaj+str(aboneNo)+' nolu abonemiz ' +
mesajSonu+str(tuketimTutari)+" TL' dir. Python Belediyesi"
print(mesaj)
```
```
Sayın 29101923 nolu abonemiz  Mayıs 2020 dönemi faturanız : 20.0 TL' dir.
Python Belediyesi
```

MODÜL 3

PYTHON'DA TEMEL FONKSİYONLAR

Şekil 3.1: Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.

3.1. Temel Fonksiyonlar

Programlama dillerinde fonksiyonlardan sıkça faydalanılmaktadır. Fonksiyonları kendimiz yazabileceğimiz gibi, programlama dillerinin kütüphanelerinde bulunan hazır fonksiyonlar da kullanılabilir. Python'da pek çok hazır fonksiyon bulunmaktadır. Bunlara **yerleşik fonksiyonlar** da denilmektedir. Peki, neden fonksiyonlara ihtiyaç duyulmaktadır? Fonksiyonlar yapılan işlemleri kolaylaştırır ve zaman alıcı işlevleri kolay bir şekilde yerine getirilmesini sağlar. Bunun yanında **print( )** ve **input( )** gibi kullanıcıya bir çıktı vermek ve kullanıcıdan girdi almak için kullanılan, programlama dilinin olmazsa olmaz fonksiyonları vardır. Kullanıcıdan bir girdi almadan ya da yapılan işlemlerin sonucunu kullanıcıya vermeden

program yazmanın bir anlamı olmaz. Örneğin, kullanıcının vücut kitle indeksini hesaplamak için kullanıcının boy ve kilo verisine ihtiyacımız vardır. Eğer kullanıcıdan bir girdi alınmazsa bu hesaplamayı yapmak mümkün değildir.

Aynı şekilde yapılan işlemin sonucunu görmek için de sonucun ekranda görüntülenmesi, yani programın kullanıcıya çıktı vermesi gerekmektedir.

Python'da konsola (etkileşimli kabuk) veriler yazdırmak istendiğinde **print( )** fonksiyonu, kullanıcıdan bir girdi almak istendiğinde ise **input( )** fonksiyonu kullanılmalıdır. Ayrıca her iki fonksiyonun alacağı parametreler ve/veya beraber kullanılabileceği fonksiyonlar bulunmaktadır.

Kullanıcıdan girdi alırken kimi zaman sayısal ifadeler (integer) kimi zaman da metinsel ifadeler (string) istenebilir. Yani programın bir aşamasında kullanıcıdan ismini ya da yaşadığı şehri girmesi, başka bir aşamasında ise yaş verisini alarak bununla ilgili işlem yapılabilir. Peki, Python girilen verinin sayısal değer mi, yoksa metinsel bir ifade mi olduğunu nasıl anlayacak? İşte bu gibi durumlarda input( ) fonksiyonu ile beraber farklı fonksiyonlar da kullanılabilir.

3.2. print( ) Fonksiyonu

print( ) fonksiyonu, konsola çıktı göndermek amacıyla kullanılır. Programların genellikle yapılan işlemler sonucunu kullanıcıya sunması gerekir. Programda veri **print( )** fonksiyonu ile Python'daki konsolda görüntülenebilir. **print( )** fonksiyonunun kullanımı:

Örnek 1

`print` "print sözcüğü yazılır"

`print( )` parantez açılır

`print("değer")` konsola gönderilecek değer bir metinsel ifade ise bu ifade çift tırnak, tek tırnak ya da üç çift tırnak içinde yazılmalıdır.

```
print('değer')
```

```
print("""değer""")
```

Her üç kullanım da bize aynı çıktıyı verir. Eğer konsola gönderilecek değer bir değişken ya da sayısal bir ifade ise bu durumda tırnak içerisinde yazmaya gerek yoktur.

Örnek 2

```
print(5)
```

Burada **print( )** fonksiyonu kullanılırken parantez içerisinde kullanılan değerlere **argüman** denilmektedir. Python, **print( )** fonksiyonu argümanını kontrol ederek, belirtilen kurallara uyup uymadığını kontrol eder. Söz diziminin doğruluğu veya değişkenin tanımlanmış olması kontrol edilir. Kod, Python'un izin verdiği tanımlamalara uyuyorsa çalıştırılarak konsol üzerinden sonuç görüntülenir. Aksi durumda hata mesajı alınır.

Örnek 3

```
print("python)
SyntaxError: EOL while scanning string literal
```

Örnek 4

```
print(a)
SyntaxError: unexpected indent
```

Yukarıdaki durumlarda eksik olan tanımlamalar neticesinde, hata mesajı ile karşılaşılmıştır. Örnek 3'te tırnak kapatılmamış, örnek 4'te ise değişken tanımlanmadığı için hata vermiştir. Doğru kullanım:

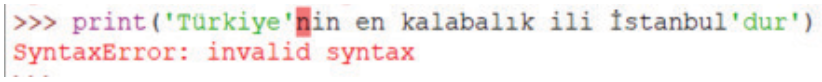
Örnek 5

```
print("Merhaba, Python!")
Merhaba, Python!
```

print( ) fonksiyonu kullanılırken, karakter dizilerinde çift tırnak, tek tırnak ya da üç çift tırnak kullanılabilir. Aslında bu ayrıntının Python'da çok önemli bir yeri vardır.

Örnek 6

```
print('Türkiye'nin en kalabalık ili İstanbul'dur')
SyntaxError: invalid syntax
```



Şekil 3.2: Örnek 6'nın çıktısı

Örnek 6'daki kullanıma bakıldığında karakter dizisi tek tırnak işareti ile başlamış, ancak kesme işareti olarak kullanıldığı yerde sonlanmıştır **'Türkiye'nin** daha sonraki ifadeler ise karakter dizisinden ziyade, farklı bir argüman olarak değerlendirilmiş ancak bunlarla ilgili bir tanımlama yapılmadığı için hata mesajı vermiştir. Burada çift tırnak ve tek tırnak beraber kullanılarak sorunun çözümüne gidilebilir.

Örnek 7

```
print("Türkiye'nin en kalabalık ili İstanbul'dur")
Türkiye'nin en kalabalık ili İstanbul'dur
```

print( ) fonksiyonu kullanılırken argüman değerleri arasında aritmetiksel işlemler yapılabilir.

Örnek 8

```
print(2+2)
4
```

print( ) fonksiyonu, bu şekilde aritmetik işlemler yapmak amacıyla kullanılabilir.

3.2.1. print( ) Fonksiyonu ile Kullanılabilen Parametreler

print( ) fonksiyonu kullanılırken karakter dizilerinde tek tırnak ve çift tırnak işaretleri (‘ ,“ ,“““) kullanılır. Bu işaretler karakter dizilerinin nerede başladığını ve bittiğini ifade eder.

3.2.2 Ters Taksim(\ )

Örnek 9

```
print("Merhaba," Python "kullanıyorum")
SyntaxError: invalid syntax
```

Örnek 9’da çift tırnak işareti arasındaki “Merhaba, ” kelimesi karakter dizisi olarak algılandı ama sonrasında **Python** kelimesi çift tırnak içinde ve argümanlar arasında “,” virgül olmadığı için hata verdi. Bu problemi çözmek için **kaçış parametreleri (escape character)** kullanılmalıdır.

Aslında bu sorun, bir önceki bölümde çift tırnak ve tek tırnak’lar beraber kullanılarak çözülmüştü. Ancak burada ters taksim (\ ) işareti kullanılarak da bu sorun çözülebilir.

Örnek 10

```
print('Bursa\'nın iskenderi meşhurdur.')
Bursa'nın iskenderi meşhurdur.
```

Örnek 10’da ters taksim işareti kullanılmamış olsaydı önceki bölümdeki gibi hata ile karşılaşılırdı. Burada \ karakteri kendinden sonra gelen kesme ‘ işaretinin dikkate alınmaması gerektiği anlamı vermektedir. Aynı işlem çift tırnak içinde de yapılabilir.

3.2.3. Alt Satır Başı (\n)

print( ) fonksiyonu kullanılırken, karakter dizilerinde bazen alt satıra inme ihtiyacı duyulabilir. Python'da en sık kullanılan kaçış parametresi **\n** parametresidir.

Örnek 11

```
print("1. satır\n2. satır\n3. satır")
1. satır
2. satır
3. satır
```

Örnek 11'de görüldüğü üzere **\n** parametresi kullanılarak program çıktısının alt alta yazılması sağlandı.

3.2.4. Sekme(\t)

Klavyeden **tab** tuşuna basıldığındaki gibi belirli karakter boşluk bırakılmasını sağlayan bir parametredir.

Örnek 12

```
print("pazartesi \tsalı \tçarşamba")
pazartesi     salı     çarşamba
```

3.2.5. end( ) Parametresi

Bu parametre **print( )** fonksiyonu ile ekrana gönderilen değerlerin sonunda hangi işlemin yapılacağını belirtmektedir.

Örnek 13

```
print("Merhaba!")
print("Python")
Merhaba!
Python
```

Örnek 13'teki gibi bir kullanımda kodların çıktısı alt alta verilmiştir. Ancak bazı durumlarda programın çıktısı birleştirilmek istenebilir. İlerleyen bölümlerde döngü konusunda, döngü değerini her seferinde konsola yazdırılmak istendiğinde bu değerler alt alta yazılacaktır. Döngünün büyük olduğu düşünülürse program sayfalar dolusu çıktı verebilir. İşte bu gibi durumlarda end parametresi çok işe yaramaktadır.

Örnek 14

```
print("Merhaba!",end=" ")
print("Python")
Merhaba! Python
```

Örnek 14'te görüldüğü üzere **end** parametresi içinde iki tırnak arasında boşluk karakteri kullanıldığı için program çıktısını birleştirerek araya boşluk eklendi. Aynı işlem virgül kullanılarak yapılabilir:

Örnek 15

```
print("Merhaba!",end=",")
print("Python")
Merhaba!,Python
```

Aslında **end** parametresinin içerisinde standart olarak **\n** parametresi vardır. Burada değer atanarak varsayılan değer değiştirilmiştir.

3.2.6. sep () Parametresi

Önceki örnekte, **end** parametresi ile değerlerin sonunda hangi işlemin yapılacağı gösterilmişti. Tek bir **print( )** fonksiyonu birden fazla argüman alabilir. Her bir argümanın arasında farklı işlemler **sep** parametresi ile yapılabilir.

Örnek 16

```
print("pazartesi","salı","çarşamba","perşembe","cuma")
pazartesi salı çarşamba perşembe cuma
```

Örnek 16'daki gibi bir kullanımda, her bir argüman birbirinden virgül işareti ile ayrılmış ve program çıktısı olarak tüm argümanların arasına birer boşluk bırakılarak ekrana yazdırılmıştır. Burada dikkat edilmesi gereken bir başka nokta, **print( )** fonksiyonu içerisine gönderilen tüm değerleri belirli kurallar çerçevesinde ekrana yazılmıştır. Her bir argüman'ın arasına **sep** parametresi yardımıyla birer kural belirlenmek istenirse,

Örnek 17

```
print("pazartesi","salı","çarşamba","perşembe","cuma",sep="-")
pazartesi-salı-çarşamba-perşembe-cuma
```

3.2.7. Format( ) Metodu ile Biçimlendirme İşlemleri

Program yazarken bazı durumlarda bir string'in içinde daha önceden tanımlı string, float, int gibi farklı türden değerleri yerleştirmek isteyebiliriz. Böyle durumlar için Pythonda **format( )** metodu bulunmaktadır. Örneğin, programda 3 adet tam sayı değeri, bir string ifade ile beraber ekrana yazdırılmak istenebilir. Bunun için **format( )** metodu kullanılmalıdır.

Örnek 18

```
a=5
b=6
c=9
print("girdiğiniz",a, b, "ve",c,"değerlerinin toplamı: ",a+b+c,"dir")
girdiğiniz 5 6 ve 9 değerlerinin toplamı:  20 dir
```

Örnek 18'deki gibi bir kullanım ve hata yapmaya müsait bir kullanımdır. Python bu gibi durumlar için **print( )** fonksiyonunda format metodunun kullanımına olanak sağlar.

Kullanımı aşağıdaki şekildedir:

Örnek 19

```
print("çıktı işlemi {} {} {}".format(1,2,3))
çıktı işlemi 1 2 3
```

Burada **print( )** fonksiyonunda kullanılan her bir **{}** ifadesine karşılık olarak **format( )** metoduna bir adet argüman verilmelidir. Önceki örnek bu şekilde yapılmak istenirse.

Örnek 20

```
a=5
b=6
c=9
print("girdiğiniz {} , {} ve {} değerlerinin toplamı= {} dir".format(a,b,c,a+b+c))
girdiğiniz 5 , 6 ve 9 değerlerinin toplamı= 20 dir
```

Süslü parantez içine sayılar girerek format metodu ile hangi sıradaki değerin geleceği belirlenebilir.

Örnek 21

```
print("{1} {0} {2}".format(10,"Python",20))
Python 10 20
```

Uygulama

Aşağıdaki örnekleri etkileşimli kabuk üzerinde yaparak çıktılarını gözlemleyiniz.

1.
```
print("Memleket isterim,\nGök mavi, dal yeşil, tarla sarı olsun,")
```

2.
```
print("Memleket isterim,","Gök mavi, dal yeşil, tarla sarı olsun,")
```

3.
```
print("Memleket isterim,")
print("Gök mavi, dal yeşil" ,"tarla sarı olsun,")
```

4.
```
print("Memleket isterim,",end=" ")
print("Gök mavi, dal yeşil" ,"tarla sarı olsun,")
```

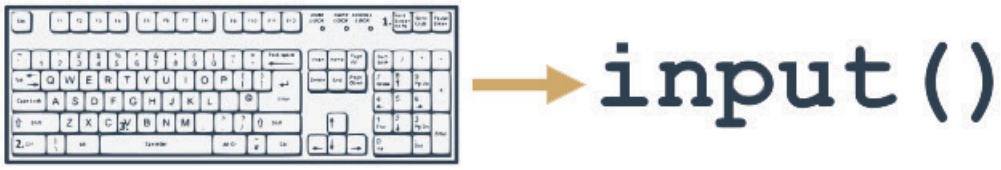
5.
```
print("Memleket","isterim","Gök mavi","dal", "yeşil",sep="-")
```

6.
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

7.
```
not1 = 55
not2 = 100
print('1.Sınav Notu:{} \n2.Sınav Notu:{}'.format(not1,not2))
```

3.3. input( ) Fonksiyonu

Python'da **print( )** fonksiyonu sayesinde konsol ekranına çıktılar gönderilebilir. Önceki örneklerde bu işlem tanımlı veriler üzerinden yapılmıştır, yani kullanıcıdan bir değer almadan, program içerisinde yapılan değişken tanımlalamaları ile **print( )** fonksiyonu ile değerler ekrana yazdırılmıştır. Neredeyse tüm programlama dilleri verileri okur ve işler. Kullanıcıdan girdi almayan bir program sağır bir programdır. **input( )** fonksiyonu kullanıldığında genellikle kullanıcının klavyeden bir girdi yapmasını bekler.



Şekil 3.3: *input( ) girişi*

Python'da **input( )** fonksiyonu diğer programlama dillerinden daha işlevsel bir yapıya sahiptir. **input( )** fonksiyonuyla, **print( )** fonksiyonuna ihtiyaç duymadan kullanıcıya bilgi verilebilir. Ancak **input( )** fonksiyonu kullanılırken kullanıcıdan alınan değer bir değişkene atanmalıdır.

Örnek 22

```
isim=input("isminizi giriniz: ")
print("merhaba! ",isim)
isminizi giriniz: ahmet
merhaba!  ahmet
```

Örnek 22'de **input( )** fonksiyonu ile kullanıcıdan bir karakter dizisi girmesi beklenmiş ve sonuç ekrana yazdırılmıştır. Peki, Python girilen değerin sayısal bir değer mi yoksa bir karakter dizisi mi olduğunu nasıl anlayacak?

Örnek 23

```
a=input("birinci sayıyı giriniz: ")
b=input("ikinci sayıyı giriniz: ")
print("girdiğiniz sayıların toplamı: ",a+b)
birinci sayıyı giriniz: 6
ikinci sayıyı giriniz: 8
girdiğiniz sayıların toplamı: 68
```

Örnek 23'te görüldüğü üzere uygulama bize hatalı bir çıktı vermiştir. Bu kullanımda her iki değer bir karakter dizisi olarak algılanmış ve Python iki değer üzerinde toplama işlemi yapamadığı için yan yana yazarak birleştirmiştir. Aşağıdaki örnekle bu durum daha iyi anlaşılacaktır:

Örnek 24

```
print("ahmet"+"şahin")
ahmetşahin
```

Burada yapılan işlemle gerçekte bir toplama işlemi yapılmamış iki argüman birbirleriyle birleştirilmiştir. **input( )** fonksiyonu kullanılırken girdi olarak sayısal ifadeler kullanılacağı zaman bu durumun Python'a bildirilmesi gerekmektedir. Bunun için Örnek 25'teki kullanım uygulanmalıdır.

Örnek 25

```
a=int(input("Bir sayı giriniz: "))
Bir sayı giriniz: 5
```

Örnek 25'te görüldüğü üzere **input( )** fonksiyonu, **int( )** fonksiyonunun içerisine alınarak girdi sayısal ifadeye çevrilmiştir.

Örnek 26

```
a=int(input("birinci sayıyı giriniz: "))
b=int(input("ikinci sayıyı giriniz: "))
print("girdiğiniz sayıların toplamı:",a+b)
birinci sayıyı giriniz: 5
ikinci sayıyı giriniz: 8
girdiğiniz sayıların toplamı: 13
```

Görüldüğü üzere burada yapılan işlemle girdiler sayılara dönüştürülmüş ve toplama işlemi yapılmıştır. **input( )** fonksiyonu kullanılırken sık yapılan hataların başında hatalı veri girişleri gelmektedir. **int( )** fonksiyonu ile beraber integer ifade yerine string bir değer girilirse Python hata verir. Çünkü metinsel ifadeler (a, b, isim, soyisim) sayılara dönüştürülemez.

Örnek 27

```
a=int(input("birinci sayıyı giriniz: "))
b=int(input("ikinci sayıyı giriniz: "))
print("girdiğiniz sayıların toplamı:",a+b)
birinci sayıyı giriniz: 8
ikinci sayıyı giriniz: Ali
Traceback (most recent call last):
  File "C:/Users/asus/Desktop/1.py", line 2, in <module>
    b=int(input("ikinci sayıyı giriniz: "))
ValueError: invalid literal for int() with base 10: 'Ali'
```

Yine benzer şekilde girilen ifadelerden birisi sayısal olup diğer ifade sayıya dönüştürülmezse, sayılarla metinsel ifadeler arasında aritmetiksel işlem yapılamayacağı için Python hata verecektir. Hataların ayıklanmasına yönelik çözümlere modül 13'te detaylı olarak değinilecektir.

Örnek 28

```
a=int(input("birinci sayıyı giriniz: "))
b=input("ikinci sayıyı giriniz: ")
print("girdiğiniz sayıların toplamı:",a+b)
birinci sayıyı giriniz: 6
ikinci sayıyı giriniz: 9
Traceback (most recent call last):
  File "C:/Users/asus/Desktop/1.py", line 3, in <module>
    print("girdiğiniz sayıların toplamı:",a+b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Örnek 28'de **integer** ve **string** değerler arasında işlem yapılamayacağı için Python hata vermiştir.

Örnek 29

Kullanıcıdan abone numarası ve tüketim bilgisi alarak fatura tutarını hesaplayan programı yazınız.

```
on_mesaj="Sayın "
mesaj_sonu=" nisan dönemi faturanız: "
abone_no=input("abone numaranız")
tuketim=input("tuketim miktarı")
tuketim_tutari=int(tuketim)*4.0
mesaj=on_mesaj+abone_no+mesaj_sonu+tuketim+" tl dir."
print(mesaj)
abone numaranız: 123456
tuketim miktarı: 20
Sayın 123456 nisan dönemi faturanız: 20 tl dir.
```

3.4. Bölüm Sonu Örnekleri

1. Ekrana “Merhaba Dünya” yazan programı yazınız.

2. Kullanıcıdan ismini alarak merhaba “Ali” şeklinde çıktı veren programı yazınız.

3. Girilen iki sayının toplamını bulan ve ekrana yazdıran programı yazınız.

4. Girilen iki sayının ortalamasını bulan ve ekrana yazdıran programı yazınız.

Cevaplar

1.
```
print("Merhaba Dünya")
```

2.
```
isim = input('İsminizi Girin : ')
print("Merhaba "+isim)
```

3.
```
sayi1 = int(input('1. sayıyı girin: '))
sayi2 = int(input('2. sayıyı girin: '))
print("girdiğiniz sayıların toplamı: ",sayi1+sayi2)
```

4.
```
sayi1 = int(input('1. sayıyı girin: '))
sayi2 = int(input('2. sayıyı girin: '))
print("girdiğiniz sayıların ortalaması: ",(sayi1+sayi2)/2)
```

MODÜL 4

KOŞULLU VE MANTIKSAL İFADELER

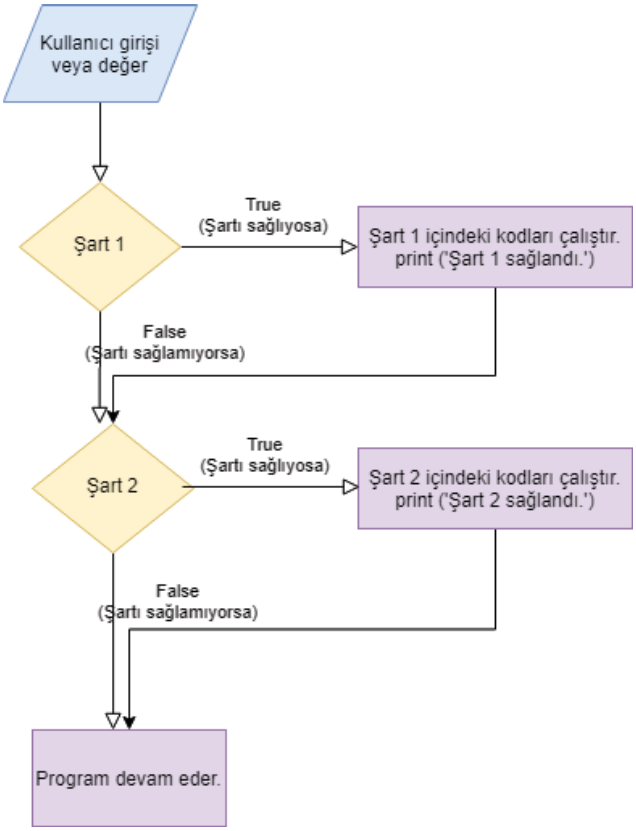
Şekil 4.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

4.1. Koşullu İfadeler

Program yazarken kodlar sıralı olarak alt alta satırlar şeklinde yazılır. Programın akışında bazı dallanmalar (farklı durumlar için farklı kodların çalışmasını istediğimiz durumlar) olabilir. Programlarda belirli koşul veya durumlar için çalışması istenen kodlar koşullu ifade blokları kullanılarak oluşturulur.

Örneğin, sadece üyelerin giriş yapabileceği bir program hazırlarken kullanıcı adı ve parolası doğru olan kişilerin sisteme erişimine izin veren kodlar yazılır. Bunun için programa koşul ifadeleri eklenmesi gerekir. Koşul sağlanıyorsa menülere erişim izni verecek kodlar çalışmaya başlar. Koşul sağlanmıyorsa uyarı mesajı verilerek sisteme giriş izni verilmez.

Mantıksal operatörler sonuç olarak "boolean" veri tipinde değer verir. Eğer koşul sağlanırsa "True" değeri döndürürken koşul sağlanmazsa "False" değeri döndürür. "Boolean" veri tipi bu iki değerden başka bir değer alamaz. Bu durum koşullu ifadeler üretme olanağı sağlar. Koşullu ifadelerin sonucu "boolean" değer kontrol edilerek program akışı yönlendirilebilir. Koşul ifadesi ve "True-False" akışı Şekil 4.2'de daha iyi görülebilir.



Şekil 4.2: *Koşullu ifadeler*

Şekil 4.2'de görüldüğü üzere kullanıcıdan alınan veri "Şart 1" yapısına geldiğinde şartı sağlıyorsa (True) bu kod girintisi (blok) içindeki komutlar çalışır. "Şart 1" yapısında şart sağlanmıyorsa "blok" atlanarak sonraki kodlara geçilir. Sonrasında yeni şart yapısı (Şart 2) aynı şekilde kontrol edilir ve akış devam eder.

4.2. Mantıksal İfadeler ve Bağlaçlar

Bir mantıksal ifadeyi diğer mantıksal ifadelerle bağlamanın farklı yolları vardır. Modül 2'de açıklanan ilişkisel operatörler ve mantıksal bağlaçlar kullanılarak (or, and, not vb. gibi) farklı koşul durumları oluşturulabilir.

kulaniciAdi==’Admin’ and kullaniciParola==’123456’ Kullanıcı adı ve parola doğru girilirse “True” değeri döndürür.

Örnek 1

Aşağıdaki örnek kodda kullanıcı adı ve şifresi doğru girilirse ekrana “True” ikisinden biri bile yanlış olursa “False” değeri döndürür.

```
kullaniciAdi=input('Kullanıcı Adı:')
kullaniciParola=input('Parola:')
print(kullaniciAdi=='Admin' and kullaniciParola=='123456')
Kullanıcı Adı:Admin
Parola:123456
True
```

Örnek 2

Bölümü Bilgisayar veya Elektronik olanları seçmek için bir kod yazalım. bolum==’Bilgisayar’ “or” bolum==’Elektronik’ koşullarından biri doğruysa “True” değilse “False” değeri döndürür.

```
bolum=input('Bölümünüzü giriniz: ')
#Büyük harf küçük harf duyarlılığını ulatmayın
print(bolum=='Bilgisayar' or bolum=='Elektronik')
Bölümünüzü giriniz: Bilgisayar
True
```

Örneklerde sadece mantıksal operatörlerin sonucu “boolean” değeri ekrana yazdırıldı. Mantıksal operatörler koşul ifadeleriyle birlikte kullanıldığında belirli şartlarda belirli kod blokları çalıştırılabilir.

4.3. Python Blok Yapısı

Python'da (başka programlama dillerinde de) kodlar belirli alt kümeler hâlinde (blok) ifade edilir. Bu yapı Python'da girinti (Indentation) ile oluşturulur. Python'da dikey olarak aynı hizadaki kodlar aynı blok yapısında yer alır. Kod bloklarının kolaylıkla ayırt edilebilmesi için bir sekme (4 karakter boşluk) kullanılması önerilmektedir. Eğer bir IDE (bütünleşik program geliştirme ortamı) kullanılıyorsa bu girintiler otomatik olarak ayarlanacaktır. Kullanılan geliştirme ortamında ayarlar bölümünde bir girintinin kaç karakter olacağının belirlendiği ayarlar bulunmaktadır. Bir karakter boşluk ile oluşturulan girinti bile yeni blok yapısını oluşturur. Bir kod bloğu kendi içinde tutarlı bir yapıdır. Döngüler, fonksiyonlar ve koşul ifadeleri kod blokları kullanılarak oluşturulur.

```
[ ]  1  yasi=17
     2  adi='Tahsin'
     3  if (yasi>=18):                    1. Blok
     4      print('1. Şart sağlandı')      if Bloğu
     5      print ('1. if bloğunun içindesiniz')
     6  if (adi=='Tahsin'):               2. Blok
     7      print('2. Şart sağlandı')      if bloğu
     8      print ('2. if bloğunun içindesiniz')
     9  print('Normal program akışı girinti yok')
```

Şekil 4.3: *Python blok yapısı*

Şekil 4.3'te "if" ifadesinin altında girinti oluşturularak bir blok (1. Blok) oluşturulmuştur. "yasi>=18" mantıksal ifadesinin sonucu blok içindeki kodların tamamının atlanmasına veya çalıştırılmasına neden olur. Şekilde iki blok yapısı görünmektedir. Python'da kod yazarken girintilere dikkat etmek gerekir. Girintiler blok yapısını belirlediği için programın yanlış çalışmasına veya çalışmamasına neden olabilir.

Örnek 3

Aşağıda herhangi bir karar yapısı, döngü veya fonksiyon olmadan ikinci satırdaki kodu girintili yazıldığı için hata verir.

```
print('Blok yapısı')
   print('Girinti')
File "<ipython-input-7-b54d1a37c7f1>", line 2
    print('Girinti')
    ^
IndentationError: unexpected indent
```

4.4. if yapısı

Bu yapıda, belirli komutların çalışması, bir koşula bağlıdır. Koşul sağlanmazsa herhangi bir işlem yapılmaz.

Kullanımı: Aşağıda bir "if" bloğu gösterilmektedir "if" bloğunun dikey hizasının sağında olan kod satırları koşul gerçekleştiğinde çalışır. Bu kodlar "if" bloğunda yer almaktadır. Büyük eşittir operatörü karşılaştırma sonucu "boolean" (True veya False) bir değer verir. True değer verirse "if" bloğu içinde (girintide olan) kodlar çalışır. Koşul sağlanmazsa yani "False" değeri verirse bloğun içine girilmez bloktaki kodlar atlanır.

Örnek 4

Kullanıcının yaş değerini alarak 18'e eşit veya büyük olması hâlinde ona mesaj veren kod:

```
yasi=int(input('Lütfen yaşınızı giriniz: '))
if (yasi>=18):
  print('Oy kullanabilirsiniz.')
print ('Program bitti.')
Lütfen yaşınızı giriniz: 18
Oy kullanabilirsiniz.
Program bitti.
```

"if" bloğunun içindeki kod ancak şart sağlandığında çalışır ve blok bittikten sonra program akışı devam eder. Şart sağlanmazsa blok atlanır. Python alt satırdaki kodları yorumlar ve ona göre işlem yapar. Aşağıdaki örnek 18'den küçük bir yaş girilerek çalıştırılmıştır.

```
yasi=int(input('Lütfen yaşınızı giriniz: '))
if (yasi>=18):
 print('Oy kullanabilirsiniz.')
print ('Program bitti.')
Lütfen yaşınızı giriniz: 15
Program bitti.
```

Örnek 5

"adi" ve "yasi" değişkenlerinin değerlerine göre hangi blokların çalıştığına dikkat ediniz.

```
yasi=17
adi='Tahsin'
if (yasi>=18):
 print('1. Şart sağlandı.')
 print ('1. if bloğunun içindesiniz.')
if (adi=='Tahsin'):
 print('2. Şart sağlandı')
 print ('2. if bloğunun içindesiniz.')
print('Normal program akışı girinti yok.')
2. Şart sağlandı.
2. if bloğunun içindesiniz.
Normal program akışı girinti yok.
```

Koşul ifadelerinde birden fazla koşul birlikte kullanılabilir. Bunun için operatörler konusunda açıklanan mantıksal operatörler kullanılır. Örnekte "and" operatörü ile iki koşulun birlikte sağlanma şartı koşulmuştur.

Örnek 6

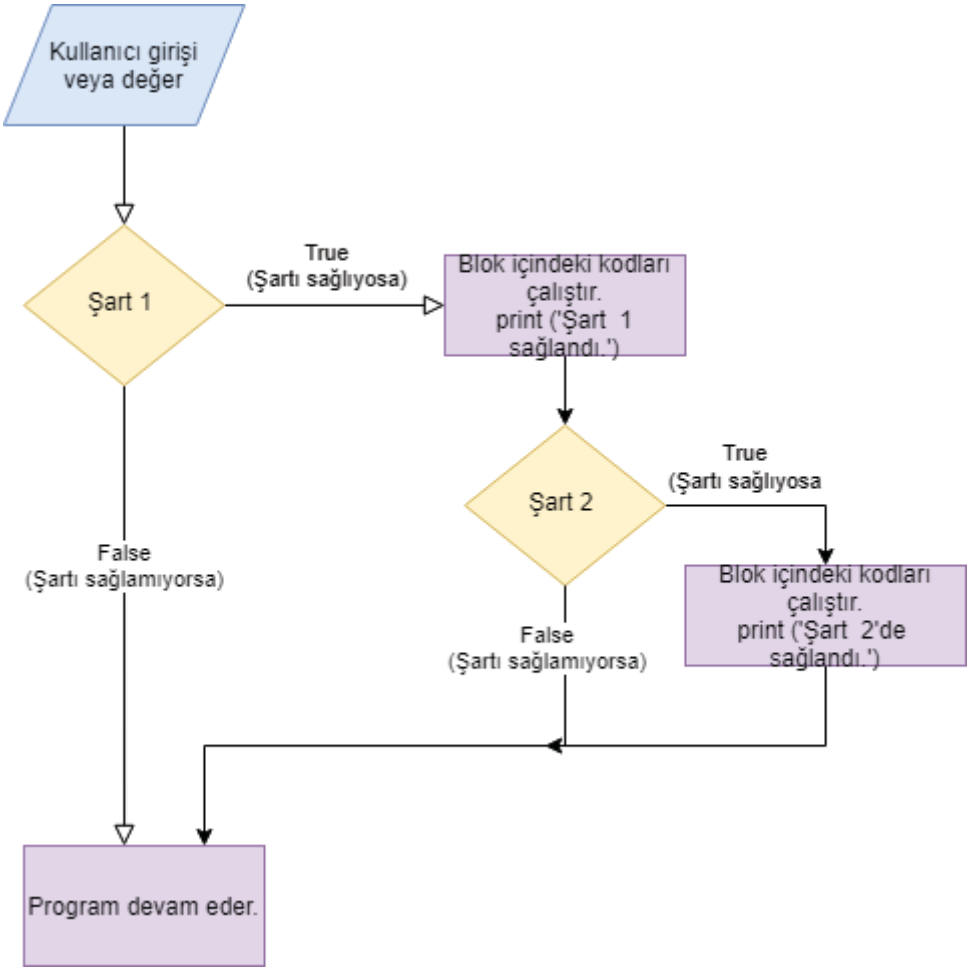
Kullanıcının girdiği kullanıcı adı ve parolayı kontrol ediniz.

```
kullaniciAdi=input('Kullanıcı Adı:')
kullaniciParola=input('Parola:')
if (kullaniciAdi=='Admin' and kullaniciParola=='123456'):
  print('Giriş başarılı.')
  print ('Menülere erişebilirsiniz.')
Kullanıcı Adı:Admin
Parola:123456
Giriş başarılı.
Menülere erişebilirsiniz.
```

Örnekte şartlardan biri bile sağlanmazsa "if" bloğundaki kodlar çalışmaz.

4.5. İç İçe Koşul İfadeleri

Yukarıdaki örnekte birden fazla koşulu "and" operatörünü kullanarak kontrol edilmiştir. Aynı işlem, iç içe koşul ifadeleri kullanarak da yapılabilir.



Şekil 4.4: *İç içe koşul ifadeleri*

Şekil 4.4'te görülen koşullu ifadelerden birincisinin yani ilk "if" bloğundaki şart sağlanırsa (Şart 1) o bloktaki kodların çalıştığı bilinmektedir. Bu şart ifadesinin içine bir koşul ifadesi yani ikinci bir "if" bloğu (Şart 2) daha eklenebilir. Şart 2'deki koşul da sağlanırsa bu kez "if" bloğunun içindeki ikinci "if"

bloğundaki kodlar da çalışır. İkinci koşul ifadesindeki şart sağlandığında birinci koşul ifadesindeki şart zaten sağlanmış olacağından (sağlanmasaydı ikinci "if" bloğu kodu birincinin içinde olduğu için zaten çalışmazdı) iki koşul ifadesi de (Şart 1 and Şart 2) sağlanmış olur. Koşul ifadelerinden birincisi sağlanır ikincisi sağlanamazsa ikinci "if" bloğundaki kodlar atlanır. Bu kullanım örnekteki "and" operatörü kullanımına benzemekle birlikte koşullardan sadece birincisinin sağlandığı durumlar için kodlar oluşturmaya olanak verir. İç içe koşul ifadelerinin sayısı (iç içe 3 koşul gibi) ihtiyaca göre artırılabilir.

Örnek 7

Kolay anlaşılması için verilen örnekteki ikinci "if" bloğu içindeki kodların girintisine dikkat ediniz.

```
kullaniciAdi=input('Kullanıcı Adı:')
kullaniciParola=input('Parola:')
if (kullaniciAdi=='Admin'):
 print('Kullanıcı adı doğru')
  if (kullaniciParola=='123456'):
    print('Giriş başarılı.')
    print ('Menülere erişebilirsiniz.')
Kullanıcı Adı:Admin
Parola:123456
Giriş başarılı.
Menülere erişebilirsiniz.
```

İlk şart yapısında kullanıcı adının doğru olup olmadığı kontrol edilmektedir. Eğer şart doğruysa içteki 2. şart bloğu çalışarak parola kontrolü yapacaktır. İki şart sağlanırsa ekrana tüm mesajlar yazdırılacaktır. Sadece 1. şart sağlanırsa kullanıcı adı doğru girilip parola yanlış girilirse "Kullanıcı adı doğru" mesajı ekranda görünecektir. Kullanıcı adı yanlış girilirse hiçbir mesaj görünmeyecektir.

4.6. if-else Yapısı

"if" yapısında şart sağlanırsa blok içindeki kodlar çalışmaktadır. Ancak şartın sağlanmadığı durumlarda herhangi bir işlem yapılmaz. "else" ifadesi değilse anlamındadır. Yani şartın sağlanmadığı durumda çalışacak kodlar "else" bloğuna yazılır.

Kullanımı: "else" bloğu da if bloğu gibi ayrı bir blok olarak yazılır. Bir "if" bloğundan sonra gelen else bloğu aynı girinti seviyesinde olmalıdır. "else" bloğu "if" ile birlikte kullanılır.

Örnek 8

Aşağıdaki örnekte kullanıcının girdiği sayının çift - tek olduğunu bulan bir kod yazılmıştır.

Çift bir sayı girildiğinde "if" bloğunun içindeki kodlar çalışır.

```
sayi1=int (input ('Lütfen bir sayı giriniz: '))
if ((sayi1%2)==0):
  print('Girdiğiniz sayı çifttir.')
else:
  print('Girdiğiniz sayı tektir: ')
Lütfen bir sayı giriniz: 12
Girdiğiniz sayı çifttir.
```

Tek bir sayı girildiğinde "else" bloğunun içindeki kodlar çalışır.

```
sayi1=int (input ('Lütfen bir sayı giriniz: '))
if ((sayi1%2)==0):
  print('Girdiğiniz sayı çifttir.')
else:
  print('Girdiğiniz sayı tektir: ')
Lütfen bir sayı giriniz: 13
Girdiğiniz sayı tektir:
```

Örnek 9

Koşullu ifade, operatörler ve bağlaçlarla daha etkili yapılabilir. Yukarıdaki kullanıcı adı ve parolası örneğinde kullanıcının girdiği kullanıcı adı ve parola bu örnekte "and" ile birlikte kontrol edilmiştir.

```
kullaniciAdi=input('Kullanıcı Adı:')
kullaniciParola=input('Parola:')
if (kullaniciAdi=='Admin' and kullaniciParola=='123456'):
  print('Giriş başarılı.')
  print ('Menülere erişebilirsiniz.')
else:
  print ('Yanlış kullanıcı adı veya şifre')
Kullanıcı Adı:Admin
Parola:123456
Giriş başarılı.
Menülere erişebilirsiniz.
```

Örnek 10

"if else" yapısına ilişkin başka bir örnek aşağıda verilmiştir. Bu örnekte şartlar "and" bağlacıyla birleştirilmiştir.

```
yasi=int(input('Lütfen yaşınızı giriniz: '))
bolum='Bilgisayar'
yabanciDil=True
if (yasi>=18 and yasi<35 and bolum=='Bilgisayar' and yabanciDil==True):
#Aşağıdaki kodun çalışması için yukarıdaki  3 şartın da sağlanması gerekir.
  print('Mülakata katılabilirsiniz.')
else:
  print('Şartlarınız tutmuyor.')
Lütfen yaşınızı giriniz: 20
Mülakata katılabilirsiniz.
```

4.7. if-elif-else Yapısı

Bu yapıda koşullar art arda verilir. if ile verilen koşulun devamında 'değilse şu ise' anlamına gelen "elif" ifadesi yer alır. Yapının en sonunda ise 'hiçbiri değilse' anlamında else ifadesi yer almaktadır. Her ifade kendi bloğundaki kodları çalıştırır. "if, elif ve else" bloklarının girinti düzeyleri aynı olmalıdır.

Her koşul ifadesi bir "if" bloğu formatında yazılabilirdi ancak bu durumda program akışında tüm koşul ifadeleri tek tek kontrol edilirdi. "if-elif-else" yapısında ise şart sağlandığında veya else ifadesine gelindiğinde ilgili bloktaki kodlar çalışır ve tüm "if-elif-else" bloğundan çıkılır. Birbirleriyle bağlantısı olmayan koşullar ayrı "if" blokları şeklinde verilebilir. Ama bir değerin belirli aralıktaki şartları sağlayıp sağlamadığı kontrol edilirken "if-elif-else" yapısını kullanmak daha uygundur. Bu yapıda koşullardan biri sağlanıyorsa diğer koşullar kontrol edilmez. Alınan değer "if-elif-else" yapısındaki yalnız bir koşulu sağlayabilir.

Örnek 11

Bir kullanıcının sınav puanını alarak durumunun değerlendirilmesi:

```
sinavPuani=int(input('Puanınız giriniz (0-100): '))
if sinavPuani>=85:
  print('Pek iyi')
elif sinavPuani>=70:
  print('İyi')
elif sinavPuani>=55:
  print('Orta')
elif sinavPuani>=45:
  print('geçer')
else: print('Kaldı')
Puanınız giriniz (0-100): 65
Orta
```

Örnek 12

Öğrencilerin başarı puanlarını hesaplayan ve sonucu değerlendirerek harf notu olarak döndüren bir program yazımı:

```
basariPuani=-1#kontrol değerimiz
vizePuani=int(input('Vize puanını giriniz: '))
finalPuani=int(input('Final puanını giriniz: '))
vizeOraniYuzde=(int(input('Vize oranını % olarak giriniz (30, 40 gibi): ')))
finalOraniYuzde=(int(input('Final oranını % olarak giriniz (70, 60 gibi): ')))
#Alınan verilerin kontrol edilmesi
if (vizePuani>100 or vizePuani<0 or finalPuani>100 or finalPuani<0 ):
  print ('Girdiğiniz vize veya final puanlarını kontrol ediniz.')
elif (vizeOraniYuzde+finalOraniYuzde)<100:
  print ('Girdiğiniz vize veya % oranlarını kontrol ediniz toplam 100 olmalıdır.')
else:
  basariPuani=(vizePuani* vizeOraniYuzde/100)+(finalPuani* finalOraniYuzde/100)
  #Bir hata yoksa basariPuani hesaplanır.
  #Hesaplanmazsa ilk verilen -1 değeri kalır.
if (basariPuani>=0):
  print('Başarı puanı: ', basariPuani )
  if (basariPuani>= 80 and basariPuani <= 100):
    print('Başarı notu : A')
  elif (basariPuani>= 70 and basariPuani <80):
    print('Başarı notu : B')
  elif (basariPuani>= 60 and basariPuani <70):
    print('Başarı notu : C')
  elif (basariPuani>= 50 and basariPuani <60):
    print('Başarı notu : D')
  elif (basariPuani< 50 ):
    print('Başarı notu : F')
Vize puanını giriniz: 70
Final puanını giriniz: 80
Vize oranını % olarak giriniz (30, 40 gibi): 30
Final oranını % olarak giriniz (70, 60 gibi): 70
Başarı puanı:  77.0
Başarı notu : B
```

4.8. Bölüm Sonu Örnekleri

1. Kullanıcının girdiği tam sayının "Negatif", "Pozitif" ya da "Sıfır" olduğunu yazdıran programın kodlarını yazınız.

2. Öğrencinin sınav ortalamalarını kullanıcıdan alan ortalama en az 50 ise geçti değilse kaldı yazan programı yapınız.

3. Kullanıcın girdiği iki sayıyı karşılaştırarak sayı sayıdan büyüktür, küçüktür veya sayılar eşittir mesajı veren kodları sadece "if" koşul yapısını kullanarak yazınız.

4. Yukarıdaki programı "if-elif-else" koşul yapısını kullanarak yazınız.

5. Aşağıdaki kodun çıktısı ne olur?

```
sayi1=12
sayi2=60
toplam=0
if sayi1<=sayi2:
  if sayi1%2==0:
    sayi1=sayi2
    toplam=sayi1+sayi2
  else: toplam=sayi2-sayi1
toplam+=toplam
print(toplam)
```

6. Yukarıdaki kodda sayi1=40 sayi2=13 değerleri için kodun çıktısı kaç olur?

7. Kullanıcıdan (1-4) arasında sayı alınacak, bu sayıya göre sırasıyla İlkbahar-yaz-sonbahar-kış yazan programın kodlarını yazınız.

8. Kullanıcıdan alınan dört kenar uzunluğuna göre şeklin kare, dikdörtgen veya diğer dörtgenlerden olduğunu belirten kodu yazınız.

9. Kenar uzunlukları girilen üçgenin çeşidini bulan programın kodlarını yazınız.

10. Beden kitle endeksini kilo/(boy**2) formülü ile hesaplanarak bireyin kilo durumunu kontrol eden programın kodlarını aşağıdaki aralık durumlarına göre yazınız.

```
Kitle Endeksi (KE) < 18.5 ise Zayıf,
18.5 < (KE) <=25 ise Normal,
25 < (KE) <= 30 ise Kilolu, (KE) > 25 ise birey obez sınıfına girmektedir.
```

Cevaplar

1.
```
sayi = input('Sayı : ')
if(int(sayi)<0):
  print("Sayı Negatif")
elif(int(sayi)>0):
  print("Sayı Pozitif")
else:
  print("Sayı Sıfır")
Sayı : -30
Sayı Negatif
```

2.
```
ort = input('Ortalamanızı Girin : ')
if(int(ort)>=50):
  print("Geçtiniz")
else:
  print("Kaldınız")
Ortalamanızı Girin : 60
Geçtiniz
```

3.
```
sayi1= int(input('1. sayıyı giriniz: '))
sayi2= int(input('2. sayıyı giriniz: '))
if sayi1>sayi2:
  print('1. sayı 2. sayıdan büyüktür.')
if sayi1<sayi2:
  print ('1.sayi 2. sayıdan küçüktür.')
if sayi1==sayi2:
  print ('Sayılar eşittir.')
1. sayıyı giriniz: 12
2. sayıyı giriniz: 24
1.sayi 2. sayıdan küçüktür.
```

4.
```
sayi1= int(input('1. sayıyı giriniz: '))
sayi2= int(input('2. sayıyı giriniz: '))
if sayi1>sayi2:
  print('1. sayı 2. sayıdan büyüktür.')
elif sayi1<sayi2:
  print ('1.sayi 2. sayıdan küçüktür.')
else: print ('Sayılar eşittir.')
1. sayıyı giriniz: 12
2. sayıyı giriniz: 24
1.sayi 2. sayıdan küçüktür
```

5. 240

6. 1

7.
```
a=int(input("Mevsim No:"))
if(a==1):
  print("İlkbahar")
elif(a==2):
  print("Yaz")
elif(a==3):
  print("Sonbahar")
elif(a==4):
  print("Kış")
else:  print("Aralıkta olmayan bir değer girdiniz")
Mevsim No: 2
Yaz
```

8.
```
a=int(input("1. kenar:"))
b=int(input("2. kenar:"))
c=int(input("3. kenar:"))
d=int(input("4. kenar:"))
if(a==b==c==d):
  print("Kare!")
elif(a==c and b==d or a==b and c==d ):
  print("Dikdörtgen")
else: print("Diğer Dörtgen")
1. kenar:6
2. kenar:7
3. kenar:8
4. kenar:9
Diğer Dörtgen
```

9.
```
a=int(input("1. kenar:"))
b=int(input("2. kenar:"))
c=int(input("3. kenar:"))
if(a!=b and a!=c and b!=c):
  print("Çeşitkenar Üçgen!")
elif(a==b==c):
  print("Eşkenar Üçgen!")
else: print("İkizkenar Üçgen")
1. kenar:3
2. kenar:5
3. kenar:6
Çeşitkenar Üçgen!
```

10.
```
boy = float(input("Boy: Örnek 1.73----:"))
kilo = float(input("Kilo: Örnek: 78.40----:"))
endeks  = kilo/(boy**2)
if endeks<18.5:
  print("Zayıfsınız")
elif endeks > 18.5 and endeks <=25 :
  print("Normalsiniz")
elif endeks > 25 and endeks <=30:
  print("Kilolusunuz")
elif endeks > 30:
  print("Dikkat! obez")
Boy: Örnek 1.73----:1.75
Kilo: Örnek: 78.40----:95
Dikkat! obez
```

MODÜL 5

LİSTELER VE ÖZELLİKLERİ

Şekil 5.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

Programlamada bir değişken üzerinde sadece bir değer tutulabilmektedir. Listeler ise bir değişkenin altında birden fazla değer tutulabilmesine yarar. Listeler, içinde **farklı türlerden verileri barındırabilen** taşıyıcılar olarak adlandırılmaktadır. Listeler Python'daki veri tiplerinden biridir. Listeler sıralı olarak kaydedilebilen veri yapılardır. Verilere döngü gibi yapılarla sıralı olarak erişmek istenildiğinde bize büyük avantaj sağlayıp, iki köşeli parantez arasında tanımlanırlar. Listelerde sözlüklerden farklı olarak dilimlenebilir ve elemanları sonradan değiştirilebilir. Aşağıda görüldüğü üzere boş liste ve int, string ve float verilerini bulunduran listeler tanımlanmıştır.

```
liste= []      #boş list
liste=list()
liste1=[1,2,3,4,5,6]
liste2=['a','b','c']
liste3=[0.5,1.7,67.89,3.14]
liste4=['ali','veli','yılmaz','hayri']
liste5 = list(ifade for degisken in sequence) # Hesaplanan bir liste
```

5.1. Liste Veri Tipleri

Bir listede her veri tipinden eleman saklanabilir. Bu anlamda sıralı bir diziye benzemektedir.

Örnek 1

```
liste=[1,2,'ali',0.25]
print(liste)
[1, 2, 'ali', 0.25]
```

Örnek 1'de int, string ve float gibi farklı veri tiplerini içerisinde barındıran 4 elemanlı bir listedir.

Karakter dizilerin indis değerleri değiştirilmek istendiğinde aşağıdaki gibi hata mesajı alınır.

```
meyve="erik"
meyve = 5 +meyve[0:]
print(meyve)
Traceback (most recent call last):
File "<ipython-input-9-e4575713155c>", line 2, in <module>
meyve = 5 +meyve[0:]
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Örnek 2

```
meyve ="erik"
meyve = "ayva-" +meyve[0:]
print(meyve)
liste=['a','b']
liste[1]=2
print(liste)
ayva-erik
['a', 2]
```

Örnek 2'de karakter dizisi ile liste kullanımı beraber verilmiştir. Karakter dizisi string ama liste 1.indis numarasındaki öge 'b' iken sonra int veri tipinde 2 olarak değiştirilmiştir.

Listeyi oluşturmak için önce bir değişken adı verilir, daha sonra bu değişkene çeşitli değerler atanır. Atamalar yapılarak liste oluşturulur. Bir liste oluşturulup bir değişkene atandığı zaman aslında tüm atamalarda olduğu gibi liste nesnesinin adresi değişkene atanmaktadır.

Örnek 3'te oluşturulan liste içine herhangi bir eleman eklenmediği için bu liste içi boş bir listedir. Çıktı olarak boş liste vermektedir.

Örnek 3

```
liste=[ ] #veya liste=list()
print(liste)
[ ]
```

5.2. Liste Kavramı ve İndis Değerleri

Öncelikle liste üzerinden veri okuyabilmek için hangi indis elemanının okunmak istendiği doğru bir şekilde belirtilmelidir. Örneğin Harfler isminde bir liste olduğunu düşünün. Listelerde ilk eleman her zaman 0. indistir. Listenin ilk elemanına erişmek veya yazmak istendiğinde Harfler[0] yazılması gerekmektedir. Diğer elemanları için de sırasıyla Harfler[1], Harfler[2] şeklinde yazılması gerekmektedir. Buradaki bir diğer önemli nokta da listenin ilk elemanı 0. indisten başladığı için son elemanı da liste uzunluğun bir eksiğidir.

Tablo 1. Listelerde indis numarasına göre veri sıralaması

Başlangıç			Bitiş(Uzunluk-1)
Harfler[0]	Harfler[1]	Harfler[2]	…

Örnek 4

```
renkler = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
print('listeyi ekrana yazdırıyoruz')
print(renkler)
listeyi ekrana yazdırıyoruz
['black', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
```

Örnek 4'te görüldüğü üzere renkler listesindeki tüm ögeleri listelenmiştir.

Python'da listeler üzerinde değişik türden veriler bir arada tutulabilir. Python'ı güçlü kılan özelliklerden biri olan listelerde, her bir eleman bir indis (indis) numarasına sahiptir. Bir listenin başlangıç indisi 0 (sıfır)'dır.

Tablo 2. İndis numarasına göre liste örneği

İndis numarası	Liste[0]	Liste[1]	Liste[2]	Liste[3]
Öge sıralaması	Birinci öge	İkinci öge	Üçüncü öge	Dördüncü öge
Liste =	'ali',	5,	3.14,	'ayşe'

Başlangıç elemanları belli olan bir listenin tanımlanması ve yazdırılması:

Örnek 5

```
#liste1 adında bir liste tanımladık. İçine verileri girdik
#bu verileri ekrana yazdırdık
liste1 = ['a','b','c','d','e','f']
print(liste1)
['a', 'b', 'c', 'd', 'e', 'f']
```

Örnek 5'te liste1 adında liste tanımlandı ve char(karakter) olarak ögeleri bulunmaktadır. İndis numarasına bakıldığı zaman; aşağıdaki şekilde ögeler vardır.

```
liste1[0] ='a'      liste1[1] ='b'      liste1[2] ='c'       liste[3] ='d'
liste1[4] ='e'      liste1[5] ='f'
```

Listeler hem string hem de sayı türünde veriler barındırabilir. Örnek 6 incelendiğinde 7 elemanlı bir liste tanımlıdır. 6 ve 7. elamanları sayı diğerleri metin olan bir listedir.

Örnek 6

```
liste2=['python','geleceği','olan','bir','dil',500,0.567]
print(liste2)
['python', 'geleceği', 'olan', 'bir', 'dil', 500,0.567]
```

Örnek 6'da sayı, float ve string ögelerine sahip liste ekrana yazdırılmıştır.

5.3. Liste Elemanlarına Erişim

Listedeki ögelere ulaşmak için bir değer, listenin indis numarasına göre atanıp çağrılabilir. Liste oluştururken içine herhangi bir değer girilmeden de oluşturulup, sonradan değer ataması yapılabilir ya da herhangi bir indis numarasından başlanıp belirlenen indis numarasına kadar olan ögelere ulaşılır. Aşağıdaki örneklerde görüldüğü gibi hem indis numarasından hem de başlangıcı ve bitişi belli olan indis numarası aralıklarına kadar ifadelere erişilmiştir. Listenin belirli aralıktaki öğelerini alma işlemine dilimleme denir. Liste dilimlerken adımlama da başlangıç indisi alır ama bitiş indisi almaz.

Kullanımı:

```
eleman = liste[indis]
dilim = liste[baslangic:bitis]
```

Liste, elemanlarına erişim için tamamını veya indis numarasına göre çağırmaktadır.

Örnek 7

```
liste=["birinici veri","ikinci veri","üçüncü veri ","dördüncü veri","beşinci veri"]
#beş elemenlı listenin ilk verisi
print(liste[0])
#beş elemenlı listenin son verisi
print(liste[4])
birinici veri
beşinci veri
```

Örnek 7'de listenin indis numaraları yazılarak, başlangıç ve bitiş ögelerine ulaşılmıştır.

Örnek 8

```
liste=["ayva","armut","kiraz","vişne"]
print(liste)
print(liste[1])
['ayva', 'armut', 'kiraz', 'vişne']
armut
```

Örnek 8'de ise listenin tüm ögeleri ve 1. indis numarasındaki öge olan "armut" değeri listelenmiştir.

Liste içindeki ögeleri kontrol edilebilir. Eşya adında bir liste var. Liste tanımlanır, daha sonra if karar yapısı ile içindeki ögeler de "perde" olup olmadığı kontrol edilir.

Örnek 9

```
esya = ["ayna", "televizyon", "perde"]
if("perde" in esya):
     print("Bu değer listede var.")
else:
     print("Bu değer listede yok")
Bu değer listede var.
```

Aşağıdaki örnekte 5 elemanlı bir liste tanımlanmıştır. İndis numaralarını kullanarak, listenin 1. indisten başlayarak, 3. indise kadar öge listesine aktarıp listeleme işlemi yapılmıştır.

```
liste = [1,2,3,4,5]
öge = listem[1:3]
print(öge)
[2,3]
```

Örnek 10

```
listem=[10,20,30,40,50]
eleman = listem[3]
print(eleman)
40
```

Örnek 10'da listem adında bir liste tanımlanarak 5 elemanlı ifade girilmiştir. Listem adlı liste için 3. indis numarasına ait ifade eleman adlı değişkene atama işlemi yapılmıştır. Eleman adlı değişken de ekrana yazdırılarak 40 sonucu elde edilmiştir.

Örnek 11

```
listem = [10,20,30,40,50]
eleman = listem[1:3]
print(eleman)
[20, 30]
```

Örnek 11'de listem adlı listede eleman adlı değişkene sadece 1 ve 2. indis numaralarının dâhil edildiği 3. indis numarasının dâhil edilmediği atama işlemi yapılmıştır. Eleman adlı değişken de ekrana yazdırılarak 20 ve 30 değerleri listelenmiştir.

Örnek 12

```
liste = [1,2,3,4,5,6,7,8,9,10]
print(liste)
# 1. eleman
print(liste[0])
# 6. eleman
print(liste[5])
# Baştan 5. indekse kadar (dahil değil)
print(liste[:5])
# 1.indisten 5.indise kadar
print(liste[1:7])
print(liste[5:])
print(liste[::2])
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
1
7
[1, 2, 3, 4, 5]
[2, 3, 4, 5, 6, 7]
[6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
```

Örnek 12'de, 10 elemanlı bir liste tanımlanmış ve indis numaraları kullanılarak ekrana listeleme işlemi yapılmıştır.

Listelerde **negatif indisler** de kullanılabilir. Negatif indis numarası listenin sonuncu elemanından başlayarak sayıldığında (sondan başa) sıra numarasını verir. Örnek 13'te listenin sonuncu elemanına liste[-1] olarak, sondan ikinciye ise [-2] olarak ulaşılır. Güle güle elemanı liste de -1.indis olarak görülmektedir. Merhaba ise geriye doğru sayıldığında -4. indis'tir.

Örnek 13

```
liste = [ "merhaba", "dünya", "merhaba", "güle güle" ]
print (liste [- 1 ]) #son ögeyi listeler
print( liste [- 3 ]) #sondan üçüncü ögeyi listeler
print(liste [- 4 ]) #sondan dördüncü ögeyi listeler
print(liste[::-1]) #sondan başa doğru listeleme yapmak için kullanılır
güle güle
dünya
merhaba
['güle güle', 'merhaba', 'dünya', 'merhaba']
```

5.4. Temel Liste Metotları

Metotlar listelerin işlevlerine erişilmesini sağlar. Listenin metotları için dir( ) fonksiyonunu kullanarak tüm metotlar görülebilir. Bu metotlar yardımıyla listeler ekleme, çıkarma, arama, sıralama vb. birçok işlemin kolaylıkla yapabilmesini sağlamaktadır.

Tablo 3. Temel liste metotları

Sıra No	Metot Adı	Görevi
1	'append'	Listeye yeni eleman ekleme işlemini yapar. Bu metot ile listeye sadece bir eleman eklenebilir ve eklenen eleman listenin sonunda yer alır.
2	'clear'	Listeyi değil içindeki tüm ifadeleri silmeye yarar.
3	'copy'	Listeden listeye kopyalama işlevine yaramaktadır.
4	'count'	Listenin içinde sorgulanan elemandan kaç adet olduğunu bulmamızı sağlar.
5	'extend'	Listeler arası genişletme işlevini görür.
6	'indis'	Listedeki elemanları almamızı sağlar.
7	'insert'	Listenin istenilen indis numarasına eleman eklenebilir.
8	'pop'	Listedeki elemanın indisi ile silme işlem yapar. indis belirtmediğinizde ise varsayılan olarak listenin son elemanını siler. Ayrıca bu metot silinen elemanı ekrana yazmaktadır.
9	'remove'	Listede istenilen elemanın değerini yazarak silme işlemi yarar.
10	'sort'	Listenin elemanlarını alfabetik olarak sıralar.
11	'reverse'	Bu metot sort metodunun aksine listedeki elemanları ters alfabetik olarak sıralar.
12	'del'	Liste içerisinden bir elemanı silmek için kullanılır. Silme işlemi indis numarasına göre yapılmaktadır.

5.4.1. 'append' kullanımı

Örnek 14

```
takimlar=["gs","fb","bjk"]
takimlar.append("ts")
print(takimlar)
['gs', 'fb', 'bjk', 'ts']
```

Örnek 14'te takimlar listesine "ts" ögesi eklenmiş ve son indis numarasında yer almıştır.

5.4.2. 'insert' kullanımı

Örnek 15

```
sebzeler =["lahana","marul","pırasa","ıspanak","fasulye"]
sebzeler.insert(2,"patlıcan")
print(sebzeler)
['lahana', 'marul', 'patlıcan', 'pırasa', 'ıspanak', 'fasulye']
```

Örnek 15'te insert metodu kullanılarak 2. indis numarasına **"patlıcan"** ögesi eklenerek, listeleme işlemi yapılmıştır.

5.4.3. 'copy' kullanımı

Örnek 16

```
iller1 =["konya","karaman","kocaeli","kayseri","kahramanmaraş"]
iller2=[]
iller2 = iller1.copy()
print(iller2)
['konya', 'karaman', 'kocaeli', 'kayseri', 'kahramanmaraş']
```

Örnek 16'da iller1 listesi copy metodu ile iller2 listesine aktarılmıştır.

5.4.4. 'count' kullanımı

Örnek 17

```
takimlar = ['GS','FB','BJK','TS']
print(takimlar.count('FB'))
1
```

Örnek 17'de takimlar listesinde **'FB'** ögesinin kaç adet olduğu count metodu bulunmuştur.

5.4.5. 'extend' kullanımı

Örnek 18

```
kus1=["bıldırcın","papağan","kartal","akbaba","şahin"]
kus2=["baykuş","muhabbet"]
kus1.extend(kus2)
print(kus1)
['bıldırcın', 'papağan', 'kartal', 'akbaba', 'şahin', 'baykuş', 'muhabbet']
```

Örnek 18'de extend komutu listelerdeki ögelerin kendi elemanlarını koruyarak genişletme işlemi yapılmıştır.

5.4.6. 'indis' kullanımı

Örnek 19

```
sebzeler =["lahana","marul","pırasa","ıspanak","fasulye"]
print(sebzeler.indis("ıspanak"))
3
```

İndis metodu yardımıyla Örnek 19'da görüldüğü gibi verilen bir ögenin indis numarasını vermektedir.

5.4.7. 'clear' kullanımı

Örnek 20

```
liste =["ayva","nar","kiraz","kayısı","Üzüm"]
liste.clear()
print(liste)
[]
```

clear( ) metodu kullanılarak örnek 20'deki listenin tüm ögeleri silinmiştir.

5.4.8. 'pop' kullanımı

Örnek 21

```
sebzeler =["lahana","marul","pırasa","ıspanak","fasulye"]
sebzeler.pop(2)
print(sebzeler)
['lahana', 'marul', 'ıspanak', 'fasulye']
```

Örnek 21'de pop metodu ile sebzeler listesinden 2.indis numarasına ait olan **"pırasa"** adlı öge silinmiştir.

5.4.9. 'remove' kullanımı

Örnek 22

```
sehirler =["adana","ağrı","bursa","konya","ankara"]
sehirler.remove("konya")
print(sehirler)
['adana', 'ağrı', 'bursa', 'ankara']
```

Örnek 22'de öge adına göre silme işlemi yapılmıştır ve **"konya"** adlı öge sehirler listesinden silinmiştir.

5.4.10. 'reverse' kullanımı

Örnek 23

```
sayilar=[10,20,30,40,50,60,70]
sayilar.reverse()
print(sayilar)
[70, 60, 50, 40, 30, 20, 10]
```

Örnek 23'te reverse metodu ile liste öge elemanları tersten sıralanmıştır.

5.4.11. 'sort' kullanımı

Örnek 24

```
isimler=["elif","ayşe","kemal","kaan","hafsa"]
isimler.sort()
print(isimler)
['ayşe', 'elif', 'hafsa', 'kaan', 'kemal']
```

Sort metodu ile Örnek 24'te isimler listesi ögeleri alfabetik olarak sıralanmıştır.

5.4.12. 'del' kullanımı

Örnek 25

```
takimlar = ['GS','FB','BJK','TS']
del takimlar[2]
print(takimlar)
['GS', 'FB', 'TS']
```

Örnek 25'te del metodu ile takimlar listesine ait **'BJK'** ögesi indis numarasına göre silinmiştir.

Yukarıdaki örneklerde temel liste metotlarının her birine yönelik örnek ve çıktıları verilmiştir. Ayrıca __xxx__ şeklinde özel metotlar da bulunmaktadır. Bu metotlarda **dir(list)** şeklinde komut satırına yazıldığında aşağıdaki şekilde çıktı alınır.

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy',
'count', 'extend', 'indis', 'insert', 'pop',  'remove', 'reverse', 'sort']
```

5.5. Len() Fonksiyonu ile Uzunluk Bilgisi

len( ) fonksiyonu, İngilizce length'in (uzunluk) kısaltılmış hâlidir. String ifadesinin uzunluğunu yani karakter sayısını verir. Örnek 14'te a adlı string değişkene değer atadığımızda değişkendeki karakterlerin sayısını vermektedir.

Örnek 26

```
a="Galatasaray"
print(len(a))
11
```

Örnek 26'da "len( )" kullanımına bakıldığında, takimlar adında bir liste tanımlanmış. Bu listeye veri girişi yapılmıştır. Len( ) komutu ile listenin adı yazılarak, kaç elemanlı olduğu ekrana yazdırılmıştır. Örnek 27'de len( ) fonksiyonu ile takimlar listesinin eleman listesi 4 olarak verilmiştir.

Örnek 27

```
takimlar = ['GS','FB','BJK','TS']
print( len(takimlar))
4
```

Örnek 28'de 2 adet liste1 ve liste2 adında liste tanımlanmıştır. Bu listelere elemanlar girilerek, len() komutu ile kaç elemanlı olduğu bulunmuştur.

Örnek 28

```
liste1, liste2 = ['abc',56,74 ,'python'], [12, 'opencv','a']
print ("İlk liste uzunlugu            : ", len(liste1))
print( "İkinci listenin uzunluğu : ", len(liste2))
İlk liste uzunlugu           : 4
İkinci listenin uzunluğu     : 3
```

5.6. İç İçe Listeler

Bir liste herhangi bir sıralama nesnesi içerebilir, hatta başka bir liste (alt liste) içerebilir, alt listeler de alt listeler içerebilir ve bu şekilde devam eder. Bu yuvalanmış liste olarak bilinir. Hiyerarşik yapılara veri düzenlemek için bunlar kullanılabilir.

Örnek 29

```
liste1 = [1,2,3]
liste2 = [4,5,6]
liste3 = [7,8,9]
yeniliste = [liste1,liste2,liste3]
print(yeniliste)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Örnek 29'da 3 adet liste oluşturuldu. Bu listelerin her biri ayrı öge olacak şekilde birleştirilerek iç içe liste oluşturuldu ve ekrana yazdırıldı.

Örnek 30'da sebzeler adında boş şekilde liste oluşturuldu.

Örnek 30

```
sebzeler=[]
sebzeler.append(['yeşil','ıspanak'])
sebzeler.append(['beyaz','lahana'])
sebzeler.append(['turuncu','havuç'])
sebzeler.append(['siyah','turp'])
sebzeler.append(['kırmızı','domates'])
```

Append( ) metodunu kullanarak sebzeler adlı listeye 5 adet eleman girildi. Girilen verilerin listelenmesi:

```
print(sebzeler)
[['yeşil', 'ıspanak'], ['beyaz', 'lahana'], ['turuncu', 'havuç'], ['siyah', 'turp'],
['kırmızı', 'domates']]
```

Sebzeler adında listede 4 adet eleman bulunmaktadır. Bunları teker teker sıralanırsa:

```
print(sebzeler[0])
['yeşil', 'ıspanak']
```

Tablo 4. *Sebzeler listesi elemanları*

Liste adı	sebzeler				
Eleman değeri	'yeşil', 'ıspanak'	'beyaz', 'lahana'	'turuncu', 'havuç'	'siyah', 'turp'	'kırmızı', 'domates'
İndis numarası	0	1	2	3	4

Örnek 31

```
sebzeler=[['yeşil','ıspanak'],['beyaz','lahana'],['turuncu','havuç']]
sebze=sebzeler[1]
print(sebze)
['beyaz', 'lahana']
```

Örnek 31'e bakıldığı zaman, sebze değişkenine sadece sebzeler [1] matrisindeki değer atanmıştır. Tablo 4'te sebzeler listesinde sadece havuç değeri listelenmek isteniyorsa indis numarasından faydalanılır.

Örnek 32

```
sebzeler=[['yeşil','ıspanak'],['beyaz','lahana'],['turuncu','havuç']]
print(sebzeler[2][1])
havuç
```

Örnek 33'te 4 adet liste oluşturulmuştur. İlk üç listenin tüm ögeleri son_liste adında listeye aktarılarak, ekrana yazdırılmıştır.

Örnek 33

```
# 3 Adet liste oluşturalım.
birinci_liste = [1,2,3]
ikinci_liste = ['a','b','c']
ucuncu_liste= [40,50,60]
son_liste= [birinci_liste,ikinci_liste,ucuncu_liste]
print(son_liste)
[[1, 2, 3], ['a', 'b', 'c'], [40, 50, 60]]
```

Örnek 34'te 3 adet liste oluşturulmuştur. Bunlara birinci_liste, ikinci_liste, ucuncu_liste isimleri verilmiştir. Son_liste adında liste oluşturularak diğer üç listenin elemanları bu listeye kaydedilmiştir. Ekrana sadece a ve 50 değerlerini yan yana listelemek için aşağıdaki kodlar kullanılır.

Örnek 34

```
# 3 Adet liste oluşturalım.
birinci_liste = [1,2,3]
ikinci_liste = ['a','b','c']
ucuncu_liste= [40,50,60]
son_liste= [birinci_liste,ikinci_liste,ucuncu_liste]
print(son_liste[1][0],son_liste[2][1])
a 50
```

5.7. Veri Tipi Dönüşümleri

Listelerde veri tipi dönüşümleri için, elemanlara yeni değer ataması yapıldığında string, int, float vb. veri tipleri arasında değer alabilir.

Örnek 35

```
liste=[1,2,3,4,5,'ankara']
print(liste)
liste[0]=str("kocaeli")
liste[2]=float(1.5)
liste[5]=int(20)
print(liste)
[1,2,3,4,5,'ankara']
['kocaeli', 2, 1.5, 4, 5, 20]
```

Örnek 35'te liste adında bir liste örneği tanımlanmıştır. İçindeki elemanları 1,2,3,4,5,'ankara' 'dır. İndis numarasına göre liste [0] değeri önce 1 iken veri tipi dönüşümünden dolayı 'kocaeli' olmuştur. Liste [2]'in değeri ise ilk başta 3'tür. Daha sonra float veri tipinde 1.5 değerini almıştır. Son olarak liste[5]'in değeri 'ankara'dır. Liste[5]'e int veri tipinde bir değer kaydedilerek 20 olmuştur.

Herhangi bir string türünde veriyi parçalayarak da liste oluşturulabilir. Örnek 36'da string veri türünde adı meyve olan bir değişken olarak tanımlanmış ve veri olarak elmayı atanmıştır. Meyve adlı değişken liste yardımıyla parçalanmıştır.

Örnek 36

```
meyve="elma"
liste=list(meyve)
print (liste)
['e', 'l', 'm', 'a']
```

Örnek 37'de 1 ile 15 arasındaki sayılardan oluşan liste oluşturulmuştur. Listenin ögeleri ekrana listelenmiştir. Ekrana listeleme işleminde sort( ) metodu ile ögeler küçükten büyüğe, reverse ile büyükten küçüğe doğru sıralanmıştır.

Örnek 37

```
liste=list(range(1,15,2))
print(liste)
liste.sort()
print(liste)
liste.reverse()
print(liste)
[1, 3, 5, 7, 9, 11, 13]
[1, 3, 5, 7, 9, 11, 13]
[13, 11, 9, 7, 5, 3, 1]
```

split( ) metodu, listeyi belirtilen ayıracı kullanarak yeniden döndürmeye yarar. Yani split( ) karakter dizilerini istenen şekilde böler. -ayırıcı diye tanımladığımız ilk parametre, karakter dizisinin nereden bölüneceğini seçer. Eğer ayırıcı tanımlanmazsa karakter dizisi her boşluk gördüğünde ayırır. Örnek 38'de bilgiler girildikçe listeye kaydedecektir. Listeye 4 adet öge girilmiş ve listeleme işlemi yapılmıştır.

Örnek 38

```
bilgi=input("bilgilerinizi araya virgül koyarak yazınız: ")
liste=bilgi.split(",")
print(liste)
bilgilerinizi araya virgül koyarak yazınız: Hafsa,Meva,Konya,1
['Hafsa', 'Meva', 'Konya', '1']
```

Örnek 39'da ise cümle değişkenindeki kelimeler split metodu ile kelimeler listesine aktarılmıştır. Len( ) metodu ile de kaç adet öge olduğu ekrana listelenmiştir.

Örnek 39

```
cumle="23 nisan herkese mutlu olsun"
kelimeler=cumle.split(" ")
print("cümlenizde ",len(kelimeler),"adet kelime vardır")
cümlenizde 5 adet kelime vardır
```

5.8. Bölüm Sonu Örnekleri

1. Harfler adıyla bir liste oluşturup içine 'a', 'e', 'i', 'o', 'i', 'u' elemanları kaydediniz. Bu listede i ve p harflerinin sayısını ekrana yazdırınız.

2. Liste1, liste2, liste3 ve liste4 adında dört adet liste oluşturup aynı satırda olacak şekilde tanımlayıp, her bir listeye birer adet eleman girip listeleyiniz.

3. İki adet liste tanımlayarak bu iki listeyi "+" operatörü ile toplama işlemi yaptırılıp, üçüncü bir listeye atama işlemini yapınız. Son listeyi ekrana yazdırınız.

4. Aşağıdaki listeyi hem küçükten büyüğe hem de büyükten küçüğe doğru sıralanacak şekilde ekrana listeleyiniz.

```
liste = [34,1,56,334,23,2,3,19]
```

5. Aşağıdaki kodları verilen liste örneğinde ekran çıktısını yazınız.

```
listem= ["Merhaba", "Türkiye", "Nasılsın", "Tebrikler"]
print(listem[-1])
print(listem[-3])
print(listem[-4])
```

6. Aşağıdaki örnekte liste tanımlanmış ve ekran çıktısı verilmiştir. Ekran çıktısının aşağıdaki gibi olması için boş kalan kısımları tamamlayınız.

```
liste = [1, 2, 3, 4, 5, 6, 7]
.....
.....
.....
.....
[2, 3]
[1, 2, 3]
[4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

7. Aşağıda verilen örnekte boş kalan kısımları doldurunuz.

```
isimler = ['ali','veli','ayşe']
.....
ad_soy1 = isimler[0] +' '+ soyisimler[0]
.....
ad_soy3 = isimler[2] +' '+ soyisimler[2]
print(ad_soy1)
.....
print(ad_soy3)
ali türk
veli izci
ayşe erel
```

8.
```
liste = ['bir','iki','dört']
print(liste)
```

`['bir', 'iki', 'dört']` şeklinde ekran çıktısı olmaktadır. Ama `['bir', 'iki', 'üç', 'dört', 'beş']` olacak şekilde listeyi güncelleyiniz.

9. Aşağıdaki listede listenin ilk ve son verilerine ulaşmak ve listelemek için gerekli kodları yazınız.

```
liste=["birinci veri", "ikinci veri", "üçüncü veri ", "dördüncü veri",
"beşinci veri"]
```

Cevaplar

1. Aşağıdaki örnekte harfler adında bir liste tanımlanmış ve 6 adet eleman girilmiştir. count metodu kullanılarak, i ve p harflerinin sayısı ekrana yazdırılmıştır. İ harfi için 2 ve p harfi için 0 değeri listelenmiştir.

```
harfler = ['a', 'e', 'i', 'o', 'i', 'u']
count = harfler.count('i')
print('i harflerinin sayısı:', count)
count = harfler.count('p')
print('p harflerinin sayısı:', count)
i harflerinin sayısı: 2
p harflerinin sayısı: 0
```

2. Aşağıdaki örnekte 4 adet liste tanımlanmıştır. Bu listelere elemanlar atanarak listelenmiştir.

```
liste1,liste2,liste3,liste4= ['a',100,3.14,'python']
print(liste1)
print(liste2)
print(liste3)
print(liste4)
a
100
3.14
python
```

3.

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
['a', 'b', 'c', 1, 2, 3]
```

4.
```
liste = [34,1,56,334,23,2,3,19]
liste.sort()
print('küçükten büyüğe doğru',liste)
liste.reverse()
print('büyükten küçüğe doğru',liste)
```
```
küçükten büyüğe doğru [1, 2, 3, 19, 23, 34, 56, 334]
büyükten küçüğe doğru [334, 56, 34, 23, 19, 3, 2, 1]
```

5.
```
listem= ["Merhaba", "Türkiye", "Nasılsın", "Tebrikler"]
print(listem[-1])
print(listem[-3])
print(listem[-4])
```
```
Tebrikler
Türkiye
Merhaba
```

6.
```
liste = [1, 2, 3, 4, 5, 6, 7]
print(liste[1:3])
print(liste[:3])
print(liste[3:])
print(liste[:])
```
```
[2, 3]
[1, 2, 3]
[4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

7.
```
isimler = ['ali','veli','ayşe']
soyisimler = ['türk','izci','erel']
ad_soy1 = isimler[0] +' '+ soyisimler[0]
ad_soy2 = isimler[1] +' '+ soyisimler[1]
ad_soy3 = isimler[2] +' '+ soyisimler[2]
print(ad_soy1)
print(ad_soy2)
print(ad_soy3)
ali türk
veli izci
ayşe erel
```

8.
```
liste = ['bir','iki','dört']
liste[2]='üç'
liste.insert(3,'dört')
liste.insert(4,'beş')
print(liste)
['bir', 'iki', 'üç', 'dört', 'beş']
```

9.
```
liste=["birinci veri","ikinci veri","üçüncü veri ","dördüncü veri","beşinci
veri"]
#beş elemanlı listenin ilk verisi
print(liste[0])
#beş elemanlı listenin son verisi
print(liste[4])
birinci veri
beşinci veri
```

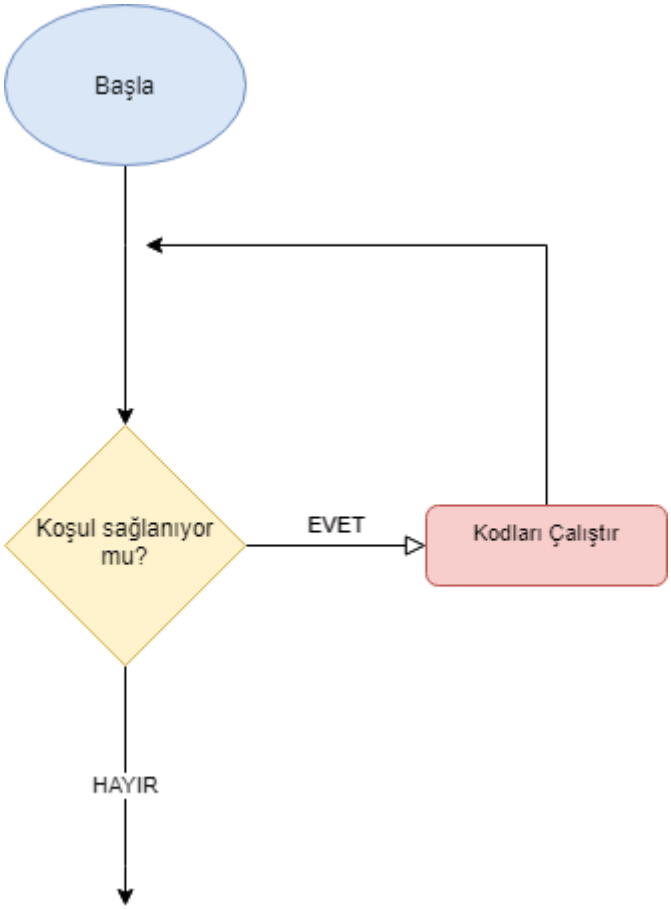
MODÜL 6

DÖNGÜ YAPILARI

Şekil 6.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

Bu bölüme kadar yapılan uygulamalar tek seferde çalışmaktadır. Örnek olarak 50 sayının ortalamasını almak istendiğinde, 50 sayıyı kullanıcıdan alarak her birini bir değişkene atamak, sonra da bu sayıların ortalamasının bulunması gerekiyordu. 50 sayı değil de bin adet sayının ortalamasını bulmak istersek bu iş daha da zorlaşacaktır.

Döngüler, istenen kodların belirli sayı veya koşullar sağlandığı sürece tekrar tekrar çalıştırılması temeline dayanır. Burada tekrarlama işlemi belirli sayıda olursa **for döngü yapısı**, belirli koşullara bağlı tekrar söz konusu ise **while döngü yapısı** tercih edilir. Örnek verilecek olursa her sabah güneş doğar ve her akşam güneş batar. Bu işlem süreklilik arz etmektedir.



Şekil 6.2: *Döngü Yapısı*

Yukarıdaki şekilde koşul sağlandığı sürece döngü devam edecektir. Ne zaman ki koşul şartı gerçekleşmezse o durumda döngüden çıkılacaktır.

Python'da **while** ve **for** döngüleri olmak üzere iki tür döngü bulunur.

6.1. While Döngüsü

While döngüsü, koşul gerçekleştiği sürece çalışan bir döngü çeşididir. Genellikle döngünün kaç defa çalışacağı belirli değilse while döngüsü tercih edilir. Ancak koşullar verilerek de while döngüsünün belirli sayıda çalışması sağlanabilir. Döngülerde koşullu ifadelerde olduğu gibi blok yapısı kullanılmaktadır. while ifadesinden sonra koşul durumu yazılır, ardından iki nokta işareti konularak alt satıra geçilir. Koşul durumu sağlandığı sürece çalışacak kodlar bir blok içeriden çalışır.

while (koşul durumu):

1. adım
2. adım
3. adım

.

.

Bu durum bir örnekle incelenmek istenirse,

Örnek 1

```
# şartın başlangıç değeri
sayac=1
#sayac 6 dan küçük olduğu sürece
while sayac<6:
    print("merhaba dünya")
    sayac=sayac+1
merhaba dünya
merhaba dünya
merhaba dünya
merhaba dünya
merhaba dünya
```

Örnek 1'de sayaç isimli değişkenin değeri 1'den başlamış, yine değişken değeri 6'dan küçük olduğu sürece konsol üzerinde "merhaba dünya" yazılır. Sayacın değeri 1 arttırılarak döngünün başına döner ve sayaç değeri 6'ya eşit olana kadar bu durum devam eder. Aynı işlemi ekrana sayaç isimli değişkenin değeri yazılarak yapmak istenirse,

Örnek 2

```
sayac=1
while sayac<6:
    print(sayac)
    sayac=sayac+1
1
2
3
4
5
```

6.1.1. Döngünün Kapsamı

Hatırlanacağı üzere döngü koşulunun sağlandığı sürece daha içteki bloklarda bulunan kodların çalışacağı belirtilmişti. Döngü bittiği zaman Python bir üstteki bloğa dönerek çalışmasına devam eder.

Örnek 3

```
# şartın başlangıç değeri
sayac=1
#sayac 6 dan küçük olduğu sürece
while sayac<6:
    print(sayac)
    sayac=sayac+1
#döngü bittiği zaman
print("döngü sonlandı")
1
2
3
4
5
döngü sonlandı
```

while döngüsü ile çalışırken sık yapılan hatalardan birisi döngü içerisinde koşulu sağlayan değişkenin değerini arttırma işleminin unutulmasıdır. Bu durumda koşul sürekli sağlanacağı için döngü sürekli çalışır ve dışarıdan bir müdahale ile sonlandırılması gerekir.

Örnek 4

```
sayac=1
while sayac<6:
    print(sayac)
```

Yukarıdaki örnekte bilinçli olarak sayaç isimli değişkenin değeri arttırılmamıştır. Uygulamayı çalıştırıldığında görüleceği üzere program hiç durmadan çalışacak ve ekrana sürekli 1 değeri yazacaktır. Bu işlem sırasında klavyeden "ctrl + c" tuşuna basarak Python'a kesme gönderebilir ve uygulama sonlandırılabilir.

Örnek 5

while döngüsü kullanarak 1-100 arasındaki (100 dâhil) çift sayıları bularak ekrana yan yana yazan programı yazınız.

```
a=1
while a<=100:
    if a%2==0:
        print(a,end=",")
    a=a+1
2,4,6,8…..,100
```

Örnek 6

while döngüsü kullanarak 1 – 100 arasındaki sayıların toplamını bulan programı yazınız.

```
toplam=0
i=1
while i<=100:
    toplam=toplam+i
    i=i+1
print("sayıların toplamı",toplam)
sayıların toplamı 5050
```

Örnek 6'da yapılan işlemde toplam isimli bir değişken oluşturularak başlangıç değeri 0 olarak belirlenmiştir. Çünkü toplam değerini hesaplarken toplam=toplam+i şeklinde bir işlem yapılmaktadır. Eğer toplam tanımlı olmasaydı, tanımlanmamış_değer=tanımlanmamış_değer+sayı şeklinde bir işlem yapılmaya çalışacak ve hata verecekti. Yani bir değişken ile işlem yapılmadan önce tanımlanması gerekmektedir.

6.1.2. While True – Break İfadeleri ve Sonsuz Döngüler

Program yazarken bazen döngünün ne zaman sonlanacağı bilinmeyebilir. Örnek olarak bir markette müşterilerin alışveriş yaparak sepetlerini doldurdukları ve sepette kaç adet ürün olduğu bilinmeyebilir. O müşteriye ait tüm ürünler barkod okuyucu ile okutulmalı ve toplam tutar hesaplanmalıdır. İşte bu gibi belirsiz durumlar için while döngüsü ile beraber **True** ifadesi ya da benzer yapılar kullanılabilir. **Break** ifadesi ise döngü sürekli çalışırken istenilen bir anda döngüden çıkmak için kullanılır.

Örnek 7

```
i=1
while True:
    print(i)
    i+=1
    if i==6:
        break
print("döngü sonlandı")
1
2
3
4
5
döngü sonlandı
```

while döngüsü ile kullanılan True ifadesinin yerine farklı kullanımlarla da karşılaşılabilir. Örneğin, bir alışveriş yapıldığını ve sürekli ürün girişi yapıldığını düşünülürse "q" harfi girilene kadar yapılan alışverişler listeye eklensin, eğer "q" harfi girilirse döngüyü sonlandırılsın.

Örnek 8

```
liste=[]
while 1:
    ürün=input("ürün adı giriniz:")
    if ürün=="q":
        break
    liste.append(ürün)
print("girdiğiniz meyveler:",liste)
ürün adı giriniz:elma
ürün adı giriniz:armut
ürün adı giriniz:peynir
ürün adı giriniz:su
ürün adı giriniz:q
girdiğiniz meyveler: ['elma', 'armut', 'peynir', 'su']
```

Örnek 9

while döngüsü kullanarak sayı tahmin oyunu yapınız. Kullanıcıdan 1-100 arası bir sayı istenmektedir. Girilen sayı, tahmin edilen sayıdan büyükse daha küçük bir sayı girmesi, büyükse daha küçük bir sayı girmesi istensin. Kullanıcı sayıyı bulana kadar bu işlem tekrar etsin. Ayrıca bir sayaç eklenerek kaç defa da tahmin ettiğini bulunuz.

```
Sayi=45
sayaç=0
print("1-100 arası bir sayı tuttum tahmin et")
while 1==1:
    sayaç+=1
    cevap=int(input("1-100 arası bir sayı girin: "))
    if cevap>sayi:
        print("daha küçük bir sayı girmelisin")
    elif cevap<sayi:
        print("daha büyük bir sayı girmelisin")
```

```
    else:
        print("tebrikler tuttuğum sayıyı bildin")
        break
print("tebrikler {} seferde sayıyı bulabildin".format(sayaç))
1-100 arası bir sayı tuttum tahmin et
1-100 arası bir sayı girin: 50
daha küçük bir sayı girmelisin
1-100 arası bir sayı girin: 40
daha büyük bir sayı girmelisin
1-100 arası bir sayı girin: 45
tebrikler tuttuğum sayıyı bildin
tebrikler 3 seferde sayıyı bulabildin
```

Yine burada **while True** yapısına benzer bir yapı kullanarak while 1==1: şeklinde bir yapı ile sonsuz döngü oluşturulmuştur.

Örnek 10

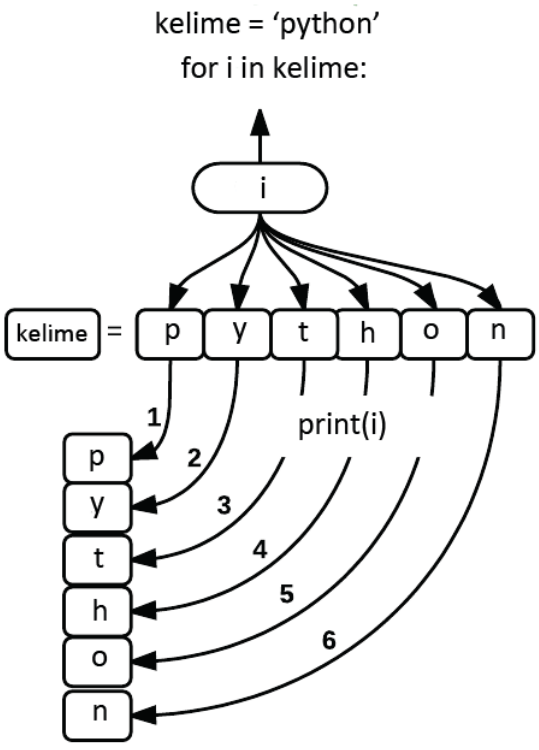
Girilen sayının faktöriyelini hesaplayan programı yazınız.

```
i=1
f=int(input("faktöriyeli alınacak sayıyı giriniz: "))
sonuc=1
while i<=f:
    sonuc=sonuc*i
    i+=1
print(sonuc)
faktöriyeli alınacak sayıyı giriniz: 5
120
```

Örnek 10'da i isimli değişkeni 1 değerinden başlayarak while döngüsünü i değeri girilen sayıdan küçük olduğu sürece çalıştırılmış, daha sonra i değeri arttırarak sonuç isimli değişkenle çarpma işlemi yapılmış, i değeri f değerine eşit olduğunda döngüden çıkılarak, sonuç isimli değişkenin değeri gösterilmiştir.

6.2. For Döngüsü

For döngüsü, Python'da genellikle döngünün **tekrar sayısı programcı tarafından belirlenmiş veya öngörülmüş ve** belli ise kullanılır. Hatırlanacağı üzere **while** döngüsü ile sonsuz döngüler yapılabiliyor ve istenilen bir anda döngüden çıkılabiliyordu. **For** döngüsü daha çok belirli sayıdaki işlemi gerçekleştirmek için kullanılır. Bunun yanında Python'da for döngüsünün **iterasyon** denilen önemli bir özelliği bulunmaktadır. **İterasyon** işlemi sayesinde karakter dizileri ve listeler üzerinde gezinme işlemi, yani ilk elemandan son elemana kadar işlem yapabilmektedir. **For** döngüsü kullanmak için **"in"** işlecinden faydalanmak gerekmektedir.



Şekil 6.3: İterasyon işlemi

6.2.1. in İşleci

in işleci bir değerin, bir liste ya da karakter dizisi içerisinde olup olmadığını kontrol eder. Önce karakter dizisi içinde bir karakter olup olmadığına bakar. Eğer değer karakter dizisi içerisinde varsa **True**, yoksa **False** değeri döndürecektir. Aşağıdaki kodları etkileşimli kabuk üzerinde çalıştırabilirsiniz.

Örneğin:

"p" in "python"

```
>>> True
```

"a" in "python"

```
>>> False
```

aynı işlemi listeler üzerinde de yapılabilir.

```
"a" in "python"
>>> False
liste=[1,2,3,4,5,6]
3 in liste
>>> True
10 in liste
>>> False
```

6.2.2. Karakter Dizileri Üzerinde İterasyon İşlemi

Python'da **for** döngüsü ile karakter dizileri üzerinde kolaylıkla iterasyon işlemi yapılabilir.

Örnek 11

```
isim="Mustafa"
for i in isim:
    print(i,end=",")
M,u,s,t,a,f,a,
```

Örnek 11'de yapılan işlemle isim adlı değişken üzerinde ilk karakterden son karaktere kadar tüm değerleri hiçbir harf kalmayana kadar sırayla i değişkenine atayarak ekrana yazdırılmıştır.

Aynı işlemi while döngüsü ile yapılmak istenirse,

Örnek 12

```
isim="Mustafa"
i=0
while i<len(isim):
    print(isim[i],end=",")
    i=i+1
M,u,s,t,a,f,a,
```

Örnek 12'de **while** döngüsü ile karakter dizileri üzerinde işlem yapılmak istendiğinde hem daha fazla kod yazılması hem de daha karışık bir yapı kullanılması gerekmektedir. Python'da genellikle listeler veya karakter dizileri üzerinde işlem yapılmak istenildiği zaman yani iterasyon yapılacağı zaman **for** döngüsü kullanılmaktadır.

Örnek 13

Bir cümle içerisinde geçen bir harfin kaç defa geçtiğini bulunuz.

```
yazi="Python üst düzey basit sözdizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği desktekeyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
harf="a"
sayac=0
for i in yazi:
    if i=="a":
        sayac=sayac+1
print("cümle içerisinde geçen a harfi sayısı: ",sayac)
cümle içerisinde geçen a harfi sayısı:  9
```

Örnek 14

Bir cümle içerisinde geçen sesli harfleri bulan programı yazınız.

```
yazi="Python üst düzey basit söz dizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
sesli="aeıioöuü"
for i in yazi:
  if i in sesli:
    print(i,end=",")
```

```
o,ü,ü,e,a,i,ö,i,i,i,e,a,i,ö,e,e,i,o,u,a,o,a,o,ü,e,i,i,o,u,a,i,i,i,i,e,e,e,e,a,o,a,ı,
ı,e,e,ö,e,i,i,o,u,a,a,i,i,i,i,i,i,i,
```

Örnek 15

İki farklı karakter dizisi belirleyerek, birinci de olup, diğerinde olmayan karakterleri bulunuz.

```
cumle1="Python üst düzey basit sözdizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
cumle2=" Python interaktif yani etkileşimli bir programlama dilidir."
for i in cumle2:
    if not i in cumle1:
        print(i,end=",")
```

```
ş,ğ,
```

Örnek 16

Kullanıcı tarafından girilen bir karakter dizisi içerisinde geçen sesli ve sessiz harfleri ayrı ayrı listelere atayan programı yazınız.

```
sesli_harfler = "aeıioöuü"
sessiz_harfler = "bcçdfgğhjklmnprsştvyz"
sesliler=""
sessizler=""
a=input("bir metin giriniz")
for i in a:
    if i in sesli_harfler:
        sesliler=sesliler+i
    if i in sessiz_harfler:
        sessizler=sessizler+i
print("sesli harfler",sesliler)
print("sessiz harfler",sessizler)
bir metin giriniz Python üst düzey basit söz dizimine sahip, öğrenmesi oldukça
kolay, modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir.
sesli harfler oüüeaiöiiieaiöeeiouaoaoüeiiouaiiiieeeeaoaııeeöeiiouaaiiiiiii
sessiz harfler ythnstdzybstszdzmnshpğrnmsldkçklymdlrlğknblrlğdsktkynpltfrmbğmsznsnynl
mlyrmlnblrbrscrptdldr
```

6.2.3. Listeler Üzerinde İterasyon İşlemi

Python'da karakter dizilerinde yapılan işlem gibi listeler üzerinde de iterasyon işlemi yapılabilir. Örnek olarak elimizde bir liste olduğunu ve içerisinde sayısal ifadeler (integer tanımlanmış) olduğunu düşünelim. Bu değeri toplayabilir, ortalamasını bulabilir ya da farklı bir listeye atayabilirsiniz. Kullanım şekli itibariyle karakter dizilerinde yapılan işlemlerin aynısı yapılabilir.

Örnek 17

İçerisinde sayısal değerler olan bir listedeki değerlerin karesini alarak başka bir listeye atayınız.

```
sayılar = [1,2,3,4,5]
kareler = []
for i in sayılar:
    kareler.append(i*i)
print(kareler)
[1, 4, 9, 16, 25]
```

Örnek 18

Bir liste içerisinde bulunan değerlerin ortalamasını bulun.

```
sayılar = [1,2,3,4,5,6,7,8,9]
toplam=0
for i in sayılar:
    toplam=toplam+i
print("sayıların ortalaması: ",toplam/len(sayılar))
sayıların ortalaması:  5.0
```

Örnek 19

Listedeki değerlerden 3'ün katları olan sayıları ekrana yazdırın.

```
sayılar = [5,8,12,17,25,36,41,49,60,72]
for i in sayılar:
    if i%3==0:
        print(i,end=",")
12,36,60,72,
```

Örnek 20

İç içe listeler üzerinde gezinme işlemi yapılabilir. [[3,4],[7,8],[10,11],[14,15]] şeklinde bir liste olduğunu varsayalım. Liste elemanları üzerinde klasik bir şekilde gezinme işlemi yapılmak istenirse:

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i in liste:
    print(i,end=",")
 [3, 4],[7, 8],[10, 11],[14, 15],
```

Listeye erişildi ancak alt listelere erişilebilmesi için aşağıdaki gibi bir yapının kullanılması gerekmektedir.

Örnek 21

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i,j in liste:
    print(i,end=",")
    print(j,end=",")
3,4,7,8,10,11,14,15,
```

Örnek 22

Ya da iç içe for döngüsü kullanarak aynı işlem gerçekleştirilebilir.

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i in liste:
    for j in i:
        print(j,end=",")
3,4,7,8,10,11,14,15,
```

6.2.4. range Fonksiyonu ile For Döngüsü Kullanımı

Python'da **for** döngüsüyle belirli değerler arasında döngü kurmak istenirse **range** fonksiyonu kullanılmalıdır. **range** fonksiyonu bir sayı dizisi oluşturur ve bu sayede oluşturulan sayı dizisi üzerinde **for** döngüsünün iterasyon yapması sağlanır.

Örnek 23

```
print(*range(10))
0 1 2 3 4 5 6 7 8 9
```

range fonksiyonu, girilen aralık arasında integer değerler oluşturur. Örnek 23'te aralık belirtilmediği için başlangıç değeri 0 alınmıştır. Başlangıç değeri verilerek girilen değerler arasında sayı dizisi oluşturulması sağlanabilir.

Örnek 24

```
print(*range(5,20))
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Ayrıca range fonksiyonuna üçüncü bir parametre verilerek atlama değeri de verilebilir.

Örnek 25

```
print(*range(1,20,3))
1 4 7 10 13 16 19
```

Örnek 26

for döngüsü ile girilen sayıya kadar olan sayıların toplamını bulunuz.

```
toplam=0
for i in range(20):
    toplam=toplam+i
print("girdiğiniz sayıların toplamı:",toplam)
girdiğiniz sayıların toplamı: 190
```

Örnek 27

20'den geriye doğru 0'a kadar olan sayıları ekrana yazdırınız.

```
for i in range(20,0,-1):
    print(i,end=",")
20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,
```

Örnek 28

100'e kadar 5'in katları olan sayıyı bulunuz.

```
for i in range(0,100,5):
    print(i,end=",")
0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,
```

Örnek 29

Kullanıcıdan satır ve sütun sayısı değerlerini alarak sayıları tablo şeklinde yazan programı yazınız.

```
a=int(input("tablonun satır uzunluğunu giriniz"))
b=int(input("tablonun sütun uzunluğu giriniz"))
for i in range(1,a+1):
     for j in range(1,b+1):
         print(j,end=" ")
     print()
tablonun satır uzunluğunu giriniz5
tablonun sütun uzunluğu giriniz4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

Örnek 29

Girilen bir metindeki sesli harf, sayı ve özel karakterlerin sayısını bulan programı yazınız.

```
sesli_harfler=["a","e","ı","i","o","ö","u","ü"]
rakamlar="1234567890"
ozel_harf=["@","!","&","?"]
ozel_harf_sayısı,rakam_sayısı,sesli_sayısı=0,0,0
kelime=input("lütfen incelemek için bir metin giriniz: ")
for harf in kelime:
    if harf in sesli_harfler:
        sesli_sayısı+=1
    if harf in rakamlar:
        rakam_sayısı+=1
    if harf in ozel_harf:
        ozel_harf_sayısı+=1
```

```
print("girdiğiniz metinde {} adet sesli harf {} adet rakam ve {} adet özel karakter bulunmaktadır. ".format(sesli_sayısı,rakam_sayısı,ozel_harf_sayısı) )
lütfen incelemek için bir metin giriniz: girilen metindeki sesli harf sayı ve özel karakterli bulan program123456!@&
girdiğiniz metinde 23 adet sesli harf 6 adet rakam ve 3 adet özel karakter bulunmaktadır.
```

6.3. Continue İfadesi

Hatırlanacağı üzere **break** ifadesi döngünün dışına çıkılmasını sağlamaktadır. Döngülerde kullanılan **continue** ifadesi, döngünün baştan sona kadar çalışmasını engellemeyen ancak belirli durumlar sağlandığında o adımı atlamamızı sağlayan yapılardır. Döngü sona ermez ancak verilen koşulun sağlanması durumunda döngüyü direk başa alır.

Örnek 30

```
for i in range(1,6):
    if i ==2 or i==4:
        continue
    print(i)
1
3
5
```

Görüldüğü üzere Python, 2 veya 4 değerlerini görünce döngünün başına gitmiş ve alt satırdaki ifadeler çalıştırılmamıştır.

Örnek 31

Aynı işlem **while** döngüsü ile yapılmak istenirse,

```
i=0
while i < 5:
    i=i+1
    if i == 2 or i==4:
        continue
    print(i)
1
3
5
```

while döngüsü ile **continue** deyimi kullanırken döngü, sonsuz döngüye girebilir. Eğer i=i+1 ifadesi **continue** deyiminin altında kullanılırsa i değeri sürekli 2 olarak kalır. Değişkenin değeri artmaz ve uygulama sonsuz döngüye girer. Örnek 32'de hatalı kullanım şekli gösterilmiştir.

Örnek 32

```
i=0
while i < 5:
    if i == 2 or i==4:
        continue
     i=i+1
    print(i)
```

6.4. İç İçe Döngüler

Döngü bloklarının içerisine kodlar eklenebileceği gibi, koşul durumları hatta başka bir döngü koymak bile mümkündür. Bir döngü yapısının içine başka bir döngü yapısının yerleştirilmesi ile elde edilen yapıya **iç içe döngü (nested loop)** adı verilir.

Örnek 33

```
dersler = ["Ders 1", "Ders 2", "Ders 3"]
konular = ["Konu 1", "Konu 2", "Konu 3"]
for x in dersler:
    for y in konular:
        print(x, y)
Ders 1 Konu 1
Ders 1 Konu 2
Ders 1 Konu 3
Ders 2 Konu 1
Ders 2 Konu 2
Ders 2 Konu 3
Ders 3 Konu 1
Ders 3 Konu 2
Ders 3 Konu 3
```

Örnek 34

İç içe **while** döngüsü kullanarak, i ve j olarak tanımlanan iki değişkenin değerlerini ekrana yazdırınız.

```
i=1
while i<4:
    j=1
    while j<4:
        print("i nin değeri: {} j nin değeri: {} ".format(i,j))
        j=j+1
    i=i+1
i nin değeri: 1 j nin değeri: 1
i nin değeri: 1 j nin değeri: 2
i nin değeri: 1 j nin değeri: 3
i nin değeri: 2 j nin değeri: 1
i nin değeri: 2 j nin değeri: 2
i nin değeri: 2 j nin değeri: 3
i nin değeri: 3 j nin değeri: 1
i nin değeri: 3 j nin değeri: 2
i nin değeri: 3 j nin değeri: 3
```

6.5. Bölüm Sonu Örnekleri

1. Girilen sayının asal olup olmadığını bulan programı yazınız.

2. Kullanıcıdan değer alarak ağaç şekli çizen programı yazınız.

3. Bir listede bulunan sayıların en büyüğünü ve en küçüğünü bulan programı yazınız.

4. Elinizde bulunan iki adet listeyi birleştirerek 3. oluşturan programı yazınız.

5. Bir cümledeki boşlukları kaldıran programı yazınız.

Cevaplar

1.
```
a=int(input("bir sayı giriniz"))
asal=0
for i in range (2,a):
    if a%i==0:
        asal+=1
if asal==0:
    print("girdiğiniz sayı asal")
else:
    print("girdiğiniz sayı asal değil")
```

2.
```
a=int(input("ağacın yükseliği"))
b=a
for i in range(1,a+1):
    print(b*" ",(2*i-1)*"*")
    b-=1
```

3.
```
liste=[5,9,7,1,2,3,6,4]
buyuk=0
kucuk=999
for i in range(len(liste)):
    if liste[i]>buyuk:
        buyuk=liste[i]
    if liste[i]<kucuk:
        kucuk=liste[i]
print("""girilen sayıların en büyüğü= {}
girilen sayıların en küçüğü= {}""".format(buyuk,kucuk))
```

4.
```
liste1=[1,2,3,4,5,6,7]
liste2=["python","java","c","c++","c#","pascal","cobol"]
liste3=[]
for i in range(len(liste1)):
    liste3.append((liste1[i],liste2[i]))
print(liste3)
```

5.
```
a="m e r h a b a"
for i in a:
    if i!=" ":
      print(i,end="")
```

MODÜL 7

FONKSİYONLAR, GLOBAL VE LOKAL DEĞİŞKENLER

Şekil 7.1: Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz

7.1. Fonksiyon Kullanımı

Bu bölüme kadar kullanılan fonksiyonlardan bazıları şunlardır: print( ), input( ), int( ) ve len( )

Fonksiyonlar, matematikteki fonksiyonlarla aynı işlevi görürler. y=f(x)=x*x gibi bir fonksiyon tanımlandığında verilen değeri kendisiyle çarpar. Bu görece kolay bir işlemdir. Bir sayının karesini hesaplamak gerektiğinde, formülü tekrar yazmak yerine bu fonksiyon kullanılabilir. Fonksiyonlar; kodları tekrar yazmayı ortadan kaldırmak, kodları daha iyi organize etmek sonrasında bu kodları rahat bir şekilde kullanmak için oluşturulur.

Python'da ve diğer programlama dillerinde birçok fonksiyon çeşitli kütüphanelerde veya varsayılan olarak tanımlı gelir. Bu sayede en temel işlemler için bile satırlarca kodu tekrarlamaya gerek kalmaz.

Fonksiyonların görevi, karmaşık işlemleri bir araya toplayarak bu işlemleri tek adımda yapmayı sağlamaktır. Fonksiyonları kullanarak bir veya birkaç adımdan oluşan işlemleri tek bir yapı altında toplamak mümkündür.

Fonksiyonun Adı	Parametreler - Argümanlar
print	(“Sultan”, “Murat”, sep=“ ”, end=“\n”)
input	(“Argüman”)

Her fonksiyonun bir adı (print, input, len vb.) ve işlevini yerine getirmesi için kullanabileceği parametre adı verilen yapıları vardır. Parametreler fonksiyonda tanımlı özellik ve girdileri belirtir. Bir fonksiyonu kullanırken (çağırırken) bu parametrelere verilen değerlere “argüman” denir. Aşağıda print( ) fonksiyonu ve parametrelerinin kullanımına örnek verilmiştir. “end” ve “sep” birer parametredir. Fonksiyonlarda bazı parametreler zorunludur ve bir argüman (değer) girmek gerekir. Ancak yine bazı parametreler fonksiyon içinde ön tanımlı olarak bir değer/boş (None) veya seçmeli olarak belirlenmiştir. Bu parametrelere değer vermek gerekmez. Bunlar, isteğe bağlıdır ve kullanılmayacak özellikler için argüman girmeye gerek yoktur.

Örnek 1

Aşağıdaki örnekte print ( ) fonksiyonun “end” ve “sep” argümanlarıyla birlikte kullanımı gösterilmiştir:

```
print ('Merhaba', 'Mars', sep=' ', end='\n') #parametreleri kullanarak
print ('Merhaba')
print ('Merhaba',' ', 'Mars') #parametreleri kullanmadan
print ('Merhaba')
Merhaba Mars
Merhaba
Merhaba   Mars
Merhaba
```

7.2. Fonksiyon Oluşturma

Temel fonksiyonlar, Python içinde hazır olarak yer alan fonksiyonlardır. Bu fonksiyonlar tanımlı olduğu için sadece fonksiyonadı( ) yazarak kullanabilmektedir.

Bir fonksiyon def fonksiyonAdi(parametre1, parametre2,..):şeklinde tanımlanır. Fonksiyon içindeki kodlar girinti şeklinde blok yapısında yazılır. Fonksiyon yapısı dışında o fonksiyonu kullanmak için fonksiyonAdi(argüman1, argüman2) şeklinde parametre değerleri verilerek kullanılır. Örneğin, girilen bir sayının çift bir sayı olup olmadığını bulan bir fonksiyon yazabilirsiniz. Fonksiyon bir parametreden oluşacak ve bir sayı değeri alacaktır.

Örnek 2

Bir sayının çift olup olmadığını ekrana yazan bir fonksiyon.

Hatırlatma: Bir sayının 2'ye bölümünden kalan yoksa o sayı çifttir varsa tektir.

```
def sayiCiftMi (sayi):
  if sayi%2==0:
    print('Sayı çifttir')
  else: ('Sayı tektir')
```

Bir fonksiyonu oluşturduktan sonra print( ) fonksiyonunda kullandığınız şekilde parametrelere uygun argümanları girip çağırabilirsiniz.

```
sayiCiftMi(10)
Sayı çifttir
```

Fonksiyonun adı ve parametrelerine değerleri yazılarak fonksiyon kullanılabilir. Fonksiyon çağrıldığında verilen argümanlara göre işlem yapar. Örnekteki fonksiyon 10 sayısını kontrol ederek sonucu "Sayı çifttir" şeklinde ekrana yazdırır.

Örnek 3

Başka bir fonksiyon tanımlayabilirsiniz. Tanımlanan fonksiyon bir metni istenildiği kadar alt alta yazdırır. Bu fonksiyonda yazdırılacak metin ve yazdırılma sayısı olmak üzere iki adet parametre olacaktır.

```
def yazdir(metin,kacKere):
  for i in range (1, (kacKere+1)):
    print (metin, end='\n')
#Fonksiyon çağırma
yazdir('Merhaba', 5)
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
```

Aynı fonksiyon için parametreleri kullanıcıdan alabilirsiniz.

```
yazdirilacakMetin=input('Yazdırılacak metni giriniz: ')
yazdirmaSayisi=int(input('Metin kaç kez yazdırılacak: '))
yazdir(yazdirilacakMetin, yazdirmaSayisi)
Yazdırılacak metni giriniz: Kara Murat
Metin kaç kez yazdırılacak: 4
Kara Murat
Kara Murat
Kara Murat
Kara Murat
```

Örnek 4

Bir sayının asal bir sayı olup olmadığını bulan bir fonksiyon yazabilirsiniz. Fonksiyon, sayı asal ise "True"; değilse "False" değerini döndürecektir.

NOT

"for" döngüsünde sayının yarısına kadar bakmamızın nedeni bir sayının kendi değerinin yarısından önce böleni yok ise sonrasında da olmayacağı kuralıdır.

```
def asalMi(sayi):
  sayac=2 # tüm sayılar 1'e bölüneceğinden 2 ile başlattık
  while sayac<=int(sayi/2):
    if sayi%sayac==0:
      return False
    sayac+=1
  return True
#Fonksiyonu çağırma
asalMi(113)
True
```

Örnek 5

Verilen sayının faktöriyelini bulan bir fonksiyon tanımlayabilirsiniz. Bir sayının faktöriyeli kendisinden başlayarak 1'e kadar olan tüm sayıların çarpımıdır. 3!=3*2*1)

```
def faktoriyelAl(sayi):
  sonuc=1
  if  (sayi==0 or sayi==1):
    print('sonuç= ', 1)
  elif sayi>1:
    for i in range(1, sayi+1, 1):
      sonuc*=i
    print ('sonuc=', sonuc)
  else: print ('0 veya daha büyük sayısal bir değer girmelisiniz')
```

Yukarıda tanımlanan fonksiyonu çağırmak için kullanıcıdan argüman almanız gerekmektedir.

```
faktoriyelAl(int(input('Faktöriyeli alınacak sayıyı giriniz: ')))
Faktöriyeli alınacak sayıyı giriniz: 5
sonuc= 120
```

7.3. Fonksiyonlarda Parametre Türleri

"faktoriyelAl" olarak tanımlanan fonksiyon "sayi" parametresiyle çalışmaktadır. Argüman olarak bir sayı verildiğinde sayının faktöriyelini hesaplamaktadır. Fonksiyonlar yaptıkları işlemlere göre farklı parametre türlerini kullanır.

Örnek 6

Döngülerde örnek olarak verilen ağaç çizme kodlarını bir fonksiyon hâline getirebilirsiniz.

```
def agacCiz(agacinYuksekligi, karakter='*'):
  b=agacinYuksekligi
  for i in range(1,agacinYuksekligi+1):
    print(b*' ',(2*i-1)*karakter)
    b-=1
```

Şimdi, fonksiyonu kullanıcıdan aldığınız parametrelerle çağırabilirsiniz.

```
agacYuksekligi=int(input("Ağacın yüksekliği kaç satır olsun? : "))
agacKarakteri=input("Ağaç için bir sembol veya karakter girin? : ")
if agacKarakteri!='' and agacYuksekligi>=1:
  agacCiz(agacYuksekligi, agacKarakteri[0])
elif agacKarakteri=='' and agacYuksekligi>=1:
  agacCiz(agacYuksekligi)
else: print('Hatalı giriş')
```

```
Ağacın yüksekliği kaç satır olsun? : 10
Ağaç için bir sembol veya karakter giriniz :
           *
          ***
         *****
        *******
       *********
      ***********
     *************
    ***************
   *****************
  *******************
```

Yukarıda bir "agacCiz" fonksiyonu tanımlanmıştır. Bu fonksiyonun "agacinyuksekligi" ve karakter adlarında iki adet parametresi bulunmaktadır. "agacinYuksekligi" ağacın satır sayısını belirten bir parametredir. Bu parametreye bir argüman (değer) verilmezse fonksiyon hata verir. Böyle parametrelere "zorunlu parametre" denir. Karakter parametresi ise parametre olarak tanımlanırken bir değerle birlikte tanımlanmıştır, bu değer parametrenin varsayılan değeridir. Varsayılan değer olarak "None" boş değer verilebilir. Zorunlu parametreler dışında fonksiyonlarda kullanılan diğer parametreler için varsayılan değerler tanımlanabilir. Eğer kullanıcı karakter olarak bir giriş yapmazsa yani boş bırakırsa fonksiyon ağacı varsayılan olarak "*" karakterini kullanarak oluşturur.

7.4. Değer Döndüren ve Döndürmeyen Fonksiyonlar

Python'da fonksiyonlardan bazıları sadece bir işlevi yerine getirir ve görevi orada biter. Ama bazı fonksiyonlar bir değer döndürür. "int( )" fonksiyonu verilen argümanı sayıya çevirerek döndürür. Yukarıdaki örneklerde "faktoriyelAl" ve "agacCiz" fonksiyonları bir değer döndürmemektedir. Değer döndüren fonksiyonlarda "return" ifadesi yer alır. Bir fonksiyonun sadece ekrana yazı yazdırması değer döndürdüğü anlamına gelmez. Eğer bir fonksiyonun çıktısı uygun bir değişkene atanabiliyorsa bu fonksiyon değer döndüren bir fonksiyondur.

Örnek 7

Örnekte kullanılan faktoriyelAl() fonksiyonunu değer döndüren bir fonksiyon hâline getirebilirsiniz.

```
def faktoriyelAl(sayi):
  sonuc=1
  if  (sayi==0 or sayi==1):
    sonuc=1
  elif sayi>1:
    for i in range(1, sayi+1, 1):
      sonuc*=i
  else: sonuc=-1 #hatalı bir işlem olduğunu anlamak için -1 değerini veriyoruz
  return sonuc
```

Değer döndüren fonksiyonu aşağıdaki gibi çağırabilirsiniz.

```
sonucumuz=faktoriyelAl(5)
#fonksiyonu bir değişkene atayabiliyoruz.
if sonucumuz!=-1: # bir hata olup olmadığını kontrol edelim
  print(sonucumuz)
else:print('Bir hata oluştu')
120
```

Kendi tanımladığınız fonksiyonların içinde temel fonksiyonları ya da tanımladığınız başka fonksiyonları da kullanabilirsiniz.

Örnek 8

Kullanıcıdan liste olarak aldığı sayıların ortalamasını bulan bir fonksiyon tanımlayabilir ve bunun içinde len( ) fonksiyonunu çağırabilirsiniz.

```
def ortalamaBul(sayilar):
  sayilarinToplami=0
  sayilarinOrtalamasi=0
  for i in range(len(sayilar)):
    sayilarinToplami+=sayilar[i]
  sayilarinOrtalamasi=sayilarinToplami/len(sayilar)
  return sayilarinOrtalamasi
```

Sayıları kullanıcıdan alarak fonksiyonunuzu çağırabilirsiniz.

```
sayiAdedi=int(input('Kaç adet sayının ortalamasını alacaksınız: '))
sayilarim=[]
for i in range(0,sayiAdedi):
  print (i+1, '. sayıyı giriniz?')
  # indis sıfırdan başladığı için +1 kullandık
  sayi=int(input())
  sayilarim.append(sayi)
ortalamaBul(sayilarim)
Kaç adet sayının ortalamasını alacaksınız: 4
1 . sayıyı giriniz?
10
2 . sayıyı giriniz?
20
3 . sayıyı giriniz?
30
4 . sayıyı giriniz?
40
25.0
```

NOT

Fonksiyonlar bir fonksiyon da dâhil olmak üzere değer olarak her türlü veri tipini döndürebilirler.

7.5. Global ve Lokal Değişkenler

Python'da blok yapısından ve girintilerden söz etmiştik. Python'da tanımlanan ve değer atanan değişkenler tanımlandıkları blok içinde geçerlidir. Bir değişken aynı düzeydeki veya daha içerideki girintilerde de değerini korur.

Örnek 9

```
for i in range (1,3):
  print ('i değişkenin değeri=', i)
print ('i değişkenin son değeri=', i)
i değişkenin değeri= 1
i değişkenin değeri= 2
i değişkenin son değeri= 2
```

"for" bloğu en dışta yer aldığı için i değeri onunla aynı hizada ve daha içerideki girintilerde tanınmakta ve değerini korumaktadır. Ancak sadece iç girintideki bir blokta değer atanan değişken dış bloklarda tanımsızdır. Bu tür değişkenlere yani kendi bloğu ve daha içerideki girintilerde tanınan değişkenlere yerel değişken denir. Bu özellik program yazarken dikkat edilmesi gereken bir durumdur.

Örnek 10

"if" bloğu içinde tanımlanan bir değişkene erişebilirsiniz.

```
yas=35
if yas ==35:
  deger='yolun yarısı'
print (deger)
yolun yarısı
```

Örnek 11

Şart sağlanmazsa "if" bloğunun içindeki kodlar çalışmayacaktır. Örnekte bunu deneyebilirsiniz.

```
yas=34
if yas ==35:
  deger2='yolun yarısı'
print (deger2)
NameError                 Traceback (most recent call last)
<ipython-input-126-3f455a188ff0> in <module>()
      2 if yas ==35:
      3 deger2='yolun yarısı'
----> 4 print (deger2)
```

Hata mesajında deger2 adında bir değişkenin tanımlı olmadığı ifade edilmektedir.

En dışta tanımlanan (girintisiz) değişkenler global değişkenlerdir. Global değişkenler ilk değerlerini tüm alt bloklara taşırlar.

Örnek 12

Alt bloklarda (iç girintide) aynı değişkene farklı bir değer atanabilir.

```
yas=34 # global bir değişken
dogumGunuMu=True
if dogumGunuMu==True:
  yas+=1 #yerelde aynı değişken 1 artırılmıştır.
  print ('Nice yıllara! Yaş:', yas)
Nice yıllara! Yaş: 35
```

Örnek 13

if koşulu sağlanamazsa bloğun içindeki kodlar atlanır. Global değişken olarak tanımlanan yas2 değerini korur.

```
yas2=34 # global bir değişken
dogumGunuMu=False
if dogumGunuMu==True:
  yas2+=1 #yerelde aynı değişken 1 artırılmıştır.
  print ('Nice yıllara! Yaş:', yas2)
print ('Yaşınız: ', yas2)
Yaşınız:  34
```

Örnek 14

Aynı şekilde fonksiyonlar içinde tanımlanmış değişkenler yerel (lokal) değişkenler olduğu için fonksiyon çağrılmazsa hata ile karşılaşılır.

```
def toplamBul (sayiListesi):
  toplam=0
  for i in range (len(sayiListesi)):
    topla+=sayiListesi[i]
  return topla
print (topla)
NameError                 Traceback (most recent call last)
<ipython-input-132-e72b6ce4bac6> in <module>()
----> 1 print (topla)
NameError: name 'topla' is not defined
```

Örnek 15

Programınızda global bir değişken yazarak işlemi tekrarlayabilirsiniz.

```
topla=0
def toplamBul (sayiListesi):
  topla=0
  for i in range (len(sayiListesi)):
    topla+=sayiListesi[i]
  return topla
```

Fonksiyonu tanımlanmasına rağmen global değişkene atanan değer ekrana yazdırılabilir.

```
print (topla)
0
```

Şimdi de fonksiyonunuzu çağırabilirsiniz.

```
toplamBul([1, 2, 3, 4, 5])
15
```

Değişkene atanan yeni değer yani fonksiyonun sonucu görülmektedir.

7.6. Bölüm Sonu Örnekleri

1. Vize, final puanlarını ve yüzdelik oranlarını parametre olarak alan ve puanlar ile yüzdelik oranları çarparak toplayan ve sonucu döndüren bir fonksiyon tanımlayınız ve (50, 60, 30, 70) argümanlarıyla fonksiyonunuzu kullanınız. kullanın. Sonuç sizce kaç olur?

2. Bir liste halinde verilen sayılardan tek olanların toplamını döndüren bir fonksiyon yazınız. [1, 2, 3, 4, 5, 6, 7, 9, 11, 1773, 1679] sayı listesiyle deneyiniz. Sonuç sizce kaç olur?

3. Aşağıdaki kodun ekran çıktısı sizce nasıl olur?

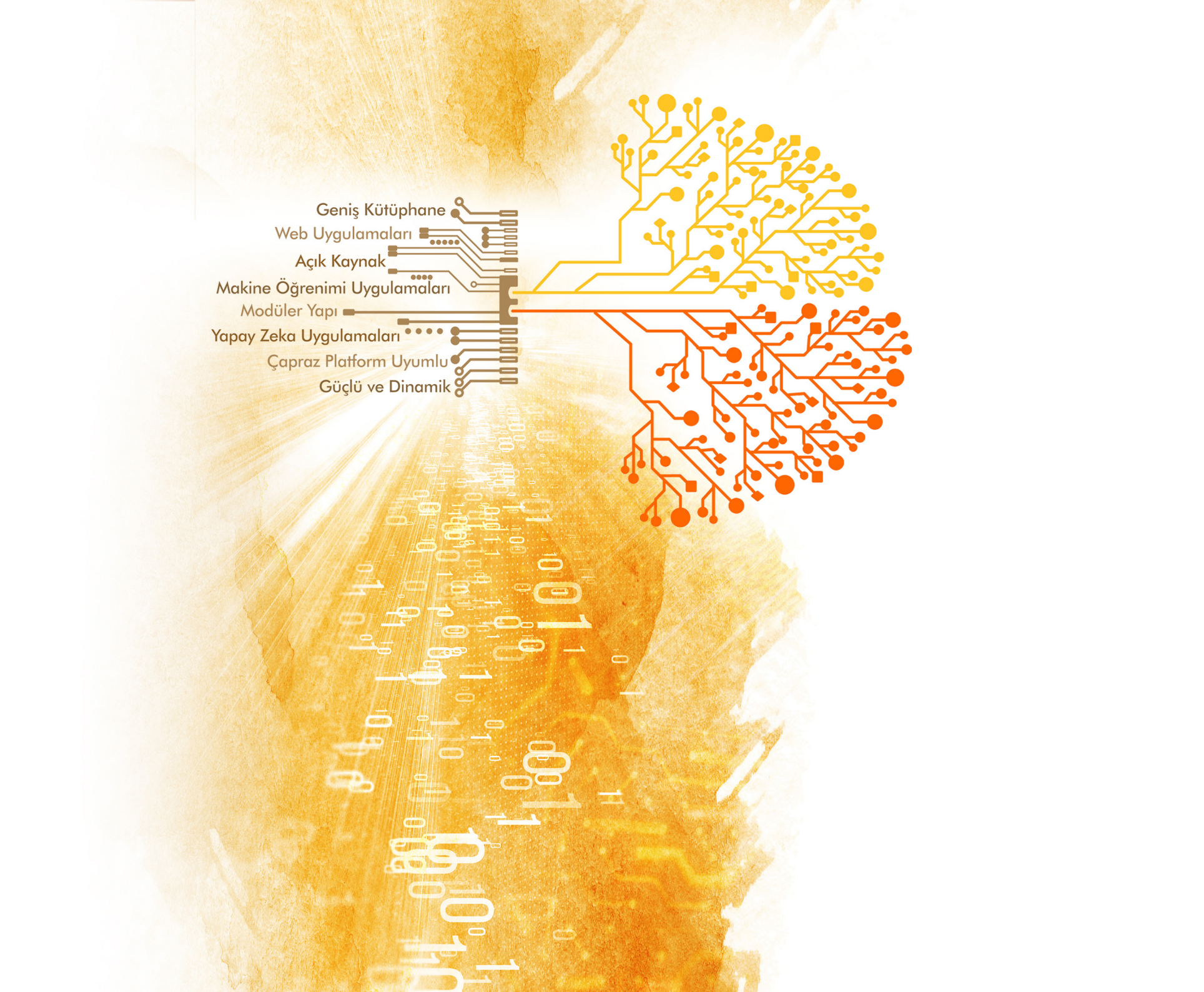
```
karsilamaMesaji='Merhaba'
kullaniciYasi=35
dogumGunuMu=True
if kullaniciYasi==35 and dogumGunuMu==True:
  karsilamaMesaji1='Yolun yarısı'
  kullaniciYasi+=1
  print(dogumGunuMu)
dogumGunuMu=False
print (kullaniciYasi)
print(dogumGunuMu)
print(karsilamaMesaji)
```

Cevaplar

1.
```
def puanHesaplama (vizePuani, finalPuani, vizeYuzdesi, finalYuzdesi):
  return(vizePuani*vizeYuzdesi/100+finalPuani*finalYuzdesi/100 )
puanHesaplama(50, 60, 30, 70)
57.0
```

2.
```
def tekSayilariTopla(sayilar):
  sayilarinToplami=0
  for i in range(len(sayilar)):
    if sayilar[i]%2!=0:
      sayilarinToplami+=sayilar[i]
  return sayilarinToplami
tekSayilariTopla([1, 2, 3, 4, 5, 6, 7, 9, 11, 1773, 1679])
3488
```

3.
```
True
36
False
Merhaba
```


Geniş Kütüphane
Web Uygulamaları
Açık Kaynak
Makine Öğrenimi Uygulamaları
Modüler Yapı
Yapay Zeka Uygulamaları
Çapraz Platform Uyumlu
Güçlü ve Dinamik

MODÜL 8

TURTLE (KAPLUMBAĞA) MODÜLÜ İLE GRAFİK ARAYÜZE GİRİŞ

Şekil 8.1: *Bölümle ilgili örnek uygulamalara karekod' dan ulaşabilirsiniz*

Bu bölüm, bilgisayar ekranlarından aşina olduğunuz pencereler (formlar) ile tanışacağınız ve kullanmaya başlayacağınız kısımdır. Grafik arayüz (pencere) kavramı, temelde yazılımın komut satırından çıkıp görsel ara birimler aracılığıyla ifade edilmesidir. Kaplumbağa grafikler grafik arayüz üzerinde çizim araçları kullanarak çalışılmasına olanak tanırlar. Kaplumbağa grafikler Python gibi metin tabanlı dillerde kod yazmaya yeni başlayanlar için en eğlenceli modüllerden biridir. Kaplumbağa grafikler ile daha önceki bölümlerde öğrendiğiniz kavramları kullanarak bol miktarda pratik yapabilirsiniz. Kaplumbağa grafiklerin tarihine bakıldığında matematik eğitimi için logo programla dilinde kullanıldığı görülmektedir. John Dewey'in öğrencisi olan Seymour Papert, öğrencilerine matematik öğretmek için kaplumbağa robotunu geliştirmiştir.

Turtle **(kaplumbağa)** standart python modüllerinden biridir. Bu nedenle turtle modülünü çalışmamızda kullanabilmek için modülü içeri aktarmamız (dâhil etmemiz) gerekmektedir. Bunun için ilk satıra **import turtle** yazmanız yeterli olacaktır. Python turtle modülünü çağırırken **turtle.py** adında bir dosya çağırdığından yapılacak örneklere turtle.py isminin verilmemesi yararlı olacaktır. turtle modülü projeye dâhil edildikten sonra bir turtle nesnesi oluşturup buna isim verilir. Bunu çizim yapmak için bir kalem almak gibi düşünebilirsiniz. **Bu bölümdeki uygulamalar idle kullanılarak yapılmıştır.**

8.1. Turtle (kaplumbağa) ile Çalışma

kalem=turtle.Turtle() yazarak kalem isminde bir Turtle (kaplumbağa) nesnesi oluşturmuş oluruz. Kalem artık Turtle nesnesinin sahip olduğu tüm özelliklere sahip olacaktır. Böylece Turtle nesnesinin sahip olduğu fonksiyonlar kullanılarak çizim işlemleri yapılabilir. Nesnenin fonksiyonlarını yazarken nesne adından sonra nokta koyup fonksiyon adı yazılır. Örneğin, **kalem. forward()** kalem nesnesinin ileri gitme fonksiyonudur. forward fonksiyonu gidilecek mesafeyi girdi olarak alır. Örneğin **kalem.forward (50)** yazılarak elli birim ilerlenir. Çizim işlemi bitince **turtle.done()** fonksiyonu ile program tamamlanır. Böylece turtle modülü kullanarak kod yazmaya hazır hale gelinir. Örnek 1'deki dört satırlık kodu yazıp çalıştırdığınızda sonuç Şekil 8.2'de görüldüğü gibi olur.

Örnek 1

100 birimlik ileri yönde çizgi çizme

```
import turtle
kalem=turtle.Turtle()
kalem.forward(100)
turtle.done()
```

Şekil 8.2: Örnek 1 kod çıktısı

8.2. Temel Hareket İşlemleri

Turtle modülünde dört temel hareket bulunmaktadır. Bunlar,

ileri: **forward (mesafe)**, geri: **backward (mesafe)**,

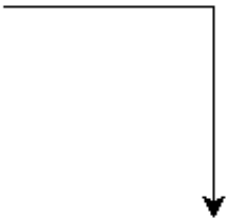
sağ: **right (açı)**, sol: **left (açı)** komutlarıdır.

Bu komutlar kullanılarak yapılan örnekleri inceleyelim.

Örnek 2

right ve forward fonksiyonlarının birlikte kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.forward(100)
kalem.right(90)
kalem.forward(100)
turtle.done()
```

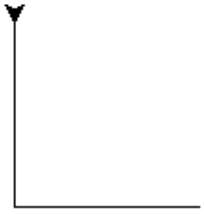


Şekil 8.3: Örnek 2 kod çıktısı

Örnek 3

right ve backward fonksiyonlarının birlikte kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.backward(100)
kalem.right(90)
kalem.backward(100)
turtle.done()
```

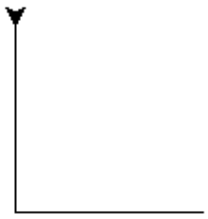


Şekil 8.4: Örnek 2 kod çıktısı

Örnek 4

right ve forward fonksiyonlarının birlikte kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.forward(-100.5)
kalem.right(90)
kalem.forward(-100.5)
turtle.done()
```

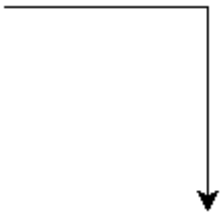


Şekil 8.5: *Örnek 4 kod çıktısı*

Örnek 5

right ve backward fonksiyonlarının birlikte kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.backward(-100.6)
kalem.right(90)
kalem.backward(-100.6)
turtle.done()
```



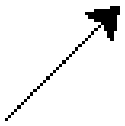
Şekil 8.6: *Örnek 5 kod çıktısı*

Şekil 8.3 ve Şekil 8.6 benzer çıktılar oluştuğunu fark ettiniz mi? Aynı durum Şekil 8.4 ve Şekil 8.5 için de geçerlidir. Bilgisayar: ileri 100 adım git ile geri eksi 100 adım git komutlarını aldığında aynı sonucu üretir. Kod çıktıları arasındaki fark mesafe girdileridir.

Örnek 6

left ve forward fonksiyonlarının birlikte kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.left(45)
kalem.forward(50)
turtle.done()
```

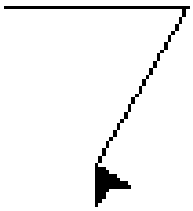


Şekil 8.7: Örnek 6 kod çıktısı

Örnek 7

Açı kullanarak şekil çizme örneği

```
import turtle
kalem=turtle.Turtle()
kalem.forward(50)
kalem.right(120)
kalem.forward(50)
kalem.right(120)
turtle.done()
```



Şekil 8.8: Örnek 7 kod çıktısı

Temel hareketler için bu örnekleri uyguladıktan sonra hareket fonksiyonlarının açı ve uzunlukları kullanıcıdan alacağınız örnekleri inceleyebilirsiniz.

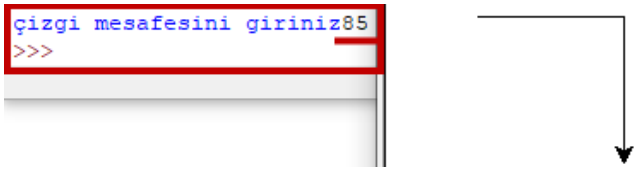
HATIRLATMA

Kullanıcıdan **input( )** fonksiyonu ile alınan metin değerini int veya float değerine dönüştürülmesi gerekir aksi durumda hata mesajı karşılaşılır.

Örnek 8

Çizgi mesafesini kullanıcıdan alıp şekil çizen program

```
import turtle
kalem=turtle.Turtle()
mesafe=float(input("çizgi mesafesini
giriniz"))
kalem.forward(mesafe)
kalem.right(90)
kalem.forward(mesafe)
turtle.done()
```

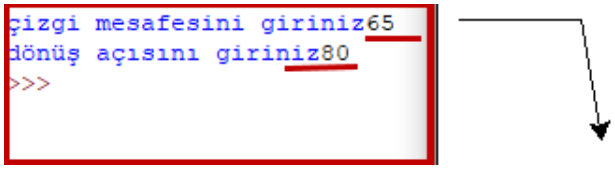


Şekil 8.9: Örnek 8 kod çıktısı

Örnek 9

Çizgi mesafesini ve dönüş açısını kullanıcıdan alıp şekil çizen program

```
import turtle
kalem=turtle.Turtle()
mesafe=int(input("çizgi mesafesini
giriniz"))
donus_açısı=int(input("dönüş açısını
giriniz"))
kalem.forward(mesafe)
kalem.right(donus_açısı)
kalem.forward(mesafe)
turtle.done()
```



Şekil 8.10: Örnek 9 kod çıktısı

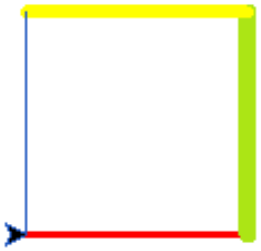
8.3. İşaretçi ve Çizim Araçları

Turtle nesnesinde çizim için kalem rengi: **pencolor(" ")** veya **color(" ")** girdi olarak iki tırnak içinde renk adı "red" veya tırnak içinde diyez ile birlikte html renk kodu "#ACE515" gibi alır. Renk kodlarını https://htmlcolorcodes.com/ adresinden öğrenebilirsiniz. Kalem kalınlığı **pensize (sayı değeri)** fonksiyonu ile ayarlanır.

Örnek 10

pencolor, pensize ve color uygulaması

```
import turtle
kalem=turtle.Turtle()
kalem.pencolor("red")
kalem.pensize(3)
kalem.forward(100)
kalem.left(90)
kalem.color("#ACE515")
kalem.pensize(8)
kalem.forward(100)
kalem.left(90)
kalem.pencolor("yellow")
kalem.pensize(6)
kalem.forward(100)
kalem.left(90)
kalem.color("#2259C1")
kalem.pensize(1)
kalem.forward(100)
kalem.left (90)
turtle.done()
```



Şekil 8.11: Örnek 10 kod çıktısı

Kaplumbağa grafiklerde pencerenin istenen noktasına gidebilme **goto (x ekseni, y ekseni)**

Kaplumbağa grafiklerde nokta çizme **dot( )**

Kaplumbağa grafiklerde çizmeden ilerleme **penup( )** veya **up( )** ardından **forward( )**

Kaplumbağa grafiklerde çizime tekrar dönmek için ilerleme **pendown( )** veya **down( )** ardından **forward( )** fonksiyonları kullanılır.

Turtle çizim aracının görünüm şekli için de **shape( )** fonksiyonu bulunmaktadır. Bu fonksiyon **'arrow'**, **'classic'**, **'turtle'**, **'circle'** olmak üzere 4 şekle sahip olabilir.

Örnek 11

dot ile çizim uygulaması

```
import turtle
kalem=turtle.Turtle()
for i in range(5):
    kalem.dot()
    kalem.forward(20)
turtle.done()
```

Şekil 8.12: Örnek 11 kod çıktısı

Örnek 12

dot ile çizgisiz şekil uygulaması

```
import turtle
kalem=turtle.Turtle()
kalem.up()
for i in range(5):
    kalem.dot()
    kalem.forward(20)
turtle.done()
```

Şekil 8.13: Örnek 12 kod çıktısı

Örnek 13

shape ve goto fonksiyonlarının kullanımı

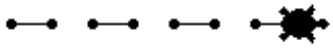
```
import turtle
kalem=turtle.Turtle()
kalem.shape("turtle")
kalem.forward(100)
kalem.penup()
kalem.goto(0,100)
for i in range(5):
    kalem.dot()
    kalem.forward(20)
turtle.done()
```

Şekil 8.14: Örnek 13 kod çıktısı

Örnek 14

shape ve goto fonksiyonlarının farklı bir çizim için kullanımı

```
import turtle
kalem=turtle.Turtle()
kalem.shape("turtle")
for i in range(4):
    kalem.up()
    kalem.forward(20)
    kalem.dot()
    kalem.down()
    kalem.forward(20)
    kalem.dot()
turtle.done()
```



Şekil 8.15: Örnek 14 kod çıktısı

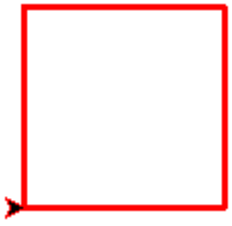
8.4. Turtle ile Geometrik Şekiller Çizme

Turtle nesnesi ile geometrik şekiller çizmek oldukça kolaydır. Çokgenler başta olmak üzere birçok geometrik şekli ileri ve sağ/sol dönüş fonksiyonları ile yapmak mümkündür. Örnek 15'te bir kare çizdirildiğini fark etmişsinizdir. Fark etmenizi istediğimiz bir diğer nokta, forward ve left fonksiyonlarının 4 defa kullanılmasıdır (Örnek 15). Kodu defalarca yazmak yerine döngüler konusunda işlendiği gibi for yapısı ile kolayca çizdirebiliriz. Aynı mantığı Örnek 16'da gösterildiği gibi altıgen çizdirmek için kullanabiliriz.

Örnek 15

turtle ile kare çizme uygulaması

```
import turtle
kalem=turtle.Turtle()
kalem.pencolor("red")
kalem.pensize(3)
for i in range(4):
    kalem.forward(100)
    kalem.left(90)
turtle.done()
```

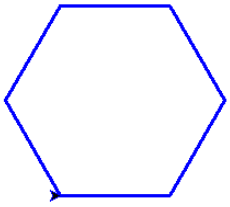


Şekil 8.16: *Örnek 15 kod çıktısı*

Örnek 16

turtle ile altıgen çizme uygulaması

```
import turtle
kalem=turtle.Turtle()
kalem.pencolor("blue")
kalem.pensize(3)
for i in range(6):
    kalem.forward(100)
    kalem.left(60)
turtle.done()
```



Şekil 8.17: *Örnek 16 kod çıktısı*

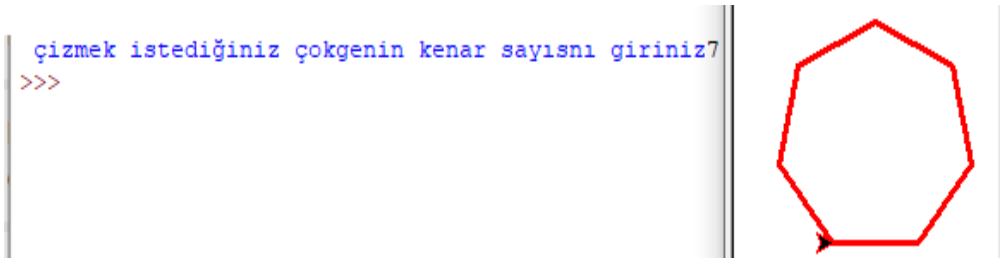
Çokgen çizdirmeyi kenar sayısını kullanıcıdan isteyerek de yapabilirsiniz. Burada dikkat edilecek nokta çokgen kaç kenarlı olursa olsun çokgen çizimi tamamlandığında 360 derecelik dönüş yapacak olmasıdır. Bu nedenle çokgenin dönüş açısı **360/ kenar sayısı** olur.

Örnek 17

Kullanıcının istediği kenar sayısında çokgen çizdirmek

```
import turtle
kalem=turtle.Turtle()
kalem.pencolor("red")
kalem.pensize(3)

kenar_sayısı=int(input(" çizmek istediğiniz çokgenin kenar sayısını giriniz"))
for i in range(kenar_sayısı):
    kalem.forward(50)
    kalem.left(360/kenar_sayısı)
turtle.done()
```



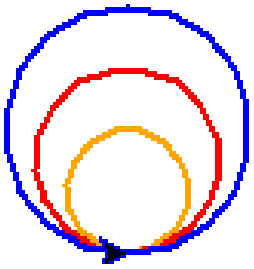
Şekil 8.18: Örnek 17 kod çıktısı

Çember çizmek için **circle (yarıçap değeri)** fonksiyonu yeterlidir. Örnek 18'de farklı renkte ve yarıçapta 3 adet çember çizdirilmektedir.

Örnek 18

circle fonksiyonu ile 3 farklı yarıçapta ve renkte çember çizdirme

```
import turtle
kalem=turtle.Turtle()
kalem.pencolor("orange")
kalem.pensize(2)
kalem.circle(20)
kalem.pencolor("red")
kalem.circle(30)
kalem.pencolor("blue")
kalem.circle(40)
turtle.done()
```



Şekil 8.19: *Örnek 18 kod çıktısı*

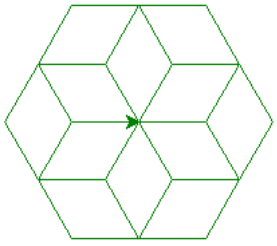
Çember ve çokgenler kullanılarak çok farklı desenler oluşturabileceğini fark ettiniz mi?

Çokgenlerle farklı desenler oluşturmaya dair Örnek 19'u inceleyebilirsiniz. Bunu yapmak için iç içe döngüler kullanılmaktadır. Yani çokgenlerden bol miktarda çizdirilmesi gerekmektedir. Bu kodlar robota yaptırıldığında istenilen desenler farklı yüzeylere çizdirilebilirdi.

Örnek 19

İç içe döngüler ile desen çizdirme

```
import turtle
kalem=turtle.Turtle()
kalem.color("green")
for i in range (6):
    for j in range (6):
        kalem.forward(50)
        kalem.left(60) # iç döngü
    kalem.left(60)# dış döngü
turtle.done()
```



Şekil 8.20: *Örnek 19 kod çıktısı*

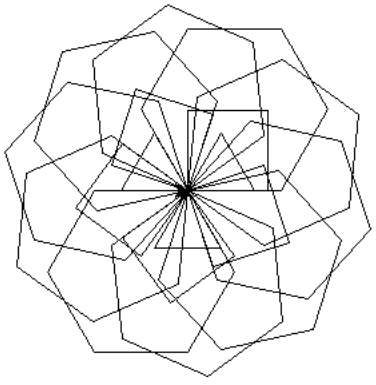
Örnek 19'daki deseni istendiğinde ve parametreler girildiğinde çizen bir fonksiyon tanımlayıp kullanınız.

Örnekte fonksiyon için verilen değerlerin doğruluğunun kontrol edildiğine dikkat ediniz. Kenar uzunluğunun eksi olarak da girilebileceği düşünüldüğünden ve if koşul yapısının karmaşık hâle getirilmemesi için dâhil edilmemiştir.

Örnek 20

Desen çizen fonksiyon uygulaması

```
import turtle
def desen_çiz (kenar_uzunluğu=50,iç_kenar=3,tur_sayısı=3):
    if(tur_sayısı <01 or iç_kenar<3):
        print("hatalı veri girdiniz")
    else:
        kalem=turtle.Turtle()
        for i in range (tur_sayısı):
            for j in range (iç_kenar):
                kalem.forward(kenar_uzunluğu)
                kalem.left(360/iç_kenar)
            kalem.left(360/tur_sayısı)
desen_çiz()
desen_çiz(60,4,5)
desen_çiz(70,6,10)
turtle.done()
```

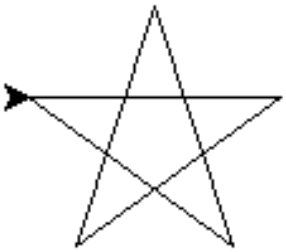


Şekil 8.21: Örnek 20 kod çıktısı

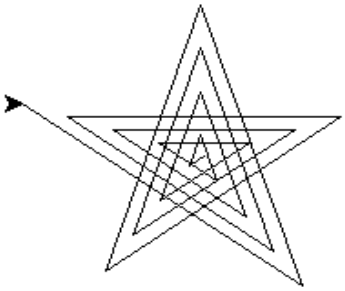
8.5. Bölüm Sonu Örnekleri

Resimlerde gösterilen şekilleri çizecek Python kodlarını yazınız.

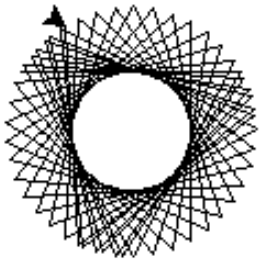
Soru-1



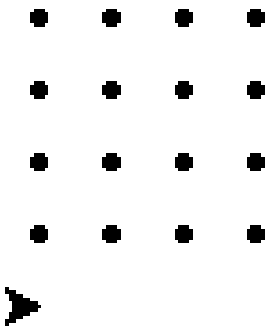
Soru-2



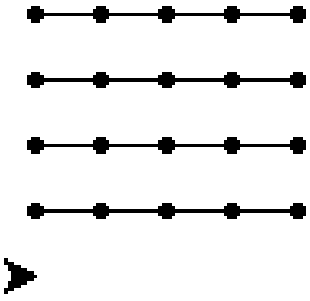
Soru-3



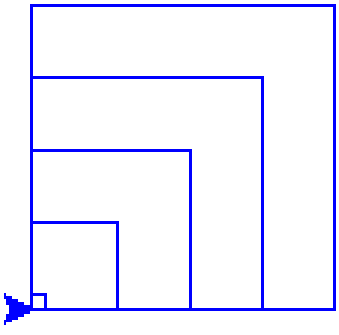
Soru-4



Soru-5



Soru-6



Cevaplar

1.

```
import turtle
yıldız = turtle.Turtle()
for i in range(5):
    yıldız.forward(100)
    yıldız.right(144)
turtle.done()
```

2.

```
import turtle
yıldız = turtle.Turtle()
for i in range(20):
    yıldız.forward(i * 10)
    yıldız.right(144)
turtle.done()
```

3.

```
import turtle
yıldız = turtle.Turtle()
for i in range(50):
    yıldız.forward(100)
    yıldız.right(123)
turtle.done()
```

4.

```
import turtle
kalem = turtle.Turtle()
kalem.penup()
for y in range(4):
    for i in range(4):
        kalem.dot()
        kalem.forward(20)
    kalem.backward(80)
    kalem.right(90)
    kalem.forward(20)
    kalem.left(90)
turtle.done()
```

5.

```
import turtle
kalem = turtle.Turtle()
for y in range(4):
    kalem.down()
    kalem.dot()
    for i in range(4):
        kalem.forward(20)
        kalem.dot()
    kalem.up()
    kalem.backward(80)
    kalem.right(90)
    kalem.forward(20)
    kalem.left(90)
turtle.done()
```

6.

```
import turtle
kalem = turtle.Turtle()
kalem.color("blue")
for k in range(5,106,25):
    print(k)
    for i in range(4):
        kalem.forward(k)
        kalem.left(90)
turtle.done()
```

MODÜL 9

SÖZLÜKLER VE DEMETLER

Şekil 9.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

Sözlükler, gerçek hayattaki sözlükler gibi davranan bir veri tipi olarak bilinmektedir. Python ile JSON formatında verilerde İsim Değer Çifti Koleksiyonu sözlük yapısıyla birebir uyumludur. Bundan dolayı sözlük yapısı JSON verilerinde kullanılmaktadır. Sözlüğün içindeki her bir eleman indeks olarak değil, anahtar (key) ve değer (value) olarak tutulur. Anahtar değer çiftleri virgülle ayrılarak bir öğeyi oluştururlar. Değer kısmı bütün veri türünü içerebilir fakat anahtar kısmı sadece string ve int tipinde olabilir. Sözlükler sırasız bir öge koleksiyonudur. Anahtar ve değer yapısında veri kümeleri oluştururlar. Değerler herhangi bir veri türüne ait olabilir ve tekrarlanabilir fakat anahtarlar kesinlikle benzersiz olmalıdır.

9.1. Sözlük Oluşturma

Sözlük oluştururken dikkat edilmesi gerekenler:

- İki noktayı ve virgülü nereye koyduğumuza,
- Ögelerimizi tanımlarken ayraç parantez "()" yerine Küme Parantezi "{}" kullandığınıza,
- Anahtar-değer ilişkisine,
- Anahtar değeri tanımlarken tırnak işareti "" kullandığınıza dikkat ediniz.

Aşağıdaki örnekte, süslü parantez ve iki nokta {:} ile anahtar değerlerinizi yerleştirerek kişi bilgilerini içeren bir sözlük oluşturunuz.

```
kisisel_bilgiler={"ad":"ali","telefon":"05554442211","e_posta":"ali@abc.
com","ülke":"Türkiye","il":"istanbul","ilçe":"beşiktaş"}
print(kisisel_bilgiler)
{'ad': 'ali', 'telefon': '05554442211', 'e_posta': 'ali@abc.com', 'ülke': 'Türkiye',
'il': 'istanbul', 'ilçe': 'beşiktaş'}
```

Şimdi de key tiplerine göre sözlükler oluşturup ekrana yazdırınız.

dict( ) fonksiyonu kullanarak sözlük oluşturabilirsiniz. Örnek 1'de görüldüğü gibi sozluk4 adında 2 adet ögesi olan sözlük oluşturulmuştur.

Örnek 1

```
# bos bir sozluk
sozluk1 = {}
print(sozluk1)
# integer veri tipi oluşturulan keyler
sozluk2= {1: 'adana', 2: 'adıyaman'}
print(sozluk2)
# string ve integer veri tipi ile oluşan keyler
sozluk3 = {'isim': 'ali', 1: [5, 4, 3]}
print(sozluk3)
# dict fonksiyonu
sozluk4 = dict({1:'erik', 2:'ayva'})
print(sozluk4)
{}
{1: 'adana', 2: 'adıyaman'}
{'isim': 'ali', 1: [5, 4, 3]}
{1: 'erik', 2: 'ayva'}
```

Örnek 2'de 3 adet anahtara sahip sözlük oluşturularak listelenmiştir.

Örnek 2

```
sozluk = {"bilgisayar" : "computer","sarı" : "yellow","masa" : "chair"}
print(sozluk)
{'bilgisayar': 'computer', 'sarı': 'yellow', 'masa': 'chair'}
```

9.2. Sözlük Anahtar ve Değerlerine Erişim

Sözlük anahtar ve değerlerini listelemek için keys ve values komutları kullanılır.

Örnek 3

#keys( ) metodu sözlükteki anahtarları yazdırır.

```
iller={ "konya" :"42","istanbul"   :"34","ankara" :"06" }
print ( iller.keys())
#values()metodu  sözlükteki değerleri bize yazdırır.
print (iller.values())
dict_keys(['konya', 'istanbul', 'ankara'])
dict_values(['42', '34', '06'])
```

Örnek 3'te iller sözlüğünde önce anahtarlar sonra değerleri listelenmiştir. Anahtarlar 'konya', 'istanbul', 'ankara' ve değerleri ise '42', '34', '06' şeklinde vermiştir.

9.3. Sözlüklerde Eleman Seçme, Silme, Ekleme, Değiştirme

9.3.1. Sözlükte Eleman Seçme İşlemleri

Örnek 4'te meyveler adında sözlük oluşturulmuştur. Bu sözlükte adı, türü ve kg adında 3 adet anahtar ve bu anahtarlara ait değerleri mevcuttur. Araçlar sözlüğünde adı ve kg anahtarları ekrana yazdırılarak eleman seçme işlemi yapılmıştır.

Örnek 4

```
meyveler={"Adı": "Portakal","Türü": "Turunçgiller","Kg": 20}
print(meyveler["Adı"])
print(meyveler["Kg"])
Portakal
20
```

Örnek 5

```
kisi_bilgileri = {'ad':'ali','yas':40,'memleketi':'konya'}
print(kisi_bilgileri)
# deger değiştirme
kisi_bilgileri['yas'] =45
    print(kisi_bilgileri)
    # deger ekleme
   kisi_bilgileri['adres'] = 'nişantaşı'
   print(kisi_bilgileri)
{'ad': 'ali', 'yas': 40, 'memleketi': 'konya'}    1.çıktı
{'ad': 'ali', 'yas': 45, 'memleketi': 'konya'}    2.çıktı
{'ad': 'ali', 'yas': 45, 'memleketi': 'konya', 'adres': 'nişantaşı'}   3.çıktı
```

Örnek 5'te kişi bilgileri adında sözlük oluşturulmuştur. Bu sözlükte ad, yas, memleketi adında 3 anahtar ve değerleri tanımlanmıştır. Bu değerler 1.çıktıda ekrana yazdırılmıştır. Daha sonra yas değeri 45 olarak değiştirilmiştir. 2. çıktıda tüm değerler ekrana yazdırılarak yas değerinin 45 olarak değiştirildiği görülmüştür. 3. çıktıda ise adres adında anahtar ve değeri eklenerek listelenmiştir.

9.3.2. Sözlükte Eleman Silme İşlemleri

pop( ) metodu, öğeyi belirtilen anahtar adıyla kaldırmak için kullanılır.

Örnek 6

```
kisi_bilgileri= {"Adı": "Ekrem","Soyadı": "Yıldırır","Yaşı": 40 } print(kisi_
bilgileri)
kisi_bilgileri.pop("Yaşı")
print(kisi_bilgileri)
{'Adı': 'Ekrem', 'Soyadı': 'Yıldırır', 'Yaşı': 40}
{'Adı': 'Ekrem', 'Soyadı': 'Yıldırır'}
```

Örnek 6'yı incelediğimizde Yaşı anahtarı pop metodu ile kaldırılmıştır. Kisi_bilgileri sözlüğü listelenerek sadece Adı ve Soyadı anahtarlarının olduğu görülmüştür.

popitem( ) metodu, eklenen son öğeyi kaldırmak için kullanılır. Örnek 7'de popitem metodu kullanılmıştır. Bu metot kullanılarak, kisi_bilgileri adlı sözlükte son anahtar olan Yaşı kaldırılmıştır.

Örnek 7

```
kisi_bilgileri= {
"Adı": "Ekrem",
"Soyadı": "Yıldırır",
"Yaşı": 40
}
print(kisi_bilgileri)
kisi_bilgileri.popitem()
print(kisi_bilgileri)
{'Adı': 'Ekrem', 'Soyadı': 'Yıldırır', 'Yaşı': 40}
{'Adı': 'Ekrem', 'Soyadı': 'Yıldırır'}
```

del (parametre) anahtar sözcüğü, belirtilen anahtar adına sahip öğeyi kaldırmak için kullanılmaktadır. Örnek 8'de del komutu ile Yaşı adlı anahtar değere ait 40 değeri silinmiştir.

Örnek 8

```
kisi_bilgileri= {
"Adı": "Ekrem",
"Soyadı": "Yıldırır",
"Yaşı": 40
}
del kisi_bilgileri["Yaşı"]
print(kisi_bilgileri)

{'Adı': 'Ekrem', 'Soyadı': 'Yıldırır'}
```

clear( ) anahtar sözcüğü sözlüğü boşaltırken, **del** anahtar sözcüğü ise ayrıca sözlüğü tamamen silmektedir. Aşağıdaki örnekte clear( ) anahtar sözcüğü kullanılmış ve sözlüğün içi boşaltılarak ekrana yazdırılmıştır.

```
sozluk ={"black":"siyah",   "green":"yeşil",   "white":"beyaz",}
print(sozluk)
sozluk.clear()
print(sozluk)
{'black': 'siyah', 'green': 'yeşil', 'white': 'beyaz'}
{}
```

Örnek 9

```
kisi_bilgileri= {"Adı": "Ekrem","Soyadı": "Yıldırır","Yaşı": 40}
kisi_bilgileri.clear()
print(kisi_bilgileri)
del kisi_bilgileri
print(kisi_bilgileri)

{}
NameError Traceback (most recent call last)
 HYPERLINK "https://localhost:8080/" <ipython-input-9-c04608b44e6f> in <module>()
      3 print(kisi_bilgileri)
      4 del kisi_bilgileri
----> 5 print(kisi_bilgileri)

NameError: name 'kisi_bilgileri' is not defined
```

Örnek 9'da kisi_bilgileri sözlüğünde clear( ) metodu ile içi boşaltılmış. Daha sonra del( ) metodu ile kisi_bilgileri sözlüğü silinmiştir. Ekrana listele işlemi yapıldığında kisi_bilgileri adlı sözlük olmadığı için hata mesajı ile karşılaşılmıştır.

9.3.3. Sözlüğe Eleman Ekleme İşlemleri

Sözlüklere veri ekleme işlemi yapılabilmektedir. Yöntem sözlükadı["anahtar"]="deger" şeklinde olmaktadır. Örnek 10'da sözlükte önce tek öğesi bulunmaktadır. İkinci ve üçüncü öge eklenerek listeleme işlemi yapılmıştır.

Örnek 10

```
sozluk={"adi":"sami", }
print(sozluk)
sozluk['soyadi']='yılmaz'
print(sozluk)
sozluk['yasi']=40
print (sozluk)
{'adi': 'sami'}
{'adi': 'sami', 'soyadi': 'yılmaz'}
{'adi': 'sami', 'soyadi': 'yılmaz', 'yasi': 40}
```

9.3.4. Sözlükte Eleman Değiştirme İşlemleri

Sözlükte eleman değiştirmek için anahtarı kullanarak güncelleme işlemi yapılmıştır.

Kullanımı:

sözlük adı['anahtar adı']=yeni değer şeklinde yapılmıştır.

Örnek 11

```
sozluk = {'isim':'ahmet','yas':40}
# deger değiştirme
sozluk['yas'] = 45
print(sozluk['yas'])
45
```

Örnek 11'de olduğu gibi sözlüğün yas adlı anahtarının değeri 40 iken 45 olarak değiştirilmiştir.

9.4. Sözlükler Üzerinde Gezinme

Python'da sözlükler öğelerine erişmek için anahtarlarını (KEY) kullanırlar ve değerler bu indexler üzerinden gösterilir. Anahtarlar köşeli parantez içerisinde ya da get( ) fonksiyonu ile kullanılır. get( ) fonksiyonunda anahtar bulunamazsa KeyError yerine "None" döndürür.

Örnek 12

```
sozluk = {'ad': 'ali', 1: [5, 4]}
print(sozluk['ad'])
print(sozluk.get(1))
ali
[5, 4]
```

Örnek 12'de sozluk adlı sözlükte ad ve 1 adında anahtarlar ve bu anahtarlara bağlı ali ve 5,4 değerleri bulunmaktadır. Ad ve 1 anahtarlarını kullanılarak değerleri ekranda listelenmiştir.

Örnek 13'te sozluk1 adında sözlük oluşturulmuş ve bir ve iki anahtarları kullanarak değerlerine erişilmiştir.

Örnek 13

```
sozluk1 = {"sıfır":0,"bir":1,"iki":2,"üç":3}
print(sozluk1)
# "bir" anahtarına karşılık gelen değeri buluyoruz.
print(sozluk1 ["bir"])
# "iki" anahtarına karşılık gelen değeri buluyoruz.
print(sozluk1["iki"])
{'sıfır': 0, 'bir': 1, 'iki': 2, 'üç': 3}
1
2
```

Örnek 14

```
deneme = {"sıfır":0,"bir":1,"iki":2,"üç":3}
print(deneme["on"])
Traceback (most recent call last):
File "<ipython-input-22-e25f3b4387f3>", line 2, in <module>
print(deneme["on"])
KeyError: 'on'
```

Örnek 14'te sözlükte olmayan bir anahtar girildiğinde ekrana KeyError:'on' diye key hatası mesajı verecektir.

9.5. Demet (tuple) Oluşturma ve Eleman İşlemleri

Demetler veri ekleme ve çıkarmanın yapılamadığı veri yapılarıdır. Bu değiştirilemez özellikleri dışında listelere benzemektedirler. Demetler ekleme çıkarma gibi işlemlerle uğraşmadığı için daha hızlı çalışır. Eklenen verilerin program boyunca değiştirilmesinin istenmediği durumlarda kullanılmaktadır.

Boş bir demet oluşturma:

Örnek 15

```
demet=()
print(demet)
()
```

Örnek 15'te boş bir demet oluşturularak, ekrana listelenmiştir.

İndis içeren demet oluşturma:

Örnek 16

```
demet = ("Python","Java",2020,"JavaScript")
print(demet)
('Python', 'Java', 2020, 'JavaScript')
```

Örnek 16'da 4 ögesi bulunan bir demet oluşturularak ekrana yazdırılmıştır.

Tek elemanlı demet oluşturma:

Tek nesneli bir demet oluşturmak için nesneden sonra virgül konulması gerekmektedir. Örnek 17'de buna yönelik bir uygulamadır.

Örnek 17

```
meyveler =("erik",)
print(meyveler)
('erik',)
```

NOT

Eğer virgül koymazsanız bu değişken tipi demet değil string olarak sistemde yer alacaktır.

Demetler de sözlükler gibi indis olarak sıfırdan başlar. Örnek 18'de 6 elemanlı bir demet oluşturulmuştur. 1. İndise ulaşmak için demetin indis numarası kapalı parantez arasında girilerek ekrana yazdırılmıştır. print(demet[-1]) ise demet'in en son elemanı bize vermektedir. print(demet[2:])'da 2 numaralı indis dâhil iki numaradan sonraki tüm indislere ait elemanları listelemek için kullanılır.

Örnek 18

```
demet = (5,10,15,20,25,30)
# 1. indise ulaşma
print(demet[0])
# 3. indise ulaşma
print(demet[2])
print(demet[-1])
print(demet[2:])
5
15
30
(15, 20, 25, 30)
```

Aynı zamanda bir demet tuple( ) fonksiyonu ile de oluşturulabilir ve constructor yöntemiyle başlangıçta elemanları ayarlanabilir. Örnek 19'da tuple fonksiyonu ile demet oluşturulmuş ve öğeleri ekrana yazdırılmıştır.

Örnek 19

```
demet = tuple(("Ankara","İstanbul","Kayseri"))
print(demet)
('Ankara', 'İstanbul', 'Kayseri')
```

Örnek 20'de sayılar ve harfler diye iki adet demet tanımlanarak elemanları girilmiştir. Sayılar ve harfler demetleri toplanarak yeni_demet adlı demete aktarılmıştır. Elemanları ekrana yazdırıldı. Bu şekilde iki demet birleştirilmiştir.

Örnek 20

```
sayılar = (0,1,2,3,4,5,6,7,8,9,) #bir demet tanımladım
harfler = ("a","b","c","d","e") #ikinci bir demet tanımladım
yeni_demet = sayılar + harfler #tanımladığım demetleri topladım
print(yeni_demet) #yeni demeti ekrana yazdırdım
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c', 'd', 'e')
```

Örnek 21'de olduğu gibi if koşul yapısı ile demetlerin içerisinde aranılan elemanların olup olmadığını kontrol edilmiştir. Gelen mantıksal cevabı ise ekrana true ya da false olarak yazdırılmıştır. "a" harfi harfler adlı demette olduğu için ekrana true olarak mesaj yazdırılmıştır.

Örnek 21

```
harfler = ("a","b","c","d","e")
if "a" in harfler: #True çıktısı verir
print(True)
else:
print(False)
True
```

Demet veri yapılarında elemanlar iki parantez arasına yazılır lakin bu parantez zorunlu değildir. Yine aynı şekilde liste veri yapılarında olduğu gibi demetlerde de dilimleme **(sliceable)** ve iç içe geçirilme **(nestable)** yapılabilir.

Örnek 22'de iç içe demetlere ait uygulama olup listeleme işlemi yapılmıştır. d adında demet oluşturulmuş ve iç içe demet yapısı olarak 2. indis numarasındaki ait 0. indis elemanına ve 3. indis numarasındaki 1. indis numarasına ait elemana ulaşılmıştır.

Örnek 22

```
d=((1,2),(3,4),(5,6),(7,8))
print(d[2][0])
print(d[3][1])
5
8
```

9.6. Demetlerin Temel Metotları

index metodu demette yer alan ögenin sırasını bulmaya yardımcı olur. Örnek 23'te demet adında demet oluşturulmuştur. 4 ögeden oluşan demette index metodu kullanılarak zambak adlı ögenin 2. sırada olduğunu görülmüştür.

Örnek 23

```
# Demet oluşturalım.
demet = ("lale","kardelen","zambak","papatya")
# "zambak" elemanının indeksini buluyoruz.
print(demet.index("zambak"))
2
```

Count metodu tuple içinde aynı elemandan kaç adet olduğunu bulunmasına yardımcı olur. Java adlı ögenin Örnek 24'te 3 adet olduğu görülmektedir.

Örnek 24

```
demet = ("Python","Java","C#","Delphi","C++","Java","Java")
print(demet.count("Java"))
3
```

Liste tipindeki veriyi tuple'a dönüştürmeyi değişkene aktarıp yapılabilir. Örnek 25'te aylar adlı liste demet adlı değişkene aktarılarak, demet dönüşümü yapılarak, ekrana yazdırılmıştır.

Örnek 25

```
aylar =["ocak","şubat","mart","nisan","mayıs","haziran"]
demet = tuple(aylar)
print(demet)
('ocak', 'şubat', 'mart', 'nisan', 'mayıs', 'haziran')
```

9.7. Demetler Üzerinde Gezinme

Demetler üzerinde gezinme işlemleri için for döngüsü kullanılır. Bu şekilde tüm öğeleri aynı anda listelenebilir. Örnek 26'da deneme adlı demet tanımlanarak içine 7 adet öge girilmiştir. Bu ögeler ise for döngüsü ile ekrana ögeleri alt alta yazdırılmıştır.

Örnek 26

```
deneme= (1,2,3,4,5,6,7)
for eleman in deneme:
print(eleman)
1
2
3
4
5
6
7
```

Örnek 27'de demette sorted( ) metodu ile ögeleri harflere göre sıralama yapılmıştır.

Örnek 27

```
demet =("lale","kardelen","zambak","papatya")
print(sorted(demet))
['kardelen', 'lale', 'papatya', 'zambak']
```

9.8. Bölüm Sonu Örnekleri

1. 5 ve 6 elemanlı olacak şekilde iki adet demet tanımlayınız. Bu iki adet demeti önce birleştirip sonra eleman sayılarını bulan programı yapınız.

2. `Sayilar=(20,24,25,79,40,39,50)` demet olarak tanımlanmıştır. Bu demet'te 5'e bölünenleri ekrana yazdıran programı yapınız.

3. Aşağıdaki kod çalıştırıldığında ekran çıktısı nedir?

```
demet = ("hasan","ali","c","mehmet","deniz","f","fatma")
yenidemet = demet[3:5]
print(yenidemet)
```

4. `Uygulama=("ali","veli","ayşe","Fatma","Hayriye","ali","deniz")` şeklinde tanımlanmış bir demet'te ali adlı öğenin kaç adet olduğunu bulunuz.

5. Aşağıdaki kod çalıştırıldığında ekran çıktısı nedir?

```
sozluk = {'renk': 'mavi', 'kıyafet': 'pantolon', 'beden': 'M'}
for anahtar in sozluk:
print(anahtar)
```

6. Aşağıdaki kod çalıştırıldığında ekran çıktısı nedir?

```
sozluk = {'renk': 'mavi', 'kıyafet': 'pantolon', 'beden': 'M'}
for anahtar in sozluk:
print(anahtar,sozluk[anahtar])
sozluk_bilesenleri=sozluk.items()
for bilesen in sozluk_bilesenleri :
print(bilesen)
print(type(bilesen))# demet veri tipinde oluyor
...
```

Cevaplar

1.
```
aylar =("ocak","şubat","mart","nisan","mayıs","haziran")
mevsimler = ("kış","ilkbahar","yaz","sonbahar")
yeni_demet=tuple(mevsimler+aylar)
print(len(yeni_demet))
10
```

2.
```
sayilar=(20,24,25,79,40,39,50)
for meyve in sayilar:
if meyve%5 ==0:
print(meyve)
20
25
40
50
```

3.
```
demet = ("hasan","ali","c","mehmet","deniz","f","fatma")
yenidemet = demet[3:5]
print(yenidemet)
('mehmet', 'deniz')
```

4.
```
uygulama=("ali","veli","ayşe","Fatma","Hayriye","ali","deniz")
print(uygulama.count("ali"))
2
```

5.
```
sozluk = {'renk': 'mavi', 'kıyafet': 'pantolon', 'beden': 'M'}
for anahtar in sozluk:
print(anahtar)
```
```
renk
kıyafet
beden
```

6.
```
sozluk = {'renk': 'mavi', 'kıyafet': 'pantolon', 'beden': 'M'}
for anahtar in sozluk:
print(anahtar,sozluk[anahtar])
sozluk_bilesenleri=sozluk.items()
for bilesen in sozluk_bilesenleri :
print(bilesen)
```
```
renk mavi
kıyafet pantolon
beden M
('renk', 'mavi')
<class 'tuple'>
('kıyafet', 'pantolon')
<class 'tuple'>
('beden', 'M')
<class 'tuple'>
```

MODÜL 10

MODÜLER PROGRAMLAMA

Şekil 10.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

Modülerin bir diğer anlamı da birbiriyle uyumlu olan ve önceden hazırlanmış parçalardır. Bu parçalar bir araya gelerek uyumlu bir şekilde bir bütünü oluşturmaktadır. Python'da aslında her bir dosya modüldür. Modüler içinde fonksiyonları sınıfları ve objeleri bulundurur. Bu modülleri projemize dâhil edersek içerisinde bulunan fonksiyonları, sınıfları ve objeleri kullanabiliriz. Python modüler yapıyı destekleyen bir programlama dilidir. Python birçok modül içerdiği gibi, kullanıcı kendisi de modül yazıp kullanabilir.

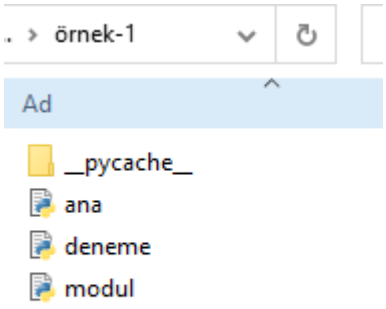
Programın önceden yazılmış parçaları, bütüne istenildiği zaman dâhil edilerek çalıştırabilir. Böylelikle programcının çalıştığı ortam sadece kendi işine yaradığı modülleri kullanacağından gereksiz yere bellek tüketimi olmayacaktır. Python birçok modül içerdiği gibi, kullanıcı kendisi de modül yazabilir. Python modülleri kütüphane olarak da adlandırılır.

Modül Kullanmanın Avantajları:

- Modüller, kod tekrarını önler.
- Modüller, projemizin daha okunabilir olmasını sağlar.
- Modüller yapıda projeyi değiştirmek, güncellemek, yeni modüller eklemek daha basittir.
- Modüller yapıda projede bakım kolaydır.
- Modüller yapıda aynı projede birden fazla kişinin çalışmasına imkân sağlar.

10.1 Modül Yazma ve Çağırma

Modül oluşturmak ve projeye dâhil etmek için bir Python dosyası oluşturulur. Çünkü Python dosyası da bir modüldür. "modul.py" adında bir Python dosyası oluşturup kaydedilir. Örnek 1, örnek 2 …. adında devam edecektir. Bunların her biri ayrı klasörlerdir. Bu klasörlerin içine Python dosyaları oluşturulur.Örnek 1'in klasör yapısı Şekil 10.2'deki gibi düzenlenir. __pycache__ klasörünü Python'un kendisi modül yazınca otomatik atmaktadır.



Şekil 10.2: *Klasör yapısı*

Örnek 1

```
# modul.py
def cagir():
print("Merhaba Öğretmen Arkadaşım")
```

Oluşturduğunuz modülü çağırmak için;

Import modül adı şeklinde çağrılır.

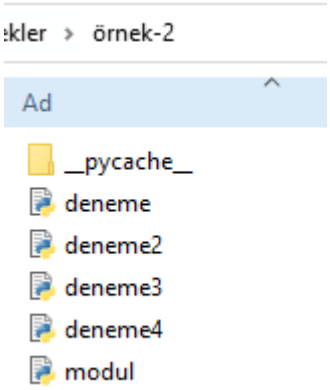
Başka bir Python dosyası oluşturulur. Buna da ana.py adı verilir. Bu dosya çalıştırıldığında ekran çıktısı aşağıdaki gibi olacaktır. Modülün içinde neler olduğunu incelediğimizde "dir" fonksiyonu ile modülün içindeki metotları görmekteyiz. Listenin son elemanı ise oluşturulan cagir fonksiyonudur.

```
#ana.py
import modul
print(dir(modul))
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__
package__', '__spec__', 'cagir']
```

Cagir foksiyonunu kullanmak için **deneme.py** adında bir Python dosyası oluşturunuz. Bu Python dosyasının içine aşağıdaki kodları yazınız.

```
#deneme.py
import modul #modülü çağırdık
modul.cagir()#modül içindeki fonksiyonu çalıştırdık
Merhaba Öğretmen Arkadaşım
```

Bir modül boş bir dosyadan oluşabileceği gibi, çeşitli değişkenlerin bulunduğu bazı işlevlerin yazıldığı, bir veya birden fazla sınıfın bulunduğu karmaşık yapıya da sahip olabilir. Örnek 2'de değişkenler tanımlanıp, bu değişkenler modül yardımıyla çağrılmıştır. Modülünüzü oluşturup **modul.py** dosyası olarak kaydedebilirsiniz.



Şekil 10.3: Örnek 2 klasör yapısı

Örnek 2

```
#modul.py
ad = 'Cemal'
soyad = 'Türk'
yas=60
meslek='Kaptan'
```

Artık yazılan modül içerisindeki değişkenlere ulaşılabilir. Eğer bir modülü bu şekilde adını kullanıp eklenirse modül içerisindeki nesnelere <modüladı>.<nesneadı> şeklinde ulaşabilme imkanı sağlar. Şimdi modülü çağırmak için deneme.py adlı Python dosyası oluşturulur. Bu dosyanın içine aşağıdaki kodları yazılır. Görüldüğü üzere modülünüzü modül adı, değişken adı şeklinde çağırarak **deneme.py** adlı Python dosyasında çalıştırabilirsiniz.

```
#deneme.py
import modul
print('Adı    :',modul.ad)
print('Soyadı :',modul.soyad)
print('Yaşı   :',modul.yas)
print('Mesleği:',modul.meslek)
Adı    :Cemal
Soyadı :Türk
Yaşı   :60
Mesleği:Kaptan
```

Program içinde sadece nesneleri sadece ismi ile çağırmak istenebilir. Bundan dolayı modülde sadece o nesneyi çağırmak gerekebilir.

Kullanımı: from <Modül adı> import <İçericek Nesne>

Örnek 2 klasörünün içinde **deneme2.py** dosyası oluşturulur. Bu dosyanın içerisine aşağıdaki kodları yazalım. Sadece kişiye ait meslek bilgisini çekmek istediğimizi farz ediyoruz. Ekrana böylelikle modül sayfamızdaki sadece meslek nesnesinin değeri gelmiş olur. Bu şekilde modül içindeki sadece tek nesneye değil de diğer nesnelere ulaşmak istenildiğinde ise hata verecektir.

```
#deneme2.py
from modul import meslek
print('Mesleği:',meslek)
Mesleği:Kaptan
```

Tüm nesneler kullanılmak istendiğinde ise; **from <modül> import*** şeklinde tanımlama gerekmektedir. **deneme3.py** adında bir Python dosyası oluşturarak, içine aşağıdaki kodları yazınız. Bu şekilde kullanımında **modul.py** dosyasındaki nesneleri ekrana yazdırmak için tekrar modül ismini yazmamız gerekmektedir. Sadece nesne ismini yazarak da çağırabilirsiniz.

```
#deneme3.py
from modul import*
print('Adı    :',ad)
print('Soyadı :',soyad)
print('Yaşı   :',yas)
print('Mesleği:',meslek)
Adı    : Cemal
Soyadı : Türk
Yaşı   : 60
Mesleği: Kaptan
```

Tüm nesneleri kullanmak istiyoruz ve modülümüze takma isim verilerek de çağırma işlemi gerçekleştirebilir.

Kullanımı: import modul_adi as takma_adi

Örnek 2 klasörümüzün içinde deneme4.py adlı bir Python dosyası oluşturarak içine aşağıdaki kodları yazıyoruz. Böylelikle modülümüzü takma ad ile birlikte çağırmış oluyoruz.

```
#deneme4.py
import modul as mod
print('Adı    :',mod.ad)
print('Soyadı :',mod.soyad)
print('Yaşı   :',mod.yas)
print('Mesleği:',mod.meslek)
Adı    : Cemal
Soyadı : Türk
Yaşı   : 60
Mesleği: Kaptan
```

10.2. Hazır Modülleri Kullanımı

Python'da herhangi bir hazır modülü kullanabilmek için öncelikle o modülü içe aktarılması gerekmektedir. İçe aktarmak bir modül içindeki fonksiyon ve niteliklerin başka bir program (veya ortam) içinden kullanılabilir hale getirilmesi anlamını taşımaktadır. Hazır modülleri kullanmak için modül ismi ile import edilerek çağrılmaktadır. Modül içindeki nesneler ise adlarıyla birlikte kullanılabilmektedir.

Kullanımı: import hazır modül adı

Örnek 3'te math modülünü projeye dâhil ediniz. Modüle ait özellikleri listeleyiniz.

Örnek 3

```
import math
print(dir(math))
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi',
'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

10.3. Random ve Math Kütüphaneleri

Örnek 3'teki math modülüne ait nesneleri adlarını kullanarak projeye ekleyebilirsiniz.

Math Kütüphaneleri

Math.pi, bize pi sayısını ve **math.pow** ise sayının üssünü alarak değerini vermektedir. Bu değerleri kullanarak dairenin alanını Örnek 4'te bulmaya çalışınız. Örnekte önce math modülümüzü çağırdık. Yaricap değişkenine 4 değerini atayalım. Dairenin alanını math.pi ve math.pow özelliklerini kullanarak alan değişkenine atayarak, ekrana yazdırılmıştır.

Örnek 4

```
import math
yaricap=4
alan=math.pi*(math.pow(yaricap,2))
print(alan)
50.26548245743669
```

Örnek 5'e bakıldığında, math modül içerisindeki cos(), factorial( ), pow( ), sqrt( ) gibi fonksiyonları çağırarak çeşitli işlemler yapılmıştır.

Örnek 5

```
import math
ustal=math.pow(5,2)
fak=math.factorial(4)
cosinus=math.cos(120)
karekok=math.sqrt(81)
print(ustal)
print(fak)
print(cosinus)
print(karekok)
25.0
24
0.8141809705265618
9.0
```

Random Kütüphanesi, Python'da rastgele sayı üretilmesini sağlamaktadır.

Örnek 6'da, random kütüphanesi çağırarak, bu kütüphane içinde random modülüne ait özellikleri listeleyiniz.

Örnek 6

```
import random
print(dir(random))
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom', 'TWOPI', '_BuiltinMethodType', '_MethodType', '_Sequence', '_Set',
'__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__
name__', '__package__', '__spec__', '_acos', '_bisect', '_ceil', '_cos', '_e',
'_exp', '_inst', '_itertools', '_log', '_os', '_pi', '_random', '_sha512', '_
sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'betavariate',
'choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits',
'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random',
'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform',
'vonmisesvariate', 'weibullvariate']
```

Örnek 7'de Random kütüphanesini kullanarak içindeki nesneler yardımıyla program yazınız. Modülün bu metodu kullanıldığı zaman 0 ile 1 arasında rastgele bir sayı üretilir. Üretilen bu sayı 0 olabilirken 1 olamaz; yani [0, 1) aralığında oluşmaktadır.

Örnek 7

```
import random
a=random.random()
print(a)
0.2980492500422903
```

Eğer integer tipinde bir sayı istenirse bu durumda randint metodu kullanmak gerekmektedir. Bu metot kullanılırken başlangıç ve bitiş aralık değerleri verilir; ancak bu durumda bitiş değeri de rastgele sayı olarak tutabilir.

Kullanımı: [başlangıç, bitiş] aralığı kullanılır.

Örnek 8'de Random modülü ile 0 dâhil değil 10 dahil olmak üzere sayı üretmesi sağlanır. Rastgele oluşturulan sayıyı değişkene aktararak, ekrana yazdırınız.

Örnek 8

```
import random
sayi=random.randint(1, 10)
print(sayi)
3
```

random.choice, ile listeden rastgele bir öge seçilmesine izin vermektedir. Örnek 9'da cicekler adında bir liste tanımlayınız ve elemanlarını ekleyiniz. Liste içerisinden rastgele 3 adet eleman seçerek, ekrana yazdırılmıştır.

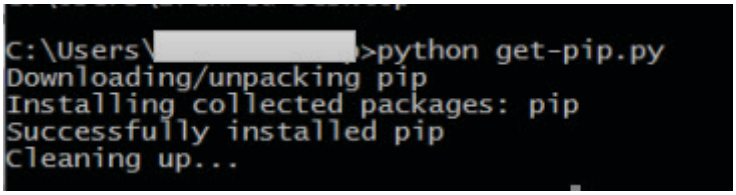
Örnek 9

```
import random
cicekler = ['gül', 'karanfil', 'papatya''begonya','menekşe']
print(random.choice(cicekler))#1. rastgele seçim
print(random.choice(cicekler))#2. rastgele seçim
print(random.choice(cicekler))#3. rastgele seçim
gül
menekşe
menekşe
```

10.4. Pip Paket Yükleyici Kullanımı

Pip bir paket yöneticisidir. Python kütüphane ve modüllerini kurmanızı sağlayan yardımcı bir programdır. Python'da standart paketlerin dışında bir de üçüncü parti modülleri vardır. Bunlar Python geliştiricileri haricindeki kişilerce yazılıp kullanımımıza sunulmuş araçlardır. Bu paketler, standart paketlerin aksine dilin bir parçası olmadığından, bu paketleri kullanabilmek için, pip yardımıyla kurmanız gerekmektedir. Bu üçüncü parti birçok modüllere https://pypi.python.org/pypi adresinden adını öğrenerek, paketleri yüklenebilir. Pip paket yükleyicisini kullanabilmek amacıyla öncelikle pip paket yüklecisinin yüklenmesi gerekmektedir. Pip yükleyicisinin kurulumunu için python yorumlayıcısından yapabilmekteyiz.

Öncelikle https://pip.pypa.io/en/latest/installing.html adresine giderek adreste gösterilmiş olan "get-pip.py" dosyasını bilgisayara indirilmesi gerekmektedir. İşletim sistemine bağlı olarak komut satırını açıp python yorumlayıcısını çalıştırarak Python yorumlayıcısına aşağıdaki komut yazılarak pip yükleyicisinin kurulumu gerçekleştirilir.



Şekil 10.4: Pip Paket yükleyici ekranı

Pip ile Paket Nasıl Yüklenir ve Kaldırılır?

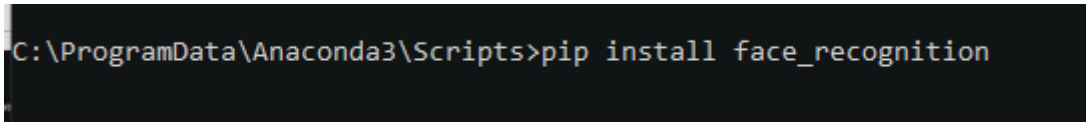
Pip ile paket kurmak için cmd açılmalı ve cmd ekranına şu kod yazılmalıdır:

```
pip install paket_adı              # paketi kurmak için
pip uninstall paket_adı            # kurulu paketi kaldırmak için
pip install PaketAdi==1.0.4        # istenilen versiyonu kurar
$ pip install 'PaketAdi>=1.0.4'    # alt limit ile verilen versiyonu kurar
```

install : Yeni bir paket yükler.
Uninstall : Varolan bir paketi siler.
Freze : Yüklü tüm paketleri requirements formatında listesini çıktıya verir.
List : Yüklü tüm paketleri normal listesini çıktıya verir.

Show : Yüklü paketler hakkında bilgi verir.
Search : Paketler içinde arama yapar.
Wheel : Paketler içinde requirements için bir arşiv yapar.
Upgrade : Kurulu bir paketi güncellemek için kullanılır.

Python2'de pipi kullanmak için pip2 ve Python3'te kullanmak için de pip3 komutu kullanılır. Cmd ekranı kullanılarak Şekil 10.5'te görüldüğü üzere "face_recognition" paketini yükleyebiliriz. Kaldırmak içinde aynı şekilde "uninstall" komutu kullanılır.



Şekil 10.5: *face_recognition paketini yükleme ekranı*

Örnek 10

Pip ile django paketini yükleyip, listele işlemini yaparak, yüklü paketler hakkında bilgi alalım.

```
pip install django
Collecting django
  Downloading Django-3.0.5-py3-none-any.whl (7.5 MB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.3.1-py2.py3-none-any.whl (40 kB)
Collecting asgiref~=3.2
  Downloading asgiref-3.2.7-py2.py3-none-any.whl (19 kB)
Requirement already satisfied: pytz in c:\programdata\anaconda3\lib\site-packages
(from django) (2019.3)
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.2.7 django-3.0.5 sqlparse-0.3.1
Note: you may need to restart the kernel to use updated packages.
```

Örnek 11

pip install django ile kurduğunuz paketi yükseltmeye çalışabilirsiniz.

```
pip install django --upgrade
Requirement already up-to-date: django in c:\programdata\anaconda3\lib\site-packages
(3.0.5)
Requirement already satisfied, skipping upgrade: pytz in c:\programdata\anaconda3\
lib\site-packages (from django) (2019.3)
Requirement already satisfied, skipping upgrade: asgiref~=3.2 in c:\programdata\
anaconda3\lib\site-packages (from django) (3.2.7)
Requirement already satisfied, skipping upgrade: sqlparse>=0.2.2 in c:\programdata\
anaconda3\lib\site-packages (from django) (0.3.1)
Note: you may need to restart the kernel to use updated packages.
```

Örnek 12

Yüklediğiniz django paketini kaldırabilirsiniz.

```
pip uninstall django
```

10.5. Time Modülü

Time modülü zaman değerlerini düzenlemekle ilgili birçok görevi yerine getirebilmektedir. Zamanı göstermek için iki standart bulunmaktadır. Birincisi zamanı Epoch'tan itibaren saniye olarak vermektir. Epoch Unix zaman başlangıcı olarak alınır ve takvime göre 1 Ocak 1970'e denk gelmektedir. Eğer Epoch türünden şimdiki zaman saniye biçiminde alınmak istenirse şu yöntemi kullanabilirsiniz.

Örnek 13

```
import time
print (time.time())
1587590353.7686868
```

Ekran çıktısı bize 1 Ocak 1970'ten itibaren kaç saniye geçtiğini vermektedir. Elde edilen değeri **gmtime( )** fonksiyonu kullanarak okunabilir tarih formatına çevrilebilir. Örnek 14'te saniye cinsinden elde edilen zaman değeri, okunabilir tarih formatına çevrilmiştir.

Örnek 14

```
import time
print (time.gmtime(time.time()))
time.struct_time(tm_year=2020, tm_mon=4, tm_mday=22, tm_hour=21, tm_min=44, tm_
sec=9, tm_wday=2, tm_yday=113, tm_isdst=0)
```

Localtime, zaman bilgilerini sıralı bir tüp şeklinde vermektedir. Örnek 15'te time modülünde localtime nesnesini kullanarak zaman bilgilerini sıralı bir şekilde işlemi yapmaktadır.

Örnek 15

```
import time
print (time.localtime())
time.struct_time(tm_year=2020, tm_mon=4, tm_mday=23, tm_hour=0, tm_min=25, tm_
sec=31, tm_wday=3, tm_yday=114, tm_isdst=0)
```

ctime fonksiyonu, içinde bulunulan zaman bilgilerini vermektedir. Örnek 16'da ctime fonksiyonu ile güncel tarih bilgilerini yazmaktadır.

Örnek 16

```
import time
print (time.ctime())
Thu Apr 23 00:36:07 2020
```

strftime( ) fonksiyonu ile kendimize ait zaman cümlesi oluşturabilirsiniz. Bu zaman cümlesinin belirlediğimiz duruma göre ekran çıktısının verilmesini sağlar. Tablo 1'de strftime() fonksiyonuna ait yönergeler verilmektedir. Bu yönergeler ister tek başına istersek yönergeleri birleştirerek de kullanabiliriz.

Tablo 1. strftime() fonksiyonu yönergeleri ve anlamları

Yönerge	Anlamı
%a	Kısaltılmış gün adı
%A	Gün adı
%b	Ayın kısaltılmış adı
%B	Ayın adı
%c	Tam tarih ve saat
%d	Ayın günü (01-31)
%H	Saat (00-24)
%I	Saat (01-12)
%j	Gün (01-366)
%m	Ay (00-12)
%M	Dakika (00-59)
%p	Öğleden önce (ÖÖ), öğleden sonra (ÖS)
%S	Saniye (00-59)
%U	Yılın kaçıncı haftası (00-53)
%w	Haftanın kaçıncı günü (0-6)
%y	Yılın son iki hanesi (15)
%d	Ayın günü (örnek: Nisan için 13)
%Y	Yıl

Örnek 17'de gün ay yıl yönergelerini kullanarak yan yana yazımı verilmiştir. Bu şekilde yönergeleri ekleyerek, güncel tarih bilgilerini strftime( ) fonksiyonu ile alabiliriz.

Örnek 17

```
import time
print (time.strftime("%d/%m/%Y"))
23/04/2020
```

sleep() Fonksiyonu, programın belirlenen süre boyunca durdurulmasına olanak sağlar. Aldığı argüman saniye cinsindendir. Örnek 18'deki kodlar çalıştırıldığında 10 saniye program duraklar.

Örnek 18

```
import time
time.sleep(10)
```

Örnek 19'da ise bugünün tarihini saniyede içinde olacak şekilde verilmiştir. Ekran çıktısı olarak başlangıç tarihi ile bitiş tarihi arasında 5 saniye olduğu görülmektedir.

Örnek 19

```
import time
print ("Başlangıç : %s" % time.ctime())
time.sleep( 5 )
print ("Bitiş : %s" % time.ctime())
Başlangıç : Thu Apr 23 01:19:27 2020
Bitiş : Thu Apr 23 01:19:32 2020
```

10.6. Bölüm Sonu Örnekleri

1. 0-20'ye kadar sayıları döngü yapısı ile ekrana yazdırırken her bir sayı arasına yarım saniye duraklamalar koyarak programı yapınız.

2. Os modülünde name fonksiyonu işletim sistemi hakkında bilgi vermektedir. İşletim sisteminiz hakkında bilgi almak için gerekli olan kodları yazınız.

3. İki sayının toplamını yaptıran programı fonksiyon ve modül yazma kullanarak yapınız.

Cevaplar

1.
```
import time
for i in range(1,21):
time.sleep(0.5)
print(i)
```
```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

2.
```
import os
print (os.name)
```
```
Nt
```

3. **#modul.py dosyası;**

```
def toplama(sayi1,sayi2):
    toplam=int(sayi1)+int(sayi2)
    return toplam
```

#deneme.py dosyası;

```
import modul
a = input("birinci sayıyı giriniz:")
b = input("ikinci sayıyı giriniz:")
sonuc = int(modul.toplama(a, b))
print (sonuc)
birinci sayıyı giriniz:6
ikinci sayıyı giriniz:7
13
```

MODÜL 11

İLERİ SEVİYE VERİ YAPILARI

Şekil 11.1: Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.

11.1. Sayıların İleri Seviye Özellikleri

Bu bölümde sayı veri tipleri derinlemesine incelenecek, ayrıca sayılarla ilgili bazı fonksiyonlar anlatılacaktır.

Onluk tabandaki bir sayıyı ikilik tabana çevirme:

10'luk sistemdeki sayıları 2'lik sistemdeki bir sayıya çevirmek için **bin( )** fonksiyonu kullanılır.

Aşağıdaki örnek kodları etkileşimli kabuk üzerinde çalıştırınız.

```
bin(5)
>>> '0b101'
```

Buradaki **0b** ifadesi sayının ikilik tabanda olduğunu belirtmektedir. 101 ifadesi ise sağdan itibaren "iki üzeri sıfır çarpı 1 + iki üzeri bir çarpı 1 + iki üzeri iki çarpı 2 şeklinde tanımlanmaktadır." Bu işlemin sonucu $(2^0*1) + (2^1*0) + (2^2*1) = 5$ olarak hesaplanmaktadır.

```
bin(12)
>>> '0b1100'
```

$(2^0*0)+(2^1*0)+(2^2*1)+(2^3*1) = 12$

şeklinde hesaplanmıştır.

Onluk tabandaki bir sayıyı onaltılık tabana çevirme:

10'luk sayı sistemdeki sayıları 16'lık sistemdeki bir sayıya çevirmek için **hex( )** fonksiyonu kullanılır.

```
hex(18)
>>> '0x12'
```

Buradaki **0x** ifadesi sayının 16'lık sistemde olduğunu göstermektedir.

```
hex(18)
>>> '0x12'
```

$(16^0*2)+(16^1*2) = 2+ 16 = 18$

olarak hesaplanır.

11.2. Sayılar Üzerinde Uygulanabilen Fonksiyonlar

abs( ) Fonksiyonu: İngilizcede 'mutlak' anlamına gelen **absolute** adlı bir sözcüğünün kısaltmasıdır. **abs( )** fonksiyonu bir sayının mutlak değerini almak için kullanılır. Fonksiyon tek parametre alarak sayının mutlak değerini alır.

```
abs(-10)
>>> 10

abs(15)
>>> 15

abs(-3.2)
>>> 3.2
```

round( ) Fonksiyonu: round( ) fonksiyonu bir sayıyı belirli kriterlere göre yukarı veya aşağı yuvarlamak için kullanılır.

```
round(5.7)
>>> 6

round(5.3)
>>> 5

round(5.5)
>>> 6
```

Eğer girilen değerin virgülden sonraki kısmı 5 ve üzerinde bir değerse yukarı, altında bir değerse aşağı yuvarlama işlemi yapar.

chr( ) Fonksiyonu: chr( ) fonksiyonu, kendisine parametre olarak verilen bir tam sayının karakter olarak **(ASCII)** karşılığını verir.

```
chr(65)
>>> 'A'

chr(100)
>>> 'd'
```

max( ) Fonksiyonu: max( ) fonksiyonu, bir dizi içindeki sayıların en büyüğünü verir.

```
max(3,5,7,8,9)
>>> 9

liste=[3,8,2,6,15]
max(liste)
>>> 15
```

min( ) Fonksiyonu: min( ) fonksiyonu, **max( )** fonksiyonunun tam tersi işlem yapar.

```
min(4,7,1,2,9)
>>> 1

liste=[4,7,6,9,3]
min(liste)
>>> 3
```

pow( ) Fonksiyonu: pow( ) fonksiyonu power sözcüğünün kısaltmasında türetilmiştir. Bir sayının üssünü almak için kullanılır. İki adet parametre alır, birinci parametre üssü alınacak sayıyı, ikinci parametre ise kuvvetini ifade eder.

```
pow(3,4)
>>> 81
pow(9,0.5)
>>> 3.0
```

sum( ) Fonksiyonu: sum( ) fonksiyonu, dizi içerisindeki değerlerin toplamını bulmamızı sağlar. **sum( )** fonksiyonuna girilen değerler liste ya da tupple türünden olması gerekmektedir.

```
sum([5,3,8,6])
>>> 22
```

11.3. İleri Seviye Karakter Dizileri (String)

Karakter dizilerinin özel metotları:

replace( ) Fonksiyonu: replace( ) fonksiyonu, bir karakter dizisi içindeki karakterleri başka karakterlerle değiştirmeyi sağlar. **replace( )** fonksiyonu iki adet parametre alır. Birinci parametre değişecek karakter ya da karakterleri, ikinci parametre ise yerine gelecek karakter ya da karakterleri ifade eder.

```
a="python"
a.replace("p","P")
>>> 'Python'
```

split( ) Fonksiyonu: split( ) fonksiyonu bir karakter dizisini verilen kurala göre bölme işlemi yapar. Eğer **split( )** fonksiyonuna parametre verilmezse boşluk karakterine göre yapar.

```
a="Milli Eğitim Bakanlığı"
a.split()
>>> ['Milli', 'Eğitim', 'Bakanlığı']
```

Ya da belirli kriterlere göre bölme işlemi de yapılabilir.

```
b="T.B.M.M"
b.split(".")
>>> ['T', 'B', 'M', 'M']
```

upper( ) ve lower( ) fonksiyonları: Bu fonksiyonlar karakter dizilerini büyük veya küçük harfe çevirme işlemini yapar.

```
"Merhaba dünya".upper( )
>>> 'MERHABA DÜNYA'

"Merhaba DÜNYA".lower( )
>>> 'merhaba dünya'
```

join( ) fonksiyonu: **split( )** fonksiyonunun tam tersi işlem yapar. Liste içerisinde bulunan karakter dizilerini verilen kurala göre birleştirmek için kullanılır.

```
"-".join(["Merhaba","Dünya"])
>>> 'Merhaba-Dünya'
```

capitalize fonksiyonu: **capitalize( )** fonksiyonu karakter dizilerinin sadece ilk harfini büyük yapmak için kullanılır.

```
a="python programlama dili"
a.capitalize( )
>>> 'Python programlama dili'
```

find( ) fonksiyonu: find( ) fonksiyonu karakter dizisi içerisindeki bir karakterin konumunu sorgular. Bulduğu ilk değeri döndürür.

```
a="armağan"
a.find("a")
>>> 0
```

İlk bulduğu indis değerini verir.

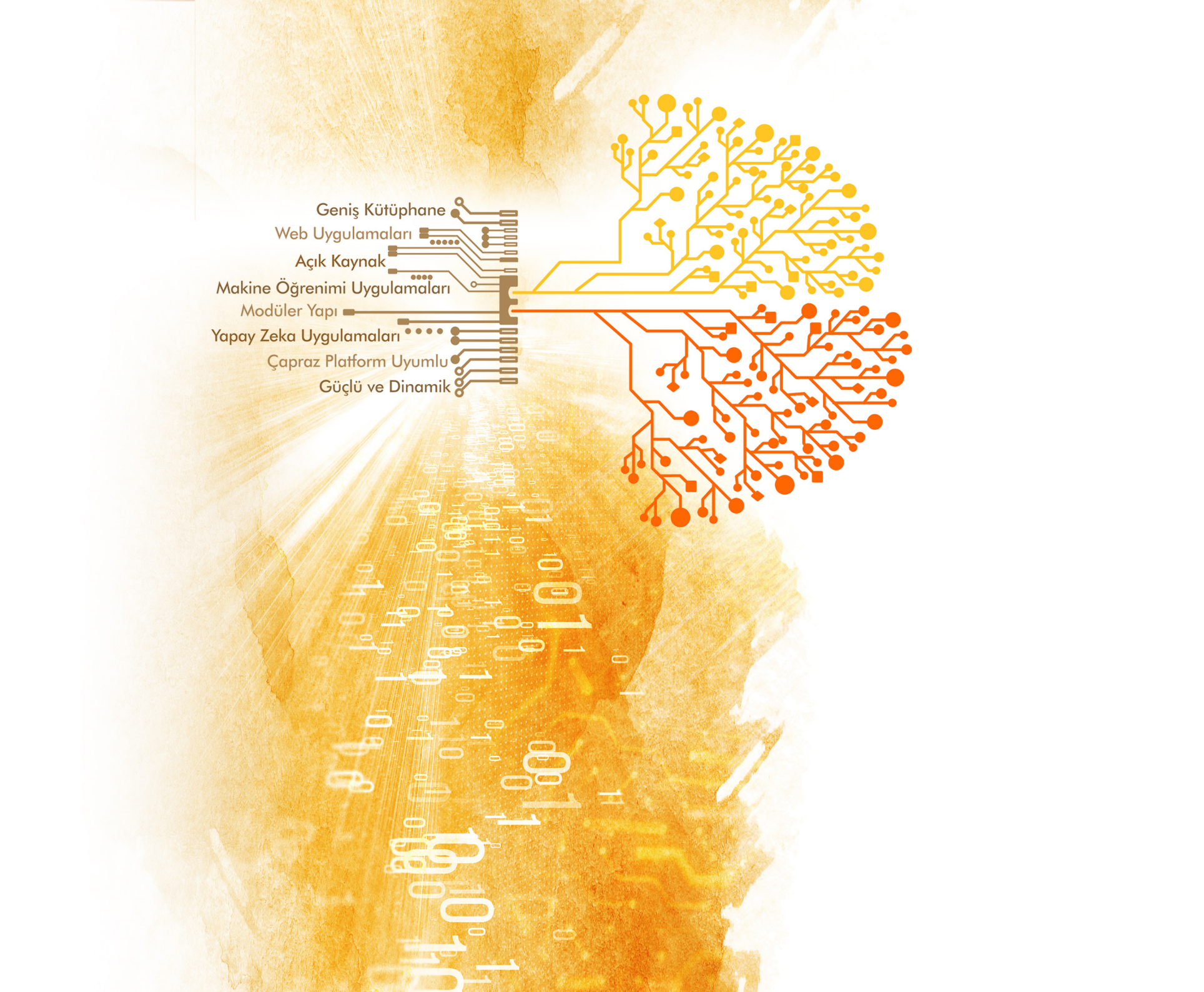
rfind( ) fonksiyonu: find( ) fonksiyonun benzeri işlemi yapar ancak arama işlemini sağ taraftan başlayarak yapar.

```
a="armağan"
a.rfind("a")
>>> 5
```

isdigit( ) fonksiyonu: isdigit( ) fonksiyonu, karaktere dizisinin bir sayısal değer olup olmadığını kontrol eder. Eğer karakter dizisindeki tüm karakterler rakamdan oluşuyorsa True, değilse False değeri döndürür.

```
metin ="12345"
print(metin.isdigit())
>>> True

metin ="asd123"
print(metin.isdigit())
>>> False
```


Geniş Kütüphane
Web Uygulamaları
Açık Kaynak
Makine Öğrenimi Uygulamaları
Modüler Yapı
Yapay Zeka Uygulamaları
Çapraz Platform Uyumlu
Güçlü ve Dinamik

MODÜL 12

DOSYALARLA ÇALIŞMAK

Şekil 12.1: Bölümle ilgili örnek uygulamalara karekod' dan ulaşabilirsiniz

Bu bölüme kadar Python ile temel programlama kavramları ile tanışıp örnekler üzerinden uygulamalar yaptınız. Koşul yapısı, döngüler listeler fonksiyonlar gibi kavramlar ile tanışıp program yazarken kullandınız. Ayrıca kullanıcıdan veri alıp işlediniz ve bir sonuç / çıktı ürettiniz. Peki elde ettiğiniz sonuç/çıktıları kalıcı olarak depolayabildiniz mi veya depolanmış veriyi programınıza dâhil edebildiniz mi?

Bu bölümde elde ettiğiniz çıktıları dosyaya nasıl kaydedebileceğinizi, dosyadan nasıl veri okuyacağınızı ve iki işlemi birlikte nasıl yapabileceğinizi öğreneceksiniz. Ayrıca var olan dosyalara erişip üzerinde güncelleme yapmayı öğreneceksiniz. **Bu bölümdeki uygulamalar idle ile yazılmıştır. Oluşturulan dosyalar ve kodlar aynı klasöre kaydedilmiştir.**

Dosyalar verilerin kalıcı olarak depolandığı ve ihtiyaç duyulduğunda okunabildiği temel yapılardandır. Verilerin depolanması ve üzerinde işlemler yapılması için kullanılan gelişmiş yapılar veritabanlarıdır. Veritabanı konusu modül on dörtte işlendiğinden bu bölümde sadece dosyalar ile ilgili kısmına değinilmiştir.

Dosyalarla çalışma mantığı ve süreci Şekil 12.2'de gösterildiği gibi üç aşamadan oluşmaktadır. Öncelikle dosya amaca uygun kipte açılır. Ardından veri okuma-yazma gibi işlemler yapılır. Son olarak dosya kapatılır.

Şekil 12.2: *Doyalar ile çalışma süreci*

12.1. Dosya İşlemleri

Python'da dosya işlemleri Python'un içinde bulunan hazır nesneler ile yürütülür. Bunun için metin sarma (text wrapper) nesnesi oluşturulur.

dosya=open(dosya adı, dosya açma türü)

Python'da bir dosyayı açmak için **open( )** fonksiyonu kullanılır. open fonksiyonu içine dosya adı ve erişim türü girdilerini alır. Bunu şöyle düşünebilirsiniz:

Hangi dosyayı açayım? Hangi amaç için açayım?

Hangi dosyayı açayım sorusunun cevabı adresini belirttiğiniz dosyadır. Bir dosya hangi amaçlar için açılır derseniz bunlar:

veri okuma, veri yazma ve ikisinin (okuma ve yazma) birlikte olduğu durumlardır.

dosya ile işimiz bittiği zaman dosyayı **close()** fonksiyonu ile kapatırız.

Ek bilgi olarak karakter kodlaması içinde 3. parametre olarak encoding kullanılır. Türkçe karakterlerde sorun yaşamamak için **encoding="utf-8"** ifadesi eklenir.

Dosya işlemlerini yaparken 2 farklı yöntem kullanabiliriz. Tablo 1'de gösterildiği gibi dosyayı kendimizin açıp kendimizin kapattığı yöntem ya da **with open** parametresi kullanılarak dosyanın sürecin bitiminde otomatik kapandığı yöntem kullanılabilir. İki yöntemi de kullanabilirsiniz.

Tablo 1: Dosya işlemleri süreci yöntemleri

dosya=open (dosya adı, açma kipi) işlemler dosya.close ()	with open(dosya adı, açma kipi) as dosya: **işlemler** # girinti kullanmaya dikkat

Dosya işlemlerine başlamadan önce ilerleyen örneklerde kullanılmak üzere deneme.txt, veri.txt gibi dosyaları Python kodlarının olduğu ve .py uzantılı olan kod dosyası ile aynı klasörde oluşturunuz. Böylece dosyanın sadece adını yazarak bilgisayarda bulunduğu yeri yazmaya ihtiyaç duymadan açabiliriz. Peki ya, dosyayı hangi amaç için açacağız ve amacımıza göre açma yöntemi hangisidir? derseniz cevabı dosya kipleri olacaktır.

12.2. Dosya Kipleri

Dosyalara erişme amaçlarımız veri okuma, yazma ve iki amacın birlikte olduğu durumlardır. Bu amaçları karşılamak üzere Python'da dört farklı dosya açma kipi bulunmaktadır.

1. Dosyadan sadece veri okumak istiyor isek **r** (read) kipinde açarız. Bu kipte yazma işlemi yapmaya çalışırsak hata ile karşılaşırız.
2. Dosyaya sadece veri kaydetmek/yazmak istiyor isek **w** (write) kipinde açarız. Bu kipte okuma işlemi yapmaya çalışırsak hata ile karşılaşırız.
3. Dosyadan hem veri okumak hem de dosyaya veri yazmak istiyor isek **r+** kipinde açarız. Bu kipte dikkat etmemiz gereken nokta dosya oluşturulmamış ise hata ile karşılaşırız.
4. Dosyadan hem veri okumak hem de dosyanın sonuna veri yazmak istiyor isek **a** (append) kipinde açarız. a kipinde dosyayı açtığımızda dosya yok ise oluşturur r+ olduğu gibi hata vermez.

Tablo 2'de dosya okuma kipleri ve özellikleri hakkında karşılaştırmalı ve detaylı bilgiler verilmiştir.

Tablo 2: Dosya okuma kipleri ve özellikleri

Dosya açma kipi	Dosya açma kodu	Dosya mevcutsa ve yoksa ne olur
yazma kipinde dosya açma w (write)	dosya=open("deneme.txt","w") işlemler dosya.close()	yoksa oluşturulur. var ise dosya oluşturulur ve eski dosya silinir. Eski dosyada bilgi var ise gider.
ekleme kipinde dosya açma a (append) (ekleme)	dosya=open("deneme.txt","a") işlemler dosya.close()	yoksa oluşturulur. var ise dosya oluşturulmaz. dosyanın sonuna ekleme yapılır. eski bilgiler silinmez. Sadece okuma işlemi yapamazsınız.
okuma kipinde dosya açma r (read)	dosya=open("deneme.txt","r") işlemler dosya.close()	yoksa hata üretilir. var ise okunur
Okuma ve yazma (birlikte) Dosya açma r+	dosya=open("deneme.txt","r+") işlemler dosya.close()	yoksa hata üretilir. var ise dosya oluşturulmaz dosyanın istenen bölümlerine eklemeler yapılır. eski bilgiler silinmez

12.3. Dosya Okuma Yazma İşlemleri

Bu bölümde dosya okuma ve yazma örnekleri bulunmaktadır. Öncelikle deneme.txt isimli bir dosya oluşturarak içine "bu basit bir dosyadır" cümlesini yazmalısınız. Örnekleri uyguladıktan sonra deneme.txt dosyasında nelerin değiştiğini görmek için dosyayı açmalısınız. Dosyayı açıp inceledikten sonra kapatmayı unutmayınız.

NOT

"\n" ifadesi print( ) örneklerinden hatırlayacağınız gibi alt satıra geçmek için kullanılmaktadır.

İlk olarak oluşturduğunuz deneme.txt dosyası okuma kipinde açılacak ve içindeki verileri ekranda görüntülenecektir.

Örnek 1

Dosyayı okuma kipinde açma:

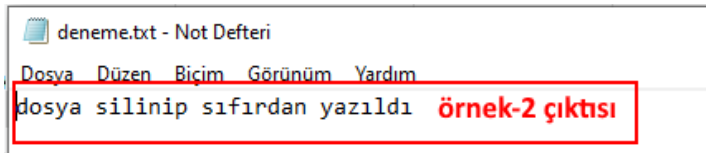
```
dosya = open ("deneme.txt","r")
belge=dosya.read()
print (belge)
dosya.close()
bu basit bir dosyadır
```

İkinci aşamada deneme.txt dosyası yazma kipinde açılıp içine veri kaydedilecektir. Yazma işlemini yaptıktan sonra Resim 12.3'te görüldüğü gibi dosyada önceden bulunan verilerin silindiğini fark ettiniz mi?

Örnek 2

Dosyayı yazma kipinde açma:

```
with open ("deneme.txt","w") as dosya:
dosya.write ("dosya silinip sıfırdan yazıldı ")
```



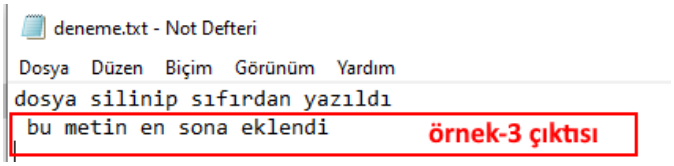
Resim 12.3: *Örnek 2 kod çıktısı*

Şimdi de dosyayı okuma ve yazma kipinde açıp neler olduğunu inceleyiniz.

Örnek 3

Dosyayı okuma ve yazma kipinde açma:

```
dosya = open("deneme.txt","r+")
belge=dosya.read()
print(belge)
dosya.write("\n bu metin en sona eklendi")
dosya.close()
dosya silinip sıfırdan yazıldı
```



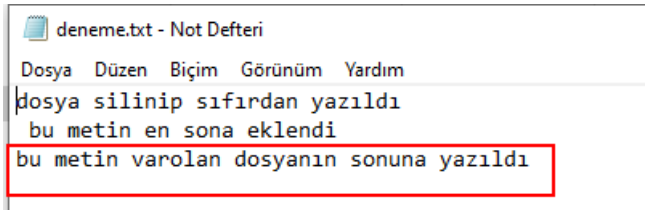
Resim 12.4: *Örnek 3 kod çıktısı*

Son olarak dosyayı genişleme kipinde açıp neler olduğunu inceleyiniz.

Örnek 4

Dosyayı genişletme kipinde açma:

```
dosya = open ("deneme.txt","a")
dosya.write ("bu metin dosyanın sonuna yazıldı")
dosya.close()
```



Resim 12.5: *Örnek 4 kod çıktısı*

Python'da dosyalara yazılan veriler string (metin) türünde kaydedilmektedir. Dosya erişim örneklerini öğrendikten sonra programlamada temel algoritmalar olarak kabul edilen arama ve sıralama algoritmalarını kullanarak dosya işlemleri yapılabilir.

12.4. Arama ve Sıralama Algoritmaları ile Dosyadan Okuma ve Dosyaya Yazma

İlk örnek şifreleme (kriptoloji) ve şifre çözme biliminde önemli bir yeri olan asal sayı örneği olacaktır. Çok büyük sayıların asal olup olmadığının kontrolü için bilgisayara hesaplatmak yerine kayıtlı olan bir listeden bakmak daha kolay bir yöntemdir. Bu nedenle bunun bilgisayarı yormayacak uzunlukta bir örneği yapılacaktır. Birinci aşamada birden bine kadar olan asal sayılar hesaplanıp asal.txt isimli dosyaya kaydedilecektir. Diğer bir değişle dosya yazma kipinde açılacaktır. Bu durumda yapılacak olan işlem asal sayıların hesaplanıp dosyaya kaydedilmesi olacaktır. Asal sayı örneği döngüler konusunda anlatıldığından ek bir açıklamaya gerek duyulmamıştır. Dosya açma kipi: yazma kipi olacak. Asal sayılar bulunduktan sonra string türüne dönüştürülmüştür. Sayılar arasına bir boşluk bırakılmıştır. Böylece Örnek 6'da dosyadan asal sayılar okunduktan sonra split metodu ile sayıları liste veri tipinde tutulması sağlanmış olacaktır. Aynı mantık virgül gibi başka bir karakter kullanılarak yapılabilir.

Örnek 5

1'den 1000'e kadar olan asal sayıları bulup dosyaya kaydeden program.

```
asal_sayı=[2]
for sayı in range (3,1001):
    for bolen_sayı in range (2,sayı):
        sayı_asalmı=False
        if sayı % bolen_sayı==0:
            sayı_asalmı=True
            break
    if sayı_asalmı==False:
            asal_sayı.append(sayı)
```

```
veri=" "
for i in asal_sayı:
    veri+=str(i) # veri=veri+str(i)
    veri+=" "
dosya=open ("asalsayı.txt","w")
dosya.write(veri)
dosya.close()
asalsayı.txt nin içindeki veriler çok yer kaplayacağından buraya eklenmemiştir.
Dosyayı açıp sonucu inceleyebilirsiniz.
```

Asal sayılar bulunup dosyaya kayıt edildikten sonra kullanıcının girdiği sayının asal olup olmadığı dosyadan kontrol edilmektedir. Bunun için öncelikle dosya açılıp veriler okunmaktadır. Ardından okunan veriler split metodu kullanarak bir listeye kaydedilmektedir. Böylece kullanıcının girdiği sayının listede olup olmadığına bakılarak sayının asal olup olmadığını kontrol edilmiş olacaktır.

Örnek 6

Girilen sayının asal olup olmadığını dosyadan karşılaştıran program

```
with open("asalsayı.txt","r") as dosya :
    veri=dosya.read()
    asal_sayılar=veri.split(" ")
kontrol_sayısı=input("asal olup olmadığını kontrol etmek istediğiniz sayıyı
giriniz")
if kontrol_sayısı in asal_sayılar :
    print("asal sayı")
else:
    print("asal sayı değil")
asal olup olmadıgını kontrol etmek istediğiniz sayıyı giriniz22
asal sayı değil
```

Aklınıza bu iki örneği tek bir programda birleştirebilir miyiz? sorusu gelebilir. Örnek 7'de dosyaya kaydetme ve dosyada arama işlemlerinin birlikte yapılabileceği bir rezervasyon uygulaması yapılacaktır. Uygulamada rezervasyon yapma ve rezervasyon sorgulama işlemleri bulunmaktadır. Bu işlemler birer fonksiyon olarak tanımlanıp örnek programda kullanılmaktadır.

Örnek 7

Rezervasyon yapan ve rezervasyon kaydını kontrol eden program

```
def rezerevasyon_yap():
    rezervasyon_bilgi=input("rezervasyon bilgilerinizi giriniz")
    veri=rezervasyon_bilgi+","
    dosya=open("rezervasyon.txt","a")
    dosya.write(veri)
    dosya.close()

def rezerevasyon_kontrol() :
    rezervasyon_bilgi=input("rezervasyon bilgilerinizi giriniz")
    with open("rezervasyon.txt","r") as dosya:
        veri=dosya.read()
        rezervasyonlar=veri.split(",")
        if rezervasyon_bilgi in rezervasyonlar:
            print("rezervasyonunuz bulunmaktadır.")
        else:
            print("rezervasyon kaydınız yoktur.")

while True:
    islem=input("rezervasyon yapmak için  1 : kontrol için 2 programı kapatmak için
3 e basınız")
    if islem=="1" :
        rezerevasyon_yap()
    elif islem =="2":
        rezerevasyon_kontrol()
    else:
        break
```

Yukarıdaki örneği yaptıktan sonra aklınıza şu soru gelebilir peki yapılan dosyaya kaydedilen rezervasyon bilgileri üzerinde değişiklik (düzeltme, silme gibi) işlemleri yapılabilir mi? Bunun örneği ilerleyen örneklerde yapılacaktır.

12.5. Dosyaların Özel Metotları

read( ) fonksiyonunu dosyanın tamamını okumak için kullandık. Aynı metot dosyanın belirli bir kısmını okumak için de kullanılabilir. Örneğin, **read(10)** ile 10 byte veri okunur. Ayrıca dosyadan veri okurken for döngüsünün de kullanıldığını belirtmekte yarar var. Bir önceki bölümde anlatılan fonksiyonlarının dışında **seek(), writeline(), tell()** ve **writelines()** metotları bulunmaktadır.

seek() fonksiyonu imlecin dosyada istenen noktaya alınması için kullanılır. **tell()** fonksiyonu da imlecin güncel olarak nerede bulunduğunu öğrenmek için kullanılır. Microsoft word gibi bir kelime işlemci programından aşina olduğunuz imleç yazma veya okuma işlemlerinde aktif olan konumu ifade eder. Örnek 8'de tell(), seek(), read(girilen bayt) ve for döngüsü ile dosya okuma örneği verilmiştir.

Örnek 8

tell(), seek(), read(girilen bayt) ve for döngüsü ile dosya okuma örneği:

```
dosya=open("deneme.txt","r")
# dosyamızı for döngüsü ile okuyoruz
for veri in dosya:
    print(veri)
#imlecin nerede olduğunu ekrana yazdırıyoruz
print(dosya.tell())
# imleci 10. bayta taşıyoruz
dosya.seek(10)
#imlecin bulunduğu yerden 20 bayt veri okuyoruz
print(dosya.read(20))
#imlecin nerede olduğunu görüntülüyoruz
print(dosya.tell())
dosya.close()
bu basit bir dosyadırbu metin en sona eklendibu metin varolan dosyanın sonuna
yazıldı
85
ir dosyadırbu metin
30
```

Dosyalarda satır satır işlem yapabilmek için verileri de farklı satırlarda olacak şekilde kaydetmeniz gerekir. Bunu yapmanın en yaygın yolu verinin sonuna "\n" eklemektir.

writelines() fonksiyonu string içerikli listeleri dosyaya yazar. **readline()** fonksiyonu bir satır veri okur.

readlines() fonksiyonu dosyanın tamamını okuyup liste veri tipinde tutar. Böylece liste metodları kullanılabilir. Örnek 9-12'de bu fonksiyonların kullanımına dair örnek uygulamalar yapılmıştır.

Örnek 9

tell(), seek(), writelines() uygulaması

```
dosya=open("deneme.txt","r+")
dosya.seek(20)
#dosyada 20. bayta gittik
dosya.write("20. bayttan itibaren yazdık")
#20. bayttan sonraki verilerin üzerine yazdık
print(dosya.tell())
# imlecin 47. bayta geldiğini öğreneceğiz
#burda dosyadan okuma yaparsak 47. bayttan sonrası okunacak
print(dosya.read())
# listemizdeki verileri en sona yazıyoruz
liste=["1","2","3","4"]
#listedeki veriler string olmazsa hata alırız.
dosya.writelines (liste)
47
metin varolan dosyanın sonuna yazıldı
```

Örnek 10 ve 11 için İstiklal Marşı'nın 10 kıtasını kodlarla aynı klasörde olacak şekilde istiklal.txt isimli bir dosyaya kayıt ediniz.

Örnek 10

Dosya metotları uygulaması

```
dosya=open("istiklal.txt","r",encoding="utf-8")
# readlines ile içeriği liste olarak alacam
liste=dosya.readlines()
# listenin uzunluğunu hesaplayarak kaç satır olduğunu ekrana yazıyorum
print(len(liste))
#listenin 6. satırını ekrana yazıyorum
#listenin ilk satırının liste[0] olduğunu hatırlayın
print(liste[5])
#dosyada başa dönüyorum
dosya.seek(0)
# baştan sona satır satır okuyorum
for satır in range(len(liste)):
    print(dosya.readline())
dosya.close()
41
Kahraman ırkıma bir gül! Ne bu şiddet, bu celal?
İstiklal marşının 10 kıtasının tamamı ekrana yazılıyor (çok yer kaplamaması adına
yazılmamıştır)
```

Örnek 11'de dosyanın içinde belirli konumlara erişip veri ekleme örneği bulunmaktadır.

Örnek 11

Dosyanın içinde belirli yerlere veri ekleme

```
dosya=open("istiklal.txt","r+",encoding="utf-8")
# readlines ile içeriği liste olarak alacam
liste=dosya.readlines()
print(len(liste))
# her 4 kıtadan sonra araya bir satır çizgi ekliyoruz
# listeye 4 satırda bir ek satır eklediğimden beşer beşer ilerleyeceğim
eklenecek_satır=30*"-"+"\n"
for ekle in range(4,52,5):
    liste.insert(ekle,eklenecek_satır)
# dosyada başa gelecem
dosya.seek(0)
# çizgi eklenmiş listeyi yazdıracağım
dosya.writelines(liste)
dosya.seek(0)
yeni_liste=dosya.readlines()
print(yeni_liste)
dosya.close()
sonuç çok yer kaplayacağından burada gösterilmemiştir. Lütfen istiklal.txt
dosyasının inceleyiniz
```

Örnek 12'de özel fonksiyon kullanımını içeren rezervasyon uygulaması bulunmaktadır.

Örnek 12

Rezervasyon kapasite kontrolü uygulaması

```
dosya=open("rezervasyon.txt","r+",encoding="utf-8")
rezervasyonlar=dosya.readlines()
rezervasyon_kapasitesi=50
sıra_no=1
for rezervasyon_sahibi in rezervasyonlar:
    print(sıra_no,"nolu rezervasyon sahibi",rezervasyon_sahibi)
    sıra_no+=1
print("toplam",sıra_no,"adet rezervasyon var")
if rezervasyon_kapasitesi-sıra_no >0 :
    print(rezervasyon_kapasitesi-sıra_no,"adet daha  rezervasyon yapabiliriz")
else:
    print("rezervasyon kapasitemiz dolmuştur.")
dosya.close ()
```

Örnek 13'te özel fonksiyonlar ile rezervasyon yapma uygulaması örneği bulunmaktadır.

Örnek 13

Özel fonksiyonlar ile rezervasyon yapma uygulaması

```
def rezervasyon_yap():
    rezervasyon_kapasitesi=50
    dosya=open("rezervasyon.txt","r+", encoding="utf-8")
    rezervasyonlar=dosya.readlines()
    mecut_rezervasyon=len(rezervasyonlar)
    if rezervasyon_kapasitesi-mecut_rezervasyon >0 :
        print(rezervasyon_kapasitesi-mecut_rezervasyon,"adet daha  rezervasyon
yapabiliriz")
        yeni_rezervasyon=input("rezervasyon bilgilerini araya virgül girerek
giriniz")
        dosya.write("\n"+yeni_rezervasyon )
        print("rezervasyon numaranız =",mecut_rezervasyon+1,end="")
        print("rezervasyon bilgileriniz : ",yeni_rezervasyon)
        print("razervasyonunuz başarıya tamamlanmıştır.")
    else:
        print("rezervasyon kapasitemiz dolmuştur.")
    dosya.close()

print("rezervasyon ekranına hoş geldiniz")
rezervasyon_yap()
```

Örnek 14'te rezervasyon güncellemesi içeren dosya uygulaması bulunmaktadır.

Örnek 14

Rezervasyon güncelleme uygulaması

```
def rezervasyon_güncelle(rezervasyon_no):
    dosya=open("rezervasyon.txt","r+",encoding="utf-8")
    rezervasyonlar=dosya.readlines()
    if rezervasyon_no <len(rezervasyonlar):
        print("rezervason bilgileriniz: ",rezervasyonlar[rezervasyon_no])
        güncelleme=input(" lütfen rezervasyon güncelleme bilgilerini giriniz")
        rezervasyonlar[rezervasyon_no]=güncelleme+"\n"
        print(" güncel rezervason bilgileriniz: ",rezervasyonlar[rezervasyon_no])
        dosya.seek(0)
        dosya.writelines(rezervasyonlar)
        dosya.close()
        print("rezervasonunuz başarıyla güncellenmişitir.")

    else:
        print("hatalı bir rezervason numarası girdiniz")

print("rezervasyon güncelleme ekranına hoş geldiniz")
rezervasyon_no=input("lütfen rezervasyon numaranızı giriniz")
rezervasyon_güncelle (int(rezervasyon_no))
```

Örnek 15'te yapılan rezervasyonun nasıl silineceğini gösteren bir dosya uygulaması bulunmaktadır.

Örnek 15

Rezervasyon silme uygulaması

```
def rezervasyon_sil(rezervasyon_no):
    dosya=open("rezervasyon.txt","r+",encoding="utf-8")
    rezervasyonlar=dosya.readlines()
    if rezervasyon_no <len(rezervasyonlar):
        print("rezervason bilgileriniz: ",rezervasyonlar[rezervasyon_no])
        emin_misiniz=input("kaydı silmek istediğinziden emin misiniz e/h")
        if emin_misiniz=="e" or emin_misiniz=="E":
            rezervasyonlar.pop(rezervasyon_no)
            dosya.seek(0)
            dosya.writelines(rezervasyonlar)
            dosya.close()
            print("rezervasonunuz başarıyla güncellenmişitir.")

        else:
            print("rezervasyon silme işleminiz iptal edilmişitir")

    else:
        print("hatalı bir rezervason numarası girdiniz")

print("rezervasyon silme ekranına hoşgeldiniz")
rezervasyon_no=input("lütfen rezervasyon numaranızı giriniz")
rezervasyon_sil(int(rezervasyon_no))
```

12.6. Bölüm Sonu Örnekleri

1. Elli adet rastgele sayı üretip sayı.txt dosyasına kaydeden program uygulaması
2. Rastgele üretilen sayıları alıp büyükten küçüğe doğru sıralayıp sıralı.txt dosyasına kaydeden program uygulaması

Cevaplar

1.
```
import random
sayılar=[]
for i in range (50):
    sayı=str(random.randint(0,1000))+"\n"
    sayılar.append(sayı)
dosya=open("sayı.txt","w")
dosya.writelines(sayılar)
dosya.close()
```

2.
```
dosya=open("sayı.txt","r")
sayılar=dosya.readlines()
dosya.close()
# sayıları string olarak aldık sayıya dönüştürüp sıralayıp tekrardan stringe
dönüştüreceğiz
for i in range(len(sayılar)):
   sayılar[i]=int (sayılar[i])
   sayılar.sort()
for i in range(len(sayılar)):
   sayılar[i]=str(sayılar[i])+"\n"
dosya=open("sıralı.txt","w")
dosya.writelines(sayılar)
dosya.close()
```

MODÜL 13

HATA YAKALAMA VE İSTİSNALAR

Şekil 13.1: *Bölümle ilgili örnek uygulamalara karekod'dan ulaşabilirsiniz.*

13.1. Hata Kavramı

Kodları yazarken Python, birtakım hata mesajları verir. Bu hata mesajları programda yapılan söz dizimi (syntax) hatalarının görülmesini sağlar. Programcılar her durumu düşünüp hataları önceden belirlemelidir. Hata mesajlarını kullanıcıların görmesi istenmediğinde Python, olası hata oluşumlarına karşı önlemler alınmasını sağlayan ifadeler sunar.

13.2. Hata Yakalama

Beklenmedik durumlarda programın bir hata mesajı vermesi ve çalışmayı durdurması yerine, hataya kullanıcının istediği şekilde cevap vermesini sağlamanın bir yolu olarak adlandırılmaktadır. Hata yakalama Python programlama dilinin önemli bir parçasıdır ve kaynak kodunu çok karışık hâle getirmeden programınızın güvenilir bir şekilde çalışmasını sağlar.

Kullanıcıdan sayılar alan ve aldığı sayıların 2 katını ekrana yazan ve boş satır okuduğunda program sonlandırılmasını sağlayan bir uygulama yapınız.

Örnek 1

```
while True:
x = input("Bir sayı girin: ")
if not x:
break
print(float(x)*2)
Bir sayı girin: 4
8.0
Bir sayı girin: 3
6.0
Bir sayı girin: ali
Traceback (most recent call last):
  File "<ipython-input-89-66782bace4ab>", line 5, in <module>
    print(float(x)*2)
ValueError: could not convert string to float: 'abc'
```

Örnek 1'de girilen "ali" ifadesi sayıya dönüştürülemediği için float() fonksiyonu bir ValueError hatası (Python terimiyle "exception") verdi. Böyle hatalar programın çalışmasını durdurur. Oysa, bir hata yakalama (exception handling) yapısı kullanılır ise bu tür sorunları programı durdurmadan halletmek mümkün olmaktadır.

13.3. Hata Türleri

1. Programcı Hatası (Error)
2. Program Kusurları veya Mantık Hatası (Bug)
3. İstisnalar (Exception)

13.3.1. Programcı Hatası

Geliştiricilerden kaynaklanan bir hatadır. Programcının syntax yazım yanlışı vb. yaptığı hatalardır. Bu hata giderilmeden program hiçbir şekilde çalışmaz. Bu tür hataların hangi satırda ve hangi kodlardan dolayı yapıldığı bellidir. Bu yüzden fark edilmesi ve çözülmesi kolaydır.

Örnek 2

```
print "Hata Ayıklama"
File "<ipython-input-86-230c524677dc>", line 1
   print "Hata Ayıklama"
      ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hata
Ayıklama")?
```

Örnek 2'de print kullanılmış parantez içine alınmadığı için kullanıcıdan kaynaklı SyntaxError hatası vermiştir. Eksik parantez var, diye mesaj verir.

Python'da sıklıkla karşılaşılan hataların bazıları aşağıda verilmiştir:

a) SyntaxError

b) ValueError

c) IndexError

d) ZeroDivisionError

e) NameError

f) IOError

g) TypeError

h) KeyError

Program hatası türünün fark edilmesi zordur. Sebebi ise program çalışır durumdadır herhangi bir hata vermez. Ama istenilen sonuçtan farklı bir sonuç verir. Örnek 3'te kullanıcıdan karenin kenar uzunluğunu alınır ve bu değere göre karenin alanını hesaplanır.

Örnek 3

```
kenar = int(input("Kenarı Girin :"))
alan = 4*kenar
print("karenin alanı :",alan)
Kenarı Girin   :6
karenin alanı : 24
```

Örnek 3'te görüldüğü üzere program hata vermeden çalıştı ama formül hatası olduğu için beklenilen sonuçtan farklı bir hata sonuç verdi.

13.3.2. İstisnalar (Exception)

Program bazı istisnalardan dolayı hata vermektedir. Bu tür hatalarda da program hata düzeltilmeden çalışmaz. Örnek 4'te kullanıcıdan iki sayıyı alıp birinci sayıyı ikinci sayıya bölüp sonucu ekran yazdıran programa bakınız.

Örnek 4

```
s1 = int(input("Birinci Sayı :"))
s2 = int(input("İkinci Sayı :"))
sonuc = s1/s2
print("Sonuc :",sonuc)
Birinci Sayı :32
İkinci Sayı :4
Sonuc : 8.0
Birinci Sayı :32
İkinci Sayı :0
Traceback (most recent call last):

  File "<ipython-input-98-62c9318aa9da>", line 4, in <module>
    sonuc = s1/s2

ZeroDivisionError: division by zero
```

Örnek 4'te birinci sayı sayı için 32 ikinci sayı için 4 değeri girilmiştir. Sonuç olarak 8 değeri yazılmıştır. Uygulamayı tekrar çalıştırdığınızda birinci sayıya 32, ikinci sayıya ise 0 değeri verildiğinde ise herhangi bir sayının sıfıra bölümü tanımsız olduğu için hata mesajı verecektir.

13.4. Try-Except Blokları

Python programlama dili beklenilen veya beklenmeyen hataları ayıklamak için iki ana blok sunmaktadır. Bunlar try ve except bloklarıdır.

13.4.1. Try

Bir blok içerisinde hata olması durumunda, Python except bloğuna atlar ve oradaki kodların çalıştırılmasını sağlar. Hata ile karşılaşılacak yerler blok içine alınan kısım olarak da bilinir.

13.4.2. Except

Try bloğunda herhangi bir hatanın olması durumunda burada yazılmış kodlar çalışır. Try bloğunda herhangi bir hata olmaması durumunda buradaki kodlar çalıştırılmayarak atlanır.

Örnek 4'teki uygulamayı try ve except blokları kullanarak yapmaya çalışınız. Örnek 5'e bakıldığında s1 değerine 5, s2 değerine ise 0 girilmiştir. Bir sayının sıfıra bölümü tanımsız olduğu için except bloğu çalışıp, ekrana Bir sayıyı sıfıra bölemezsiniz sıfır dışında bir sayı girin, diye mesaj yazmıştır.

Örnek 5

```
S1 = int(input("Birinci Sayı :"))
S2 = int(input("İkinci Sayı :"))
 try:
  sonuc = s1/s2
  print("Sonuc :",sonuc)
except ZeroDivisionError:
  print("Bir sayıyı sıfıra bölemezsiniz lütfen sıfır dışında bir sayı girin")
Birinci Sayı :5
İkinci Sayı :0
Bir sayıyı sıfıra bölemezsiniz lütfen sıfır dışında bir sayı girin
```

Örnek 6'da görüldüğü üzere sayıları sıfırdan farklı ve sadece sayı verilirse herhangi bir hata vermeyecek ve sadece try bloğu çalışacaktır. Ama sayılardan herhangi birine sıfır veya farklı veri türünde değer girilmesi hâlinde except bloğu çalışıp bilgilendirme yapacaktır. Bu örnekte de ikinci sayıya "a" girildiği için kullanıcıya lütfen sayısal bir karakter girin, diye mesaj yazacaktır. Except bloğuna hatanın türü yazılmadığında yorumlayıcı tarafından bütün hataları kapsayacaktır. Bunu da sadece except bloğu ile genel olarak beklenmeyen bir hata oluştu, diye mesaj ile belirtilebilir.

Örnek 6

```
try:
   s1 = int(input("Birinci Sayı :"))
   s2 = int(input("İkinci Sayı :"))
   sonuc = s1/s2
   print("Sonuc :",sonuc)
except ZeroDivisionError:
   print("Lütfen ikinci sayıya sıfırdan farklı bir değer girin…")
except ValueError:
   print('Lütfen sayısal bir karakter girin..')
except:
   print("Beklenmeyen bir hata oluştu")
Birinci Sayı :20
İkinci Sayı :a
Lütfen sayısal bir karakter girin.
```

13.4.3. Try Except as

Kullanıcıya Python'a ait hata mesajları gösterilmek istendiğinde "as" ifadesi kullanılmaktadır.

Örnek 7

```
try:
   s1 = int(input("Birinci Sayı :"))
   sonuc = s1**2
   print("Sonuc :",sonuc)
except ValueError as hata:
   print('Lütfen sayı giriniz')
   print(hata)
Birinci Sayı :d
Lütfen sayı giriniz
invalid literal for int() with base 10: 'd'
```

Örnek 7'de sayı yerine harf girilmiştir ve hata mesajı da ekrana yazdırılmıştır.

Try except else, oluşabilecek hataları adım adım ayıklanmak isteniyorsa "else" ifadesi kullanılmaktadır.

Örnek 8

```
try:
  s1 = int(input("Birinci Sayı :"))
  except ValueError:
  print('sayı girmediniz')
else:
  try:
    print(10/s1)
  except ZeroDivisionError:
  print('sayı sıfıra bölünemez')
Birinci Sayı :
sayı girmediniz
```

Örnek 8'de programda herhangi bir sayı girilmemiştir. Sayı girilmediğinde except ValueError bloğu devreye girerek, ekrana "**sayı girmediniz**" diye hata mesajı vermektedir. Ayrıca kullanıcı hata aldığında programın devam etmesi için Örnek 9'daki gibi bir uygulama yapılabilir. Continue ifadesi ile sayı yanlış girilirse tekrar girilmesi sağlanmaktadır. Örnek 9'da ilk önce a harfi sonra sıfır değeri girince hata mesajları verir ve tekrar sayı girilmesi istenir. En sonunda 30 sayısı girildiğinde sonuç ekrana yazdırılır.

Örnek 9

```
while True:
x = input("Bir sayı girin: ")
if not x:
break
                y = 1/float(x)
except ValueError:
print("Geçersiz sayı")
continue
except ZeroDivisionError:
   print("Sıfıra bölme")
   continue
print(y)
Bir sayı girin: a
Geçersiz sayı

Bir sayı girin: 0
Sıfıra bölme

Bir sayı girin: 30
0.03333333333333333

Bir sayı girin:
```

13.5. Bölüm Sonu Örnekleri

1. Kullanıcı tarafından 3 tane sayı girilecek. Bu üç sayı toplanıp ekrana yazdırılacaktır. Hata ayıklama ile programı yapınız.
2. Kullanıcı tarafından 2 sayı girilip ortalaması alınacaktır. Hata ayıklama ile programı yapınız ve hatayı ekrana yazdırınız.
3. Kullanıcıdan ad, soyad, yas bilgileri girilip ekrana yazdırılacaktır. Yanlış girilmesi durumunda hata giderilinceye kadar tekrar veri girilmesi sağlanacaktır. Hata ayıklama ile programı yapınız.

Cevaplar

1.
```
try:
s1 = int(input("Birinci Sayi :"))
s2 = int(input("ikinci Sayi :"))
s3 = int(input("üçüncü Sayi :"))
sonuc = s1+s2+s3
print("Sonuc :",sonuc)
except ZeroDivisionError:
print("lütfen sıfırdan farklı bir sayı giriniz¦")
except ValueError:
print('lütfen sayısal bir değer giriniz')
Birinci Sayi :6
ikinci Sayi :8
üçüncü Sayi :9
Sonuc : 23
```

2.
```
s1 = input("ilk sayi: ")
```

```
s2 = input("ikinci sayi: ")
try:
sayi1=int(s1)
sayi2=int(s2)
sonuc=(sayi1+sayi2)/2
print(sonuc)
except ValueError as hata:
print(hata)
ilk sayi: 7
ikinci sayi: 8
7.5
```

3.

```
while True:
ad= input("adınızı girin:")
soyad= input("soyadınızı girin:")
yas= input("yaşınızı girin:")
if not ad:
break
if not soyad:
break
if not yas:
break
try:
print('adınız:',ad,'soyadınız:','yaşınız:',yas)
except ValueError:
print("veri girişi yapmadınız")
continue
except ZeroDivisionError:
print("Sıfıra bölme")
continue
adınızı girin:ali
soyadınızı girin:yılmaz
yaşınızı girin:40
adınız: ali soyadınız: yaşınız: 40
adınızı girin:
```

MODÜL 14

SQLite VERİ TABANI

Şekil 14.1: *Bölümle ilgili örnek uygulamalara karekod' dan ulaşabilirsiniz*

Veri tabanları verilerin kalıcı olarak depolandığı ve ihtiyaç duyulduğunda okunabildiği gelişmiş yapılardır.

Bilgisayarda bilgiler dosyalara kaydedildiği gibi veri tabanlarına da kaydedilir. Bilginin kalıcı olarak saklanabilmesi, istendiğinde çağrılabilmesi, belirli kriterlere göre seçilebilmesi gibi konular oldukça önemlidir. Veri tabanları bilginin saklanması, istendiğinde ve sadece izin verilen yapılarca çağrılabilmesi açısından oldukça gelişmiş yapılardır. Veriler kaydedilirken ve çağrılırken bu işlerin hızlı ve hatasız yapılabilmesi gerektiğinden veri tabanlarında çalışan özel diller geliştirilmiştir. Python ile kullanılacak veri tabanı yapısında sql isimli sorgu dili kullanılmaktadır. Veritabanları yerel bilgisayarlarda tutulabildiği gibi sunucu veya bulut sistemlerde de tutulabilir. Yerel bilgisayar dışında tutulması durumunda bağlantı yapılırken gereken bilgilerin eklenmesi yeterli olacaktır. Bu bölümde yerel veritabanına bağlantı yapacak uygulamalar yapılmıştır.

Veri tabanı ile çalışma mantığı Şekil 14.2'de gösterildiği gibi beş aşamadan oluşmaktadır.

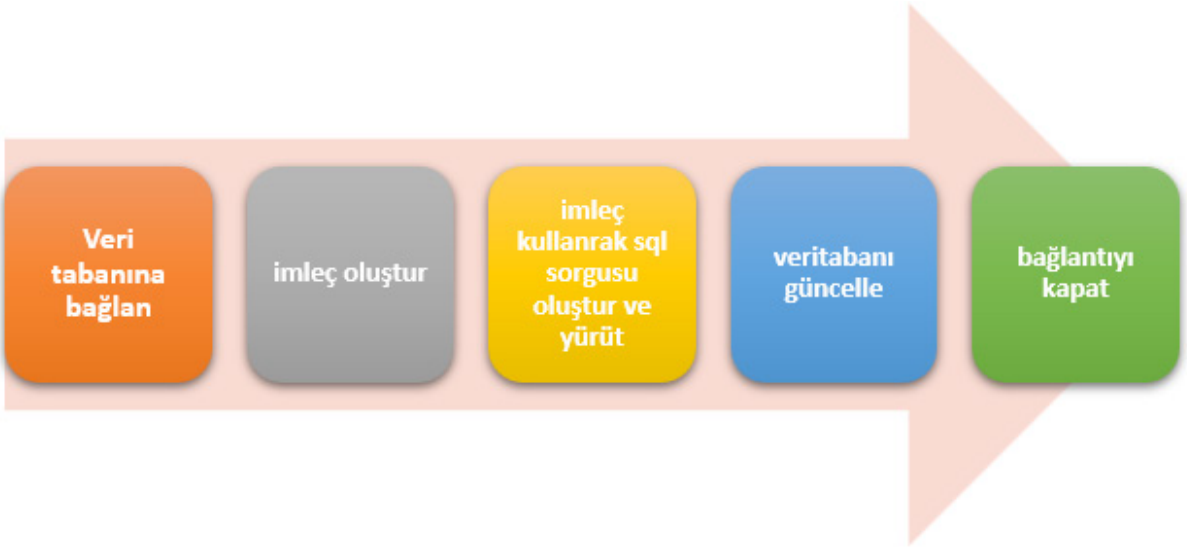
Birinci aşama veritabanı bağlantısının yapılmasıdır.

İkinci aşama sql kodlarının yürütülmesi için imleç nesnesinin oluşturulmasıdır.

Üçüncü aşama imleç nesnesi ile sql kodlarının yazılması ve çalıştırılmasıdır.

Dördüncü aşama veritabanının güncellenmesidir. Bu aşama sadece veri seçme işleminde kullanılmamaktadır.

Beşinci aşama veri tabanı bağlantısının kapatılmasıdır.



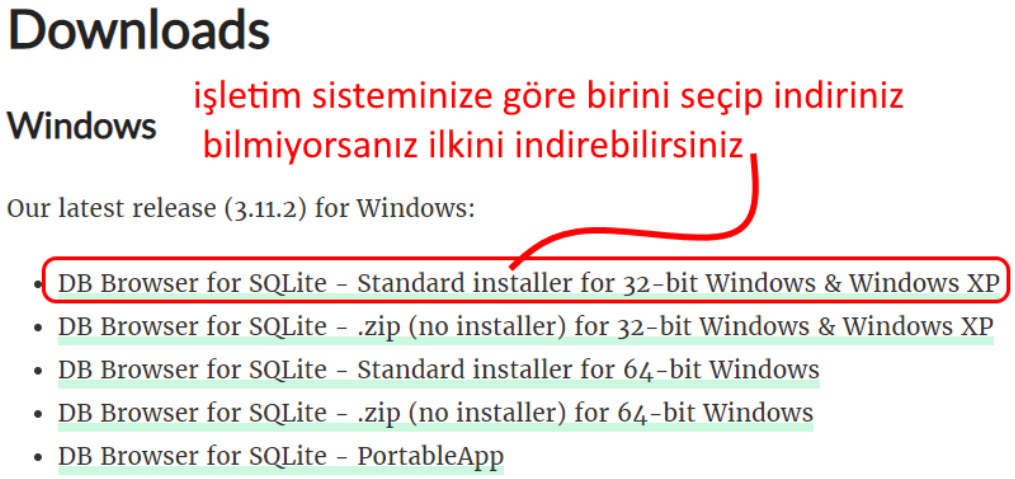
Şekil 14.2: *Veri tabanı ile çalışma süreci*

SQL, veri tabanlarında veri depolamak, işlemek ve almak için standartlaştırılmış bir dildir. Bu dilin öğrenilmesi de ayı bir konu ve uzmanlık alanıdır. Bu nedenle https://www.w3schools.com/sql/ adresinden veya başka Türkçe kaynaklardan sql dilini detaylarıyla öğrenebilirsiniz. Bu bölümde sadece Python ile veri tabanı işlemlerine değinilecektir. Sql içeren kısımlar kısaca anlatılacaktır.

Python ile kullanılabilecek birçok veri tabanı mevcuttur. Bu bölümde yaygın ve kolay olması bakımından sqlite kullanılmıştır. Sqlite'ı Python ile yerel bilgisayarınızda kullanabilmeniz için kurmanız gerekmektedir. Bunun için Şekil 14.3'te gösterildiği gibi https://sqlitebrowser.org/ adresine gidip download düğmesine tıklayınız.

Şekil 14.3: *sqlite web adresi ve indirme bağlantısı*

Ardından işletim sisteminize göre uygun olanı seçip indiriniz. Dilerseniz Şekil-14.4'te gösterildiği gibi ilk seçeneği indirip kurarsanız sorun yaşamadan kullanabilirsiniz.



Şekil 14.4: *sqlite Windows işletim sistemi indirme seçenekleri*

İndirdikten sonra dosyayı çalıştırıp kurulum adımlarını takip ederek uygulamayı bilgisayarınıza yükleyiniz. Artık sqlite ile çalışmaya hazırsınız.

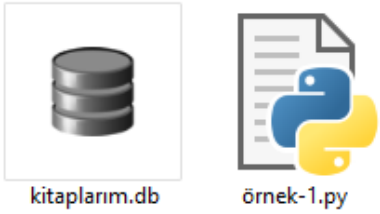
14.1. Sqlite Veritabanı ve Tablo Oluşturma

Python'da Sqlite işlemlerini çalıştırmak için sqlite3 isimli modül kullanılmaktadır. Bu nedenle **import sqlite3** komutunu kullanarak modülü programa dahil ediyoruz. Böylece sql işlemleri için ihtiyaç duyacağımız tüm fonksiyonları kullanabileceğiz. Veri tabanı işlemlerinde öncelikle veri tabanı oluşturulur. Veri tabanı var ise veri tabanına bağlanma işlemine geçilir. **sqlite3.connect(veritabanı adı)** komutu ile veri tabanı yok ise oluşturma ve bağlanma, var ise sadece bağlanma işlemi gerçekleştirilir. Bağlantı işleminden sonra veri tabanı işlemleri için imlec adı verilen bir nesne oluşturulur. İmleç oluşturmak için **sqlite3. cursor()** komutu kullanılır. İlk örnek: kitaplarım isimli veri tabanı uygulamasıdır. Bu uygulamada sadece kitaplarım adında bir veri tabanı oluşturacağız.

Örnek 1

Kitaplık isimli veri tabanı oluşturulması

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
baglantı.close()
```



Şekil 14.5: *Örnek 1 kodları çıktısı*

Kodları kaydettiğimiz klasörde kitaplarım.db dosyasının oluştuğu görülecektir.

14.1.1. Tablo Oluşturma

Veri tabanını oluşturduktan sonra veri tabanına tablo ekleme işlemi yapılacaktır. Sql gibi ilişkisel veri tabanlarında veriler tabloların içinde depolanır. Sql dilinde yeni bir tablo oluşturmak için

CREATE TABLE tablo adı (sütun adı1 veritipi, sütun adı2 veritipi,) yapısı kullanılır. Tabloyu excel tablosu gibi düşünebilirsiniz. Tabloyu oluştururken her bir sütun için isim ve ne tür veri tutulacağını belirtmiş oluyoruz. **CREATE TABLE** dan sonra **IF NOT EXISTS** parametresi de kullanılır. Bu parametre eğer tablo yoksa oluştur anlamına gelir. Eğer bu parametre kullanılmaz ise ve tablo daha önce oluşturulmuş ise **OperationalError: table .... already exists** hatası verir. Bu nedenle **CREATE TABLE IF NOT EXISTS** kullanmak hata ile karşılaşmanın önüne geçer. Bu nedenle tablonun sql yapısı:

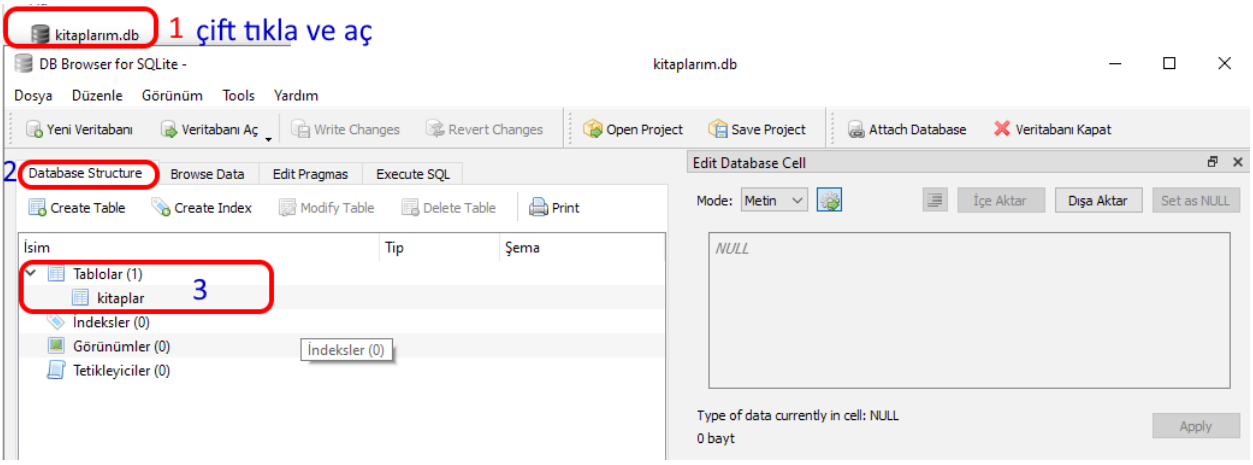
CREATE TABLE IF NOT EXISTS kitaplar (İsim TEXT, Yazar TEXT, Yayınevi TEXT , Sayfa_Sayısı INT) şeklinde olacaktır.

Sql yapısını oluşturduktan sonra bunu imlec nesnesinin execute( ) fonksiyonun içine metinsel ifade olarak ekleyiniz. Son olarak bağlantı nesnesinin commit( ) fonksiyonu ile veritabanı güncellenmektedir. Örnek 2'de kitaplarım veri tabanına tablo ekleme örneği kodları eklenmiştir.

Örnek 2

Veri tabanına tablo ekleme

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
imlec.execute("CREATE TABLE IF NOT EXISTS kitaplar (İsim TEXT, Yazar TEXT, Yayınevi
TEXT, Sayfa_Sayısı INT)")
baglantı.commit()
baglantı.close()
```



Şekil 14.6: Örnek 2 kodları çıktısı

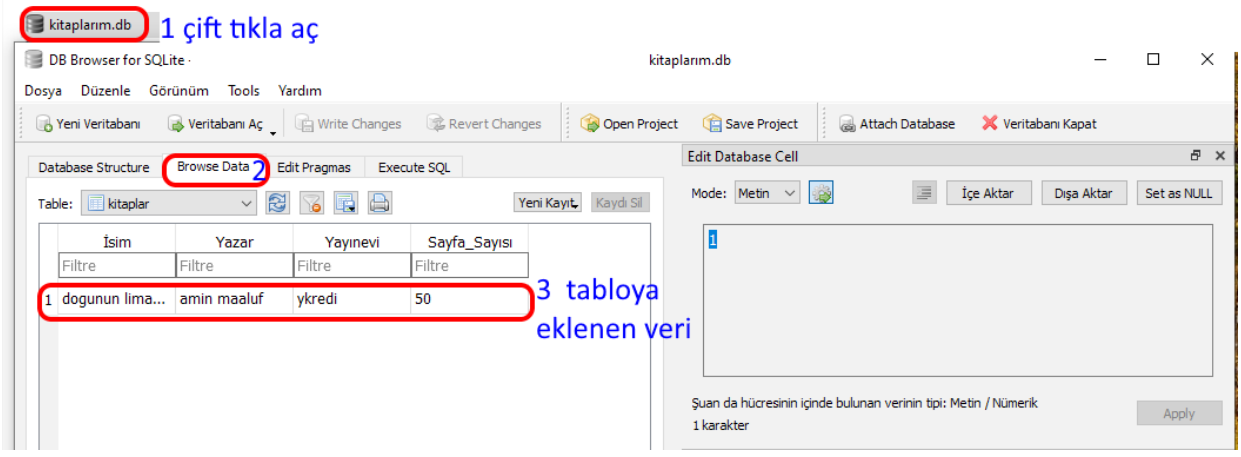
14.2. Veri Ekleme

Sql dilinde tabloya veri eklemek için kullanılan anahtar kelimeler **INSERT INTO tablo_adı (sütun1, sütun 2,) VALUES (değer1, değer2)** veya **insert into tablo_adı (sütun1, sütun 2,) VALUES (değer1, değer2)** şeklinde ifade edilir. Eğer eklenecek verilerin sırası tabloda bulunan sütunlara uyuyor ise **INSERT INTO tablo_adı VALUES (değer1, değer2)** şeklinde de kullanılabilir. Kod yapısı örnek 2- de ki gibidir. Sadece imlec.execute() fonksiyonun içine yazılan sql ifadesi değişmektedir. Ayrıca eklenecek verinin metinsel ifadelerin "tırnak içinde" yazılması gerektiğine dikkat ediniz.

Örnek 3

Tabloya veri ekleme

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
imlec.execute ("INSERT INTO kitaplar VALUES('dogunun limanları','amin maaluf',
'ykredi',50)")
baglantı.commit()
baglantı.close()
```



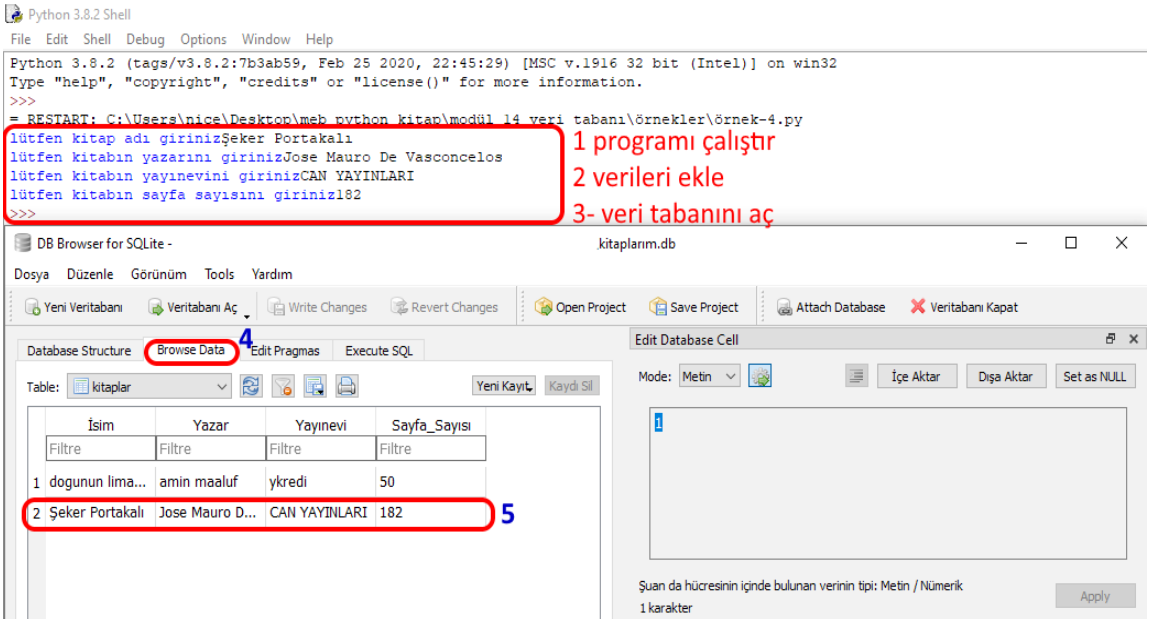
Şekil 14.7: *Örnek 3 kodları çıktısı*

Veri tabanına veri eklendikten sonra kullanıcıdan veri isteyerek tabloya nasıl veri ekleneceği Örnek 4'te gösterilmiştir. Burada dikkat edilecek nokta veriler değişkenlerden alınacağı için direkt olarak values parametresinden sonra kullanılmayacak olmasıdır. "INSERT INTO kitaplar VALUES(?,?,?,?)", (kitap_adı, kitap_yazar, kitap_yayınevi, kitap_sayfa) şeklinde bir yapı kullanılmaktadır. Bu yapı print( ) fonksiyonun kullanımına benzemektedir.

Örnek 4

Kullanıcıdan veri alarak tabloya veri ekleme

```
import sqlite3
def veri_ekle(kitap_adı,kitap_yazar,kitap_yayınevi,kitap_sayfa):
    baglantı=sqlite3.connect("kitaplarım.db")
    imlec=baglantı.cursor()
    imlec.execute ("INSERT INTO kitaplar VALUES(?,?,?,?)",(kitap_adı,kitap_
yazar,kitap_yayınevi,kitap_sayfa))
    baglantı.commit()
    baglantı.close()
kitap_adı=input("lütfen kitap adı giriniz")
kitap_yazar=input("lütfen kitabın yazarını giriniz")
kitap_yayınevi =input("lütfen kitabın yayınevini giriniz")
kitap_sayfa= int(input("lütfen kitabın sayfa sayısını giriniz"))
veri_ekle(kitap_adı,kitap_yazar,kitap_yayınevi,kitap_sayfa)
```



Şekil 14.8: Örnek 4 kodları çıktısı

Sizler programı birkaç defa çalıştırarak farklı kayıtlar oluşturunuz. Böylece sorgu yapacağınız zaman fazla sonuç elde edebilirsiniz.

14.3. Veri Seçme (Çekme)

Sql dilinde veri çekmek (tablodan veri almak) için kullanılan anahtar kelimeler **SELECT sütün1, sütun,2 FROM tablo_adı** şeklindedir. Bu yapıda tabloda belirtilen sütunlar veritabanından çıktı olarak üretilir.

SELECT * FROM tablo_adı Bu yapıda tabloda bulunan tüm sütunlar veritabanından çıktı olarak üretilir.

SELECT * FROM tablo_adı WHERE belirtilen koşullar Bu yapıda tabloda bulunan ve belirtilen koşullara uyan tüm sütunlar veritabanından çıktı olarak üretilir.

SELECT sütun1 FROM tablo_adı WHERE belirtilen koşullar Bu yapıda belirtilen tabloda bulunan, belirtilen koşullara uyan sütun1 verileri veritabanından çıktı olarak üretilir.

Birkaç örnek üzerinden konu pekiştirilecektir. Örnek 5'te kitaplar tablosundaki tüm verileri çekip ekranda yazdıran uygulama bulunmaktadır.

Örnek 5

Tablodan tüm verileri çekmek

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
imlec.execute (" SELECT * FROM  kitaplar")
gelen_veri_listesi=imlec.fetchall()
print(type(gelen_veri_listesi))
print(gelen_veri_listesi)
baglantı.close()
<class 'list'>
[('dogunun limanları', 'amin maaluf', 'ykredi', 50), ('Şeker Portakalı', 'Jose Mauro De Vasconcelos', 'CAN YAYINLARI', 182)]
```

Örnek 5'te görüldüğü gibi veritabanından gelen veriler liste veri tipinde gelmektedir. Gelen veriler üzerinde liste işlemleri yapılabilmektedir.

Bu aşamada belirli sütunları veritabanından çeken uygulama yapılması faydalı olacaktır. Örnek 6'da kitaplar tablosundaki kitap isimlerini veri tabanından çeken uygulama bulunmaktadır:

Örnek 6

Tablodan kitap adlarını çekmek

```
import sqlite3
bağlantı=sqlite3.connect("kitaplarım.db")
imleç=bağlantı.cursor()
imleç.execute (" SELECT İsim FROM  kitaplar")
gelen_veri_listesi=imleç.fetchall()
print(gelen_veri_listesi)
print("kitaplığınızda",len(gelen_veri_listesi),"adet kitap bulunmaktadır.")
bağlantı.close()
[('dogunun limanları',), ('Şeker Portakalı',)]
kitaplığınızda 2 adet kitap bulunmaktadır.
```

Belirli kriterlere uyan verileri tablodan çeken uygulama yapmak konun daha iyi kavranması açısından yararlı olacaktır. Örnek 7'de kitaplar tablosunda bulunan ve yayınevi olarak yapı kredi olan verileri veri tabanından seçen uygulama bulunmaktadır.

Örnek 7

Tablodan belirtilen kritere uygun verileri çekme uygulaması

```
import sqlite3
bağlantı=sqlite3.connect("kitaplarım.db")
imleç=bağlantı.cursor()
imleç.execute (" SELECT * FROM  kitaplar WHERE Yayınevi='ykredi'")
gelen_veri_listesi=imleç.fetchall()
print("sorgunuz ile eşleşen toplam = ",len(gelen_veri_listesi),"kayıt bulunmuştur")
print(gelen_veri_listesi)
bağlantı.close()
sorgunuz ile eşleşen toplam =  1 kayıt bulunmuştur
[('dogunun limanları', 'amin maaluf', 'ykredi', 50)]
```

14.4. Veri Güncelleme

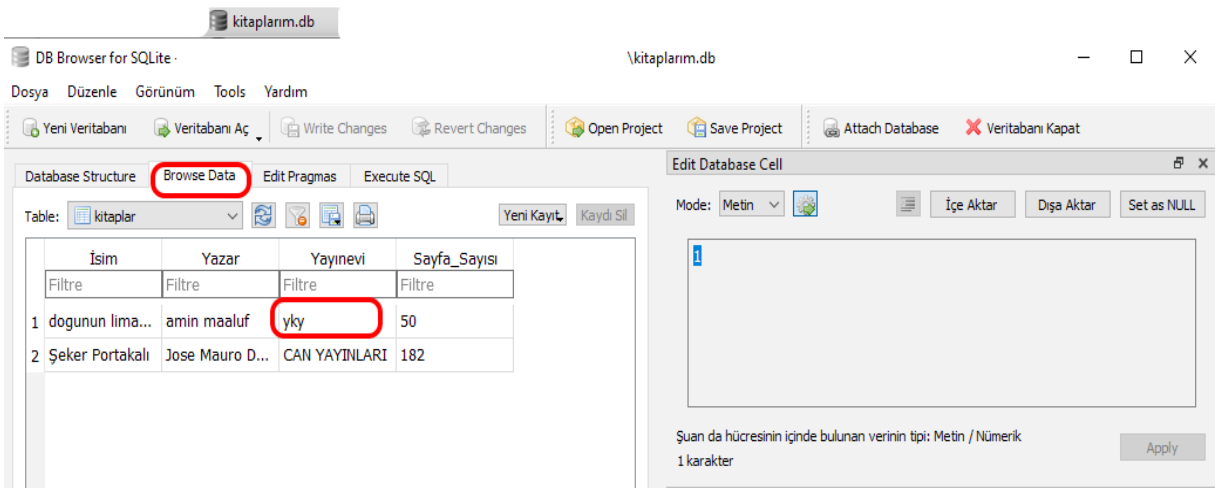
Sql dilinde veri güncelleme işlemi: **"UPDATE tablo_adı SET yeni_değer WHERE değişecek değer"**

Örnek 8'de kitaplar tablosunda yayın evi değerini ykredi'den yky'e değiştiren uygulama bulunmaktadır.

Örnek 8

Tabloda veri güncelleme uygulaması

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
imlec.execute (" UPDATE kitaplar SET Yayınevi= ? WHERE Yayınevi=?",('YKY ', ykredi))
baglantı.commit()
baglantı.close()
```



Şekil 14.9: Örnek 9 kodları çıktısı

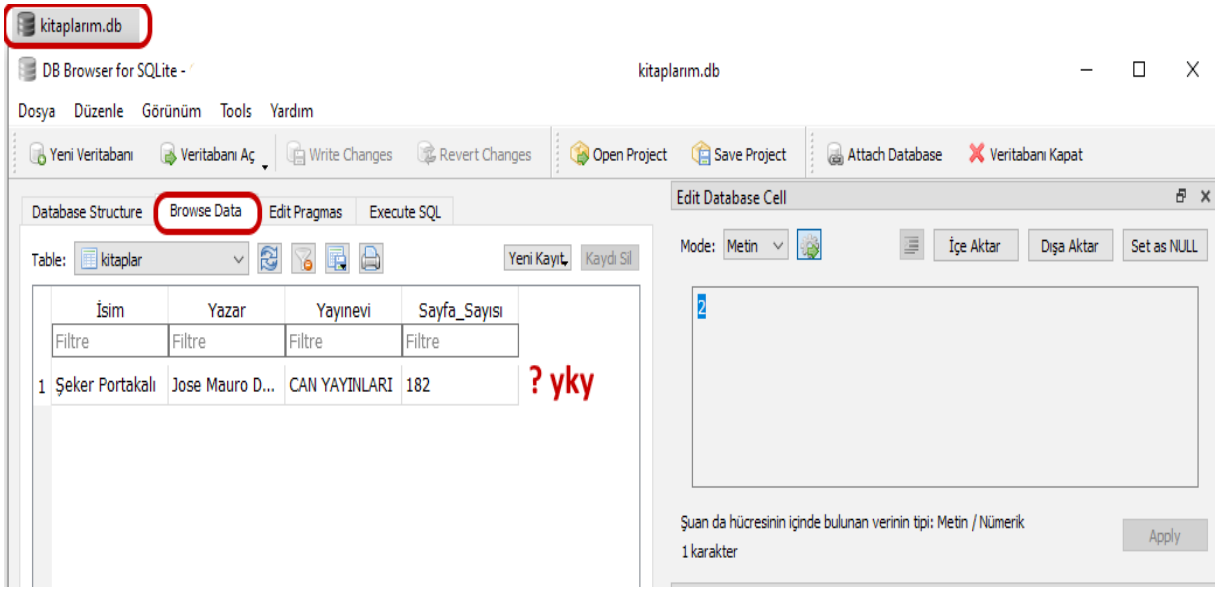
14.5. Veri Silme

Sql dilinde veri silme işlemi: **"DELETE FROM tablo_adı WHERE koşulumuz"** koşuluna uyan kayıtlar veri tabanından silinir. Örnek 9'da yky yayınevine ait kitapları silen uygulama bulunmaktadır.

Örnek 9

Tablodan veri silme uygulaması

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
imlec.execute (" DELETE FROM kitaplar WHERE Yayınevi='yky'")
baglantı.commit()
baglantı.close()
```



Şekil 14.10: Örnek 10 kodları çıktısı

Örnek 10'da kullanıcının gireceği kitap ismine uyan verileri silen uygulama bulunmaktadır.

Örnek 10

Tablodan kullanıcının girdiği kriterlere uyan verileri silme uygulaması

```
import sqlite3
baglantı=sqlite3.connect("kitaplarım.db")
imlec=baglantı.cursor()
kitap_adı=input("silmek istediğiniz kitap adını giriniz")
imlec.execute (" DELETE FROM kitaplar WHERE İsim=? ",(kitap_adı,))
baglantı.commit()
baglantı.close()
```

14.6. Örnek Proje Uygulaması

Veri tabanı işlemleri bölüm bölüm uygulandı. Örnek 11'de proje üzerinde tüm veritabanı işlemlerini içeren uygulama bulunmaktadır. Örnek proje için bir firmanın olduğu varsayılmaktadır. Bu firma için bir veri tabanı oluşturulması ve personel kayıtları için de bir tablo oluşturulması gerekmektedir. Ardından personel tablosuna veri ekleme, seçme, silme ve güncelleme işlemlerinin yapabileceği bir uygulama geliştirilmiştir. İlk aşamada veri tabanı ve tablo oluşturulacaktır. Personel tablosunda kimlik numarası, ad soyad, telefon ve e posta, rolü, çalışılan birim bilgileri kaydedilecektir.

Örnek 11

Firma veri tabanı ve personel tablosu oluşturma

```
import sqlite3
bağlantı=sqlite3.connect("firma.db")
imleç=bağlantı.cursor()
imleç.execute("CREATE  TABLE IF NOT EXISTS personel (kimlik_no INT ,ad_soyad TEXT,
telefon_no TEXT,e_posta TEXT,rolu TEXT,çalışılan_birim TEXT)")
bağlantı.commit()
bağlantı.close()
```

Veri tabanı ve tablo oluşturulduktan sonra veri ekleme, güncelleme, sorgulama ve silme işlemlerinin tek bir program ile yapabileceği Python uygulamasının kodları Örnek 12'de gösterilmiştir.

Örnek 12

Tablo işlemleri uygulaması

```
import sqlite3
baglantı=sqlite3.connect("firma.db")
imlec=baglantı.cursor()
def veri_ekle (veri):
    bilgiler=veri.split(",")
    imlec.execute("INSERT INTO personel
VALUES(?,?,?,?,?,?)",(bilgiler[0],bilgiler[1],bilgiler[2],bilgiler[3],bilgiler[4],
bilgiler[5]))
    baglantı.commit()

def veri_getir (sorgu_metni):
    imlec.execute(sorgu_metni)
    sonuc=imlec.fetchall ()
    print("veri tabanında buluna kayıtlar...")
    for satır in sonuc:
        print(satır)
    print("toplam ",len(sonuc),"adet kayıt listelenmiştir.")

def veri_sil(sorgu_metni):
    imlec.execute(sorgu_metni)
    baglantı.commit()

def veri_güncelle(sorgu_metni):
    imlec.execute(sorgu_metni)
    baglantı.commit()
```

```
while True:
    secim=input('''
|--------------------------------------------|
| yapmak istediğiniz işlemi seçiniz          |
| 1- veri ekleme işlemi                      |
| 2- veri sorgulama işlemi                   |
| 3- veri güncelleme işlemi                  |
| 4- veri silme işlemi                       |
| E- programı sonlandırma işlemi             |
|--------------------------------------------|
''' )
    if secim=="1":
        veri=input('''kimlik numarası, ad soyad, telefon, e posta, rolü,
        çalışılan_birim bilgilerini araya virgül koyarak yazınız''')
        veri_ekle (veri)
    elif secim=="2":
        sorgu=input("lütfen sorgu yapmak için gereken sql metnini giriniz")
        veri_getir (sorgu)
    elif secim=="3":
        sorgu=input("lütfen veri güncelleme yapmak için gereken sql metnini
giriniz")
        veri_güncelle(sorgu)
    elif secim=="4":
        sorgu=input("lütfen sorgu yapmak için gereken sql metnini giriniz")
        veri_sil (sorgu)
    elif secim=="E":
        baglantı.close()
        break
    else:
        print("listede olmayan bir seçim yaptınız")
print("program sonlandırıldı")
```

14.7. Bölüm Sonu Örnekleri

1. Okul adında bir veritabanı oluşturup personel adında tablo oluşturunuz. Örnek tablo alanları ve veri tipleri kimlik_no INT ,ad_soyad TEXT, telefon_no TEXT,e_posta TEXT,rolu TEXT,çalışılan_birim TEXT
2. Okul veri tabanında personel kimlik numarasına göre bilgilerini ekrana getiren programı yazınız.
3. Okul veri tabanında ogrenci adında tablo oluşturunuz.Örnek tablo alanları ve veri tipleri ogrenci_no INT ,ad_soyad TEXT, telefon_no TEXT,e_posta TEXT,sınıfı TEXT,veli_adı TEXT,veli_numarası TEXT
4. Okul veri tabanında öğrenci bilgilerinin tamamını güncelleyebilen programı yazınız.

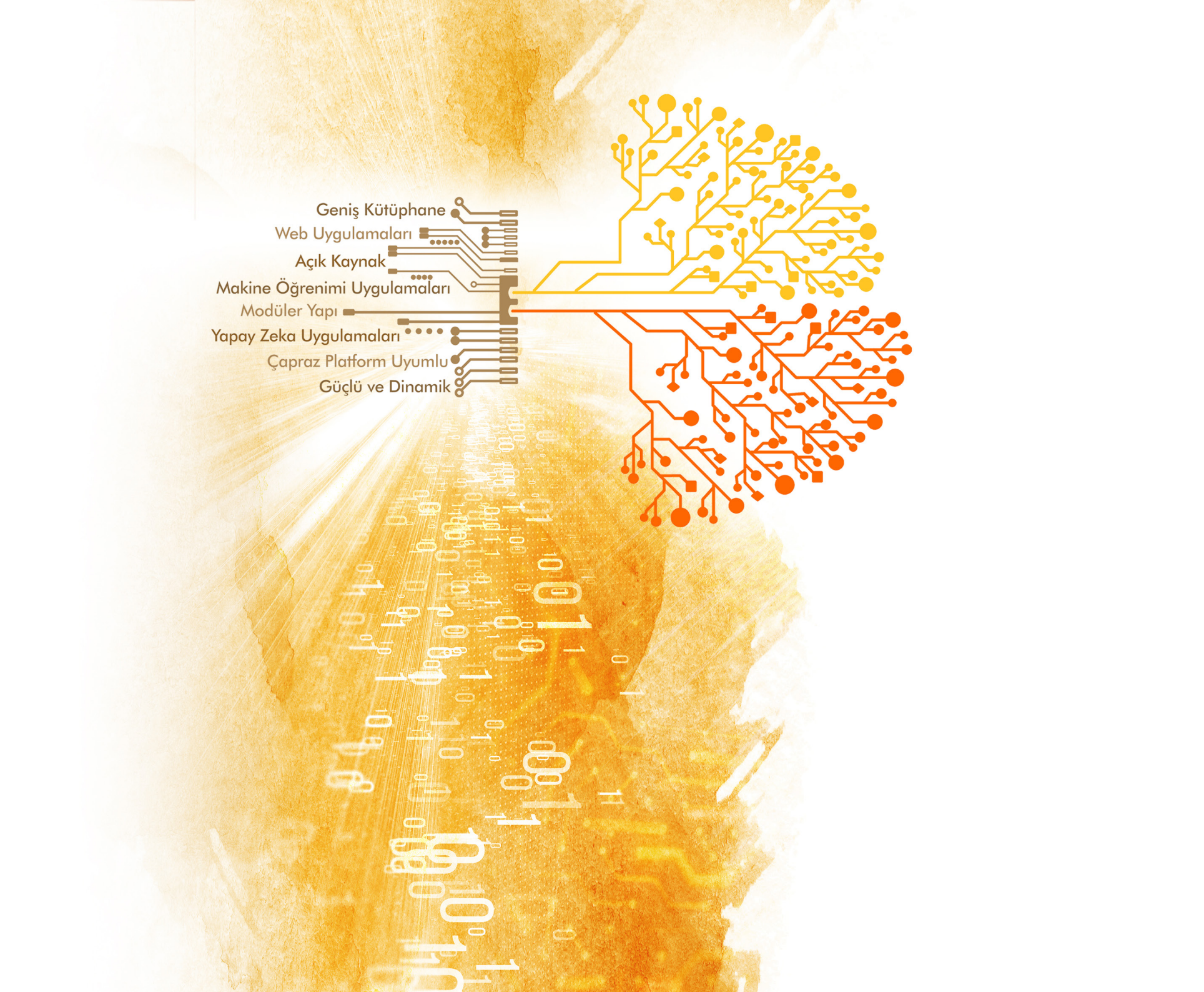
Cevaplar

1.
```
import sqlite3
baglantı=sqlite3.connect("okul.db")
imlec=baglantı.cursor()
imlec.execute("CREATE  TABLE IF NOT EXISTS personel (kimlik_no INT ,ad_soyad
TEXT, telefon_no TEXT,e_posta TEXT,rolu TEXT,çalışılan_birim TEXT)")
baglantı.commit()
baglantı.close()
```

2.
```
import sqlite3
baglantı=sqlite3.connect("okul.db")
imlec=baglantı.cursor()
kimlik_no=input("bilgilerini görmek istediğiniz personelin kimlik numarasını
giriniz")
sorgu="SELECT * FROM personel WHERE kimlik_no=' "+kimlik_no+" '"
imlec.execute(sorgu)
bilgiler=imlec.fetchall()
print(bilgiler)
baglantı.close()
```

3.
```
import sqlite3
baglantı=sqlite3.connect("okul.db")
imlec=baglantı.cursor()
imlec.execute("CREATE  TABLE IF NOT EXISTS ogrenci  (ogrenci_no INT ,ad_soyad TEXT,  telefon_no  TEXT,e_posta  TEXT,sınıfı  TEXT,veli_adı  TEXT,veli_numarası TEXT)")
baglantı.commit()
baglantı.close()
```

4.
```
import sqlite3
baglantı=sqlite3.connect("okul.db")
imlec=baglantı.cursor()
ogrenci_no=input("bilgilerini görmek istediğiniz öğrencinin numarasını giriniz")
sorgu="SELECT * FROM ogrenci WHERE ogrenci_no=' "+ogrenci_no+" '"
imlec.execute  (sorgu)
bilgiler=imlec.fetchall()
print(bilgiler)
yeni_bilgiler=input('''
lütfen yeni bilgileri araya virgül koyarak giriniz
ogrenci_no, ad soyad, telefon no, e posta, sınıfı,veli adı, veli telefon numarası ''')
liste=yeni_bilgiler.split(",")
sorgu="UPDATE ogrenci SET ogrenci_no =?,ad_soyad=?,telefon_no=?,e_posta=?,sınıfı=?,veli_adı=?,veli_numarası=? WHERE ogrenci_no=? "
imlec.execute(sorgu,(liste[0],liste[1],liste[2],liste[3],liste[4],liste[5],liste[6],ogrenci_no))
baglantı.commit()
baglantı.close()
```


Geniş Kütüphane
Web Uygulamaları
Açık Kaynak
Makine Öğrenimi Uygulamaları
Modüler Yapı
Yapay Zeka Uygulamaları
Çapraz Platform Uyumlu
Güçlü ve Dinamik

MODÜL 15

İLERİ SEVİYE FONKSİYONLAR

Şekil 15.1: Bölümle ilgili örnek uygulamalara karekod' dan ulaşabilirsiniz

Python, kodların yazılması ve okunması yönünden kolay bir programlama dilidir. Dilin en belirgin özelliklerinden biri de tek bir kod satırında güçlü ve işlevsel özellikler oluşturabilmemize olanak vermesidir. Bu özelliklere sahip olan ileri seviye fonksiyonlardan **List Comprehension, lambda, map, filter ve reduce** fonksiyonları işlenecektir. Bu fonksiyonlar python'ca (phytonic) yapılma yöntemlerini ifade ederler. Birkaç satırda yapılacak işlemi tek satırda yapabilme kolaylığı ve zevkini sunarlar.

Özyinelemeli fonksiyonlar ise birçok programlama dilinde mevcut olan bir yapıdır. Bu nedenle bu bölümde özyinelemeli fonksiyonların python ile nasıl yazıldığı anlatılmıştır.

15.1. List Comprehension

List Comprenshin, bir çırpıda liste oluşturma manasına gelmektedir. Pek çok programcı bu tarz özellikleri kullanarak kolaylıkla listeler oluşturmaktadır.

Python'da listeler, farklı veri türlerini içerisinde barındıran bir veri tipidir. Bu özelliği ile diğer pek çok dilden farklıdır. Listeleri oluşturmak için tanımlama işlemi yaparken birkaç yöntem bulunmaktadır.

```
>>> liste=[1,2,3]
>>> liste=[]
>>> liste=list()
```

Yukarıdaki yöntemlerin üçü kullanılarak liste oluşturulabilir. Bunun yanında liste oluştururken döngülerden de faydalanabiliriz. Sayı dizilerinden listeler oluşturmak isteyeceğimiz gibi, bir listeden farklı bir liste türetmek için de Python'da for döngüsünden faydalanabiliriz.

Örnek 1

for döngüsü kullanarak liste oluşturma

```
liste1=[1,2,3,4,5]
liste2=[]
for i in liste1:
    liste2.append(i)
print(liste2)
 [1, 2, 3, 4, 5]
```

Bu işlem ile listede çoğaltma işlemi yapıldı. Aynı işlem List Comprehension metodu sayesinde daha az kodla yapabilir.

Örnek 2

list comprehension metodu içinde for döngüsü kullanarak liste oluşturma

```
liste1=[1,2,3,4,5]
liste2=[i for i in liste1]
print(liste2)
[1, 2, 3, 4, 5]
```

Liste kolaylıkla çoğaltılmış oldu.

Bu işlem ile mevcut liste üzerinde işlem de yapılabilir. Örnek 3'te liste elemanlarının karesini alan list comprehension uygulaması bulunmaktadır.

Örnek 3

list comprehension metodu içinde for döngüsü kullanarak liste elemanlarının karesini alma.

```
liste1=[3,4,5,6,7]
liste2=[i**2 for i in liste1]
print(liste2)
[9, 16, 25, 36, 49]
```

list comprehension metodu ile sıfırdan belirli bir sayıya kadar da liste oluşturulabilir. Örnek 4'te 1'den 10'a kadar sayılardan oluşan bir liste oluşturma uygulması bulunmaktadır.

Örnek 4

list comprehension metodu ile 1'den 10'a kadar sayılardan oluşan bir liste oluşturma.

```
liste2=[i for i in range(10)]
print(liste2)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Aynı işlemler string'ler üzerinde de uygulanabilir.

Örnek 5

list comprehension metodu ile Stringler üzerinde gezinme işlemleri

```
yazi="python"
liste=[i*2 for i in yazi]
print(liste)
['pp', 'yy', 'tt', 'hh', 'oo', 'nn']
```

Örnek 6

list comprehension metodu ile İç içe listeleri birleştirerek tek bir liste üzerinde birleştirme.

```
liste1=[[1,2,3],[4,5,6],[7,8,9]]
liste2=[j for i in liste1 for j in i]
print(liste2)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Örnek 7

list comprehension metodu ile çift Sayıları bulan uygulama

```
liste=[i for i in range(1,20) if i%2==0]
print(liste)
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

15.2. Özyinelemeli Fonksiyonlar

Özyinelemeli (recursive) fonksiyonlar kendilerini tekrar kullanan /çağıran fonksiyonlardır. Bu fonksiyonlar, zor ve uzun problemleri daha kolay parçalara ayırarak adım adım çözme mantığına dayanırlar. Bilgisayar bilimlerinde böl ve yönet (Divide and Conquere) metodunun uygulamalarından biridir. Problem çözmede farklı bir yaklaşım olarak ta görülebilir. Örneğin faktöriyel hesaplama işlemi nasıl yapılır? Akla ilk gelen yöntem birden sayıya kadar veya tersi sayıların çarpımını kullanmaktır.

Örnek 8

Faktöriyel hesaplama

```
faktoriyel=1
sayı=int(input("faktöriyelini hesaplamak istediğiniz sayıyı giriniz"))
if sayı>=0:
    for i in range(1,sayı+1):
        faktoriyel*=i
print (faktoriyel)
faktoriyelini hesaplamak istediğiniz sayıyı giriniz6
720
```

Verilen problemin çözümü fonksiyon olarak yazılmak istenirse:

Örnek 9

Faktöriyel hesaplayan fonksiyon

```
def faktoriyel_hesapla(sayı):
    faktoriyel=1
    if sayı>=0:
        for i in range(1,sayı+1):
            faktoriyel*=i
    return faktoriyel
sayı=int(input("faktoriyelini hesaplamak istediğiniz sayıyı giriniz"))
print (faktoriyel_hesapla(sayı))
faktoriyelini hesaplamak istediğiniz sayıyı giriniz5
120
```

Aynı probleme özyinelemeli olarak yaklaşılırsa nasıl olacağı incelenmiştir. Özyineleme kullanmaktaki amaç problemi daha küçük parçalara ayırarak çözmektir. n!'in aslında n! = n*(n-1) faktöriyel olarak görülürse ve en temel parçaya kadar bu yaklaşım benimsenir ise problem çözülmüş olur. Örnekte en küçük birim 1! (bir faktöriyel) olacağından tüm işlem bir faktöriyele ulaşana kadar her sayının bir alt sayı faktöriyeli ile çarpılması olacaktır.

Örnek 10

Özyineleme metodu ile faktöriyel hesaplama

```
def faktoriyel_hesapla(sayı):
    faktoriyel=1
    if sayı==1:
        return 1
    else:
        return sayı*faktoriyel_hesapla(sayı-1)

sayı=int(input("faktoriyelini hesaplamak istediğiniz sayıyı giriniz"))
print (faktoriyel_hesapla(sayı))
faktoriyelini hesaplamak istediğiniz sayıyı giriniz7
5040
```

Özyinelemeye başka bir örnek ise fibonacci terim sayısının hesaplamasıdır.

Örnek 11

Özyineleme metodu ile Fibonacci terim sayısını hesaplama

```
def fibonacci(n):
    if n <= 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

sayı=int(input("hesaplanmasını istediğiniz fibonacci terim sayısını giriniz"))
print(fibonacci(sayı))
hesaplanmasını istediğiniz fibonachi terim sayısını giriniz8
21
```

Birden girilen sayıya kadar olan sayıların toplamını hesaplayan programın özyineleme kullanılarak nasıl yapılabileceğini inceleyelim. Örnek 12'de for döngüsü ile yapılmıştır. Örnek 13'te de özyinelemeli olarak yapılmıştır.

Örnek 12

for döngüsü ile toplam hesaplama

```
sayı=int(input("kendisine kadar olan sayıların toplamını hesaplamak istediğiniz
sayıyı giriniz"))
toplam=0
for i in range(1,sayı+1):
    toplam+=i
print(toplam)
kendisine kadar olan sayıların toplamını hesaplamak istediğiniz sayıyı giriniz20
210
```

Örnek 13

Özyineleme yöntemi ile toplam hesaplama

```
def topla(sayı):
    if sayı==1:
        return 1
    else:
        return sayı+topla(sayı-1)
sayı=int(input("kendisine kadar olan sayıların toplamını hesaplamak istediğiniz
sayıyı giriniz"))
print(topla(sayı))
kendisine kadar olan sayıların toplamını hesaplamak istediğiniz sayıyı giriniz25
325
```

15.3. Lambda Fonksiyonları

Lambda fonksiyonları küçük anonim fonksiyonlar olarak isimlendirilirler. Lambda fonksiyonlarını işlevsel tek satırlık fonksiyonlar olarak düşünebilirsiniz. Normal bir fonksiyon tanımlarken def anahtar kelimesi kullanılarak fonksiyon tanımlanır ve return anahtar kelimesi ile sonuç döndürülür. Lambda yöntemi ile bu ikisinin de kullanılma ihtiyacı ortadan kalkar. Böylece lambda kullanarak programlamada işimizi görecek pratik fonksiyonlar kullanabiliriz.

Argümanlar giriş ifadeler çıkış değerleridir. Kullanım şekli:

Lambda arguman : ifadeler şeklindedir.

Örnek 14, 15 ve 16'da lambda fonksiyonu kullanım örnekleri verilmiştir.

Örnek 14

Lambda fonksiyonu kullanımı örneği:

```
x=lambda a:a+15
print(x(5))
20
```

Örnek 15

Lambda fonksiyonu ile hipotenüs hesaplama

```
import math
hipotenus=lambda x,y:math.sqrt(x*x+y*y)
print(hipotenus(3,4))
5
```

Örnek 16

Lambdanın direkt kullanım şekli

```
print((lambda x,y:3*x+5*y)(2,5))
31
```

15.4. Map Fonksiyonu

Map fonksiyonu bir işlemin veya fonksiyonun birden çok veriye tek satırda uygulanmasını sağlayan pratik fonksiyonlardan biridir. Map fonksiyonu for döngüsü kullanarak yapılabilecek işlemlerin tek satırda çözümlesine olanak tanır. Map fonksiyonun kullanımı şu şekildedir:

map(fonksiyon,iterasyon yapılabilecek veri tipi(liste,demet vb),)

davetliler listesi ile ilgili standart bir davet mesajı uygulaması üzerinden map fonksiyonunu anlamaya çalışalım. Sabit metne listedeki isimleri ekleyerek kişiye özgü davetiye metni oluşturma uygulaması yapılacaktır. İşlemin daha iyi anlaşılması için Örnek 17'de for döngüsü kullanarak yapılmıştır. Ardından Örnek 18'de ise map fonksiyonu kullanarak yapılmıştır.

Örnek 17

for döngüsü ile kişiye özel davetiye metni oluşturma.

```
def davetiye_metni (isim):
    gonderilecek_metin ="Sayın "+isim+"\n"+" Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız."
    return gonderilecek_metin

isimler=["Ali","Ayşe","Murat","Elif"]
for birey in isimler:
    print(davetiye_metni (birey))
```

```
Sayın Ali
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
Sayın Ayşe
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
Sayın Murat
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
Sayın Elif
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
```

Örnek 18

map fonksiyonu ile kişiye özel davetiye metni oluşturma

```
def davetiye_metni (isim):
    gonderilecek_metin ="Sayın "+isim+"\n"+" Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız. \n"
    return gonderilecek_metin

isimler=["Ali","Ayşe","Murat","Elif"]
davetiye=map(davetiye_metni,isimler)
print (davetiye)#davetiye değişkeninin bellekteki konumunu yazdırır
print (*davetiye)#davetiye değişkeninin değerini yazdırır
<map object at 0x03468E50>
Sayın Ali
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Ayşe
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Murat
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Elif
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
```

Map fonksiyonu sonucunda üretilen değer değişkene atandığı zaman değişkenin değeri erkanda görüntülenmek istendiğinde Örnek 18'de görüldüğü gibi değişkeninin bellekte tutulduğu adres gösterilmektedir. Davetiye değişkeninin değeri görüntülenmek istendiğinde değişkenin başına * işaretinin eklenmesi yeterli olacaktır. Benzer yöntem ilerleyen örneklerde de kullanılacaktır.

map fonksiyonu lamba fonksiyonu ile de kullanılır. Böylece lambda fonksiyonu liste gibi çok türde veri içeren yapılara uygulanmış olur.

Örnek 19

map ve lambda fonksiyonlarının birlikte kullanımı

```
isimler=["Ali","Ayşe","Murat","Elif"]
yeni_liste =map(lambda x:"Sayın "+x+"\n Bu mutlu günümüzde sizleri de aramızda
görmekten mutluluk duyarız.\n", isimler)
print(*yeni_liste)
Sayın Ali
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Ayşe
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Murat
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız.
 Sayın Elif
 Bu mutlu günümüzde sizleri de aramızda görmekten mutluluk duyarız
```

15.5. Reduce Fonksiyonu

reduce( ) fonksiyonu değer olarak aldığı fonksiyonu soldan başlayarak listenin ilk 2 elemanına uygular. Daha sonra çıkan sonucu listenin 3. elemanına uygular. Bu şekilde devam ederek liste bitince bir tane değer döner. Reduce fonksiyonu functools modülünde olduğundan bu modülü dâhil ediyoruz

Reduce fonksiyonunun daha iyi anlaşılması için Örnek 20'yi inceleyelebilirsiniz.

Örnek 20

reduce örneği

```
from functools import reduce
print(reduce(lambda x,y : x + y , [1,2,3,4]))
10
```

Örnek 20 incelediğinde lambda fonksiyonu ile 2 sayının toplandığı ardından üretilen sonuç ile listede bir sonraki sayının aynı toplama işlemine dahil edilerek bir sonuç üretildi görülmektedir.

Örnek 20'de list comprehension, lambda ve map fonksiyonlarının birlikte kullanılarak faktöriyel hesaplamasının yapıldığı uygulama bulunmaktadır.

Örnek 21

reduce ile faktöriyel hesaplama

```
import functools
sayı=int(input("faktöriyelini hesaplamak istediğiniz sayıyı giriniz"))
liste=[i for i in range(1,sayı+1)]
print (functools.reduce(lambda x,y :x*y,liste))
faktöriyelini hesaplamak istediğiniz sayıyı giriniz 5
120
```

15.6. Filter Fonksiyonu

filter( ) fonksiyonu liste veri tipindeki yapılar için kullanılır. Kullanım şekli:

filter(fonksiyon1, liste1)

liste1 liste veri tipinde değerler,

fonksiyon1 filter fonksiyonun içinde kullanılan önceden tanımlanmış fonksiyonu ifade etmektedir.

İçinde kullanılan fonksiyonun döndürdüğü değerin doğru (True) olduğu durumlara göre liste veri tipinde çıktı üretir. Diğer bir değişle verilen liste üzerinde fonksiyon işlemlerinin çıktısı kullanılarak filtreleme işlemi yapılır. Örnek 22'de filter fonksiyonu örnek uygulaması bulunmaktadır.

Örnek 22

filter fonksiyonu örnek kullanımı

```
def buyuk_harf(isim):
return isim.isupper( )
liste=["Ali","ali","Selim","SELİM","selim"]
sonuc=filter(buyuk_harf,liste)
print(*sonuc)
SELİM
```

filter fonksiyonu lambda fonksiyonu ile de kullanılabilir. Örnek 23'te filter ver reduce fonksiyonlarının birlikte kullanılarak verilen sayılardan sadece üçe tam bölünebilen sayıların ekrana yazdırıldığı uygulama gösterilmiştir.

Örnek 23

Üçe bölünebilen sayıları ekrana yazdırma

```
sayılar = [10, 15, 4, 29, 402, 249, 210, 55, 40, ]
sonuc = filter(lambda x: (x % 3 == 0), sayılar)
print(*sonuc)
15 402 249 210
```

Örnek 24'te reduce ve lambdanın birlikte kullanımına başka bir örnek gösterilmiştir. Örnek bir bankaya ait müşteri ve mevduat listesinden bakiyeleri 150'den büyük olanların seçilip ekrana yazdırılmasıdır.

Örnek 24

Bakiyesi 150'den büyük olan müşteri seçme uygulaması

```
#müşteriler listesi müşteri no ile bakiye miktarı
musteriler= [[1, 12], [2, 600], [ 3, 500], [4,150]]
sonuc = filter(lambda x: (x[1] > 149),musteriler )
print(*sonuc)
[2, 600] [3, 500] [4, 150]
```

15.7. Bölüm Sonu Örnekleri

1. 5'ten başlayıp 20'ye kadar olan çift sayılardan oluşan bir listeyi list comprehension kullanarak yapınız.

2. liste1=["ahmet","ayşe","metin"] listesinden list comprehension metodu ile verilerin başına sayın ifadesi gelecek şekilde (['Sayın ahmet', 'Sayın ayşe', 'Sayın metin']) liste2 adında bir liste oluşturunuz.

3. Elinizde bir teste ait cevap anahtarı ve öğrenci cevaplarının olduğu listeler var. siz bu cevapları anahtar ile kontrol ederek sonuçları doğru veya yanlış olarak çıktı almak istiyorsunuz. Bunu yapacak kodları lambda ve map fonksiyonları ile nasıl yaparsınız.

Örnek: cevap_anahtarı=["a","c","e","b","a"] ogrenci_1=["a","d","d","b","a"]

İstenen çıktı Çıktı: ['doğru', 'yanlıs', 'yanlıs', 'doğru', 'doğru']

Cevaplar

1.
```
liste=[ i  for i in range(5,21,2)]
print(liste)
```

2.
```
liste1=["ahmet","ayşe","metin"]
liste2=["Sayın "+i for i in liste1]
print(liste2)
```

3.
```
cevap_anahtari=["a","c","e","b","a"]
ogrenci_1=["a","d","d","b","a"]
liste2=list(map(lambda x,y: "doğru" if x==y   else "yanlıs",cevap_
anahtari,ogrenci_1))
print(liste2)
```

MODÜL 16

NESNE TABANLI PROGRAMLAMA

Şekil 16.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

16.1. Nesne Tabanlı Programlama (NTP) Nedir?

Nesne Tabanlı Programlama bir programlama paradigmasıdır. Bu yaklaşımda programlama mantığı gerçek hayattaki nesneler gibi tasarlanmıştır. Gerçek hayatta bir nesneye ihtiyacınız olduğunda onu alır ve kullanırsınız. Her nesnenin kendine ait özellikleri ve işlevleri vardır. Odanızdaki masa lambanız bozuldu ve yeni bir masa lambası almanız gerektiğini varsayınız. İçinde birçok aydınlatma cihazının olduğu büyük bir markete gittiniz. Raflarda çeşit değişik özelliklerde çeşit çeşit masa lambaları var. Tüm ürünlerin kullanılan malzeme, ışığın rengi, aydınlatma gücü, elektrik tüketimi, garanti süresi ve fiyatı gibi farklı özellikleri bulunmaktadır. Aynı zamanda bu cihazların açma, kapama, ışık ayarı gibi bazı işlevleri de bulunmaktadır. Nesneler (canlı veya cansız) tanımlanırken onların özelliklerinden ve işlevlerinden yararlanılır. Lambaların özellikleri farklı olsa da işlevleri değişse de hepsi ortak özellikleri olan bir

masa lambası formu (çizelge) ile tanımlanabilir. Bu formda her masa lambasına ait özellikler (olmayan bölümler için boş bırakılmış) ve bu cihazların işlevleri yer almaktadır. Hazırlanan genel form her masa lambasını temsil edebilecek özelliklerin ve işlevlerin bir listesini barındırmaktadır.

16.2. Sınıflar ve Nesneler

Masa lambası örneğinde olduğu gibi bir nesne ile ilgili değişik türde veriyi (formdaki veri: sayı, metin, liste) ihtiyaç duyulduğunda bunları yönetmek için nasıl bir yapı gereklidir? Bunun için bir liste, demet veya sözlük kullanıldığında bazı sıkıntılar yaşanabilir. Masa lambasının herhangi bir özelliğine bir değer atamak istediğinizi ya da bu özelliğe ilişkin veriyi almak istediğinizi düşününüz. Lamba ile ilgili değişik veri türlerini yönetebileceğiniz bir yapıya ihtiyaç vardır. Lambanın özelliklerini ve işlevlerini yerine getirecek değişkenleri ve fonksiyonları tanımlamak gerekir. Eğer bu ihtiyaçlar NTP yaklaşımı ile ele alınmazsa tüm bu özellikler ve işlevler için farklı ve dağınık yapılar oluşacaktır. NTP, bir nesne olarak yapının tüm özelliklerinin ve işlevlerinin (metot) bir yerde toplanmasına ve kullanılmasına olanak verir. Bir sınıf tanımlandığı zaman aslında bir model oluşturulur. Bu model o sınıfa ait tüm nesneleri temsil eden ve aslında hiçbiri olmayan genel bir model, bir şablondur. Bu modeli kullanarak o sınıftaki tüm nesneler türetilebilir. Bir sınıftan bir nesne türetildiğinde yapıya ilişkin modelin bir kopyası oluşur. Nesne ile sınıfın özellikleri ve işlevleri kullanılıp özelleştirilebilir.

NTP'de oluşturulan yapılar nesneler tarafından temsil edilir. Bu nesneler kullanıcı tarafından veya başkaları tarafından oluşturulan diğer nesnelerle etkileşime girebilirler. NTP gerçek dünyada nesneler arasındaki etkileşim gibi programlama yapılarındaki etkileşimleri de modellemeye çalışır. Program akışına göre nesneler diğer nesnelerle etkileşime girerek özelliklerini ve işlevlerini o nesnelerin erişimine açabilir. Başka bir nesne, eriştiği nesneden aldığı değeri kullanarak yeni işlemler yapabilir. Bu sonuç bir argüman (girdi) olarak kullanılıp nesnenin kendi işlevlerinden biri çalıştırılabilir. Böylelikle gereksinim duyulan bir program için programcının kendi oluşturduğu veya önceden oluşturulmuş nesneler uygun şekilde bir arada kullanılabilir. Python'da tüm yapılar sınıflar içinde oluşturulur.

Python'da kullanılan veri türlerinin her biri bir sınıftır.

Örnek 1

type( ) komutunu kullanarak veri türlerini görüntüleyerek bunu görebilirsiniz.

```
print(type(int))
print(type(str))
print(type(list))
print(type(tuple))
print(type(dict))
<class 'type'>
<class 'type'>
<class 'type'>
<class 'type'>
<class 'type'>
```

16.2.1. Bir Sınıf Tanımlama

Örnek 2

Basit bir sınıf tanımlayınız. Bir sınıf class SinifAdi: veya class SinifAdi(): şeklinde tanımlanır.

Blok yapısı içinde yer alır.

Özellikler ve metotlar sınıfın içinde tanımlanır.

```
class OrnekSinif:
  __doc__='Bu sınıf örnek amaçlı oluşturulmuştur.'
  def __init__(self):
    print ('Merhaba Sınıf')
```

Sınıfı tanımlarken class OrnekSinif(): olarak yapabilirsiniz. Sınıftan bir nesne oluşturabilirsiniz.

16.2.2. Sınıftan Bir Nesne Oluşturma

Örnek 3

Sınıflardan nesne türetme (instance) örnek oluşturma olarak adlandırılır.

```
nesnem=OrnekSinif()
print (nesnem)
Merhaba Sınıf
<__main__.OrnekSinif object at 0x7fcbe181acc0>
```

Nesnenizi oluşturdunuz. print( ) kullandığınızda nesnenin ait olduğu sınıfın adı ve hafızadaki yeri yazdırılacaktır. Sınıfın açıklamasına bakmak için "nesnem.__doc__"u kullanabilirsiniz.

Örnek 4

```
nesnem.__doc__
'Bu sınıf örnek amaçlı oluşturulmuştur.'
```

16.2.3. Yapıcı "__init__()" Fonksiyonu

Bir sınıftan bir nesne oluşturulduğunda otomatik olarak çalışan bir fonksiyondur. NTP'de yapıcı (constructor) fonksiyon olarak adlandırılan bu fonksiyonlar nesne için zorunlu parametreleri işler ve nesnenin oluşturulmasını sağlar.

Örnek 5

Yapıcı fonksiyonu self olmadan tanımlayınız.

```
class OrnekSinif:
  def __init__():
    print ('Merhaba Sınıf')
nesnem=OrnekSinif()
TypeError                                 Traceback (most recent call last)
<ipython-input-51-dca29725c758> in <module>()
      2   def __init__():
      3     print ('Merhaba Sınıf')
----> 4 nesnem=OrnekSinif()

TypeError: __init__() takes 0 positional arguments but 1 was given
```

Kodu "self" olmadan kullandığınızda yukarıdaki hata ile karşılaşırsınız.

Örnek 6

Verilen metin ifadelerini istenilen kadar yazan bir sınıf tanımlayınız.

Sınıf için bazı zorunlu parametreler oluşturulacaktır.

```
class MetinSinifi:
  def __init__(self, ifade, tekrarSayisi, kacisKarakteri):
    print ((ifade+kacisKarakteri)*tekrarSayisi)
nesnem=MetinSinifi('Sınıf oluşturduk.',5, '\n')
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
```

16.2.4. Aşırı Yükleme (Over Loading)

Bilindiği gibi fonksiyonlarda, kullanmak zorunda olunan parametreler belirtilmektedir. NTP'de bir fonksiyon aynı isimle farklı parametrelerle tanımlanabilir. Python'da bir sınıf içinde aynı isimde birden fazla fonksiyon tanımlamak yani aşırı yükleme yapmak (method overloading) hataya neden olur.

Bu durumun üstesinden gelmek için fonksiyonlar konusunda belirtildiği gibi bazı parametreler için varsayılan değerler atanabilir. Aşağıdaki kodda kacisKarakteri varsayılan olarak '\n' verilmiştir. Böylece sınıf kullanıldığında bu parametreye bir argüman verilmezse hata ile karşılaşılmaz.

Örnek 7

Sınıfı farklı sayıda argümanlarla kullanabilirsiniz.

```
class MetinSinifi:
  def __init__(self, ifade, tekrarSayisi, kacisKarakteri='\n'):
    print ((ifade+kacisKarakteri)*tekrarSayisi)
nesnem=MetinSinifi('Sınıf oluşturduk.',5)
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
Sınıf oluşturduk.
```

Bir sınıf içindeki bütün fonksiyonlar "self" parametresini zorunlu olarak alırlar. Bu sayede fonksiyonlar sınıf içindeki tüm fonksiyonlara ve değişkenlere erişim sağlar. "self" yerine başka bir parametre adını kendiniz verebilirsiniz ancak bu kullanım standarttır. Kullanırken self parametresine argüman verilmez. Sınıfınızı "self" olmadan yazdığınızda hata ile karşılaşırsınız.

Örnek 8

Sınıf oluştururken verdiğiniz argümanlara erişmek için aşağıdaki kodları kullanabilirsiniz.

```
class MetinSinifi:
  def __init__(self, ifade, tekrarSayisi, kacisKarakteri='\n'):
    self.ifade=ifade
    self.tekrarSayisi=tekrarSayisi
    print ((ifade+kacisKarakteri)*tekrarSayisi)
```

Sınıfınızdan bir nesne oluşturarak parametrelerine tekrar erişebilirsiniz. Zorunlu parametrelere değer vererek nesne tanımlayabilirsiniz. Sonrasında fonksiyon içindeki değişkenlere erişim sağlayabilirsiniz.

MetinSinifi'dan tanımladığınız metinNesnesi.ifade ve metinNesnesi.tekrarSayisi argüman olarak verdiğiniz değerlere erişebilirsiniz.

NOT

Argümanlarınızı tanımlarken varsayılan değerli argümanlar, zorunlu argümanlardan sonra gelmelidir.

Örnek 9

```
metinNesnesi=MetinSinifi('Müdür', 3)
print ('Yazılan Metin: ', metinNesnesi.ifade)
print('Tekrar Sayısı: ', metinNesnesi.tekrarSayisi)
Müdür
Müdür
Müdür
Yazılan Metin:  Müdür
Tekrar Sayısı:  3
```

Program yazarken size bir masa lambası lazım olduğunda o sınıftan bir nesne oluşturur (örneğin lambam) bunun tüm özelliklerini belirtebilir, değiştirebilir ve bu sınıfa tanımlı işlevleri kullanabilirsiniz. Böylece hayalinizdeki çok ısınmayan, az enerji tüketen, ışık şiddetini ayarlayabildiğiniz ve istediğiniz malzemeden

yapılmış bir masa lambası oluşturabilirsiniz. lamba1.isikRengi='Sari' lamba1.isikSiddeti='100' lamba1.aydinlat()

aydinlat() lamba1 nesnesine ait bir metot yani fonksiyondur. Özellikler (isikRengi, isikSiddeti) ise aslında değerleri tutan değişkenlerdir.

Örnek 10

Bir Masa Lambası sınıfı oluşturunuz.

```
class MasaLambasi:
#Örnek bir sınıf
  isikRengi='sarı' # özellik
  isikDurum=False # özellik
  def isikAc(self): # metot, fonksiyon
    self.isikDurum=True
    return 'Aydınlık'
```

Örnek 11

Sınıfınızı adıyla çağırabilirsiniz.

```
MasaLambasi.isikRengi
'sarı'
```

Örnek 12

Sınıfınızdan bir nesne türetebilir ve sınıfınızda tanımladığınız özelliklere erişebilirsiniz.

```
lambam=MasaLambasi()
print (lambam.isikAc())
print (lambam.isikDurum)
Aydınlık
True
```

Örnek 13

Oluşturduğunuz nesneyi silmek için del( ) fonksiyonunu kullanabilirsiniz.

```
del(lambam)
print(lambam.isikDurum)
#Nesne silindiği için hata ile karşılaşılır.
NameError                 Traceback (most recent call last)
<ipython-input-67-c18bef2cc0fd> in <module>()
----> 1 del(lambam)
      2 print(lambam.isikDurum)
      3 #Nesne silindiği için hata ile karşılaşılır.

NameError: name 'lambam' is not defined
```

Örnek 14

Masa lambası sınıfınızı yeniden tanımlayınız. Bu sınıfa farklı özellikler ve işlevler ekleyiniz.

```
class MasaLambasi:
  __doc__='Masa Lambası Sınıfı'
  isikDurum=False
  def __init__(self,    isikSiddeti, isikRengi='sarı', garantiSuresi=2):
    self.isikSiddeti=isikSiddeti
  def isikAc(self):
    self.isikDurum=True
  def isikKapat(self):
    self.isikDurum=False
  def isigiArtir(self, artirmaMiktari):
    self.isikSiddeti+=artirmaMiktari
```

Örnek 15

Sınıfınızdan bir nesne türetiniz. Bunun için parametrelere argümanları yani değerleri girebilirsiniz.

```
lambam=MasaLambasi(60, garantiSuresi=3)
```

Argümanları parametre sırasına göre verebileceğiniz gibi argüman adlarını kullanarak da değer verebilirsiniz.

Örnek 16

Nesnenin özelliklerini ve işlevlerini kullanabilirsiniz.

```
lambam.isikAc() #şimdi ışığı açalım
print ('Işık Açık mı? ', lambam.isikDurum) #Işığın durumu
Işık Açık mı?  True
```

Örnek 17

Işığın şiddetini arttırarak son durumu kontrol edebilirsiniz.

```
lambam.isigiArtir(5)
print('Işık Şiddeti:', lambam.isikSiddeti)
Işık Şiddeti: 65
```

Örnek 18

Işığı kapatabilirsiniz.

```
lambam.isikKapat()
print('Işık Açık mı?', lambam.isikDurum)
Işık Açık mı? False
```

Masa Lambası nesnesini kullanarak programınızda ihtiyaç duyduğunuz masa lambasının tüm özelliklerini ve tüm işlevlerini tek bir yapı içinde (nesne olarak) kontrol edebilirsiniz. Bu sınıfı aynı şekilde bir bütün olarak diğer projelerinizde de kullanabilirsiniz.

Örnek 19

Aşağıdaki örnekte öğrenci verilerini ve öğrencilere ait işlemleri yaptığınız bir sınıf oluşturunuz.

```
class Ogrenci():
  __doc__='Öğrenci sınıfı'
  ogrenciler=[]
  def __init__(self, ogrenciAdiSoyadi):
    self.ogrenciAdiSoyadi = ogrenciAdiSoyadi
    self.dersleri = []
    self.sinavlar=[]
    self.ogrenciEkle(self.ogrenciAdiSoyadi)

  def ogrenciEkle(self, adSoyad):
    self.ogrenciler.append(adSoyad)
    print('{} adlı kişi öğrencilere eklendi.'.format(adSoyad))

  def ogrenciListesiYazdir(self):
    print('Öğrenci listesi')
    for ogrenci in self.ogrenciler:
      print(ogrenci)

  def dersEkle(self, dersAdi):
    self.dersleri.append(dersAdi)

  def ogrencininDersleri(self):
    print('{} adlı kişinin dersleri:'.format(self.ogrenciAdiSoyadi))
    for ders in self.dersleri:
      print(ders)

  def sinavPuaniEkle(self, sinavPuani):
    self.sinavlar.append(sinavPuani)

  def ogrencininSinavPuanlari(self):
    print('{} adlı kişinin sınav sonuçları:'.format(self.ogrenciAdiSoyadi))
    for sinav in self.sinavlar:
      print(sinav)
```

Örnek 20

Bir sınıfın içeriğine bakmak için "dir(sinifAdi)" kullanabilirsiniz. Standart parametrelerin yanı sıra tanımlı olan fonksiyonların listesine erişebilirsiniz.

```
dir(Ogrenci)
['__class__', '__delattr__', '__dict__', '__dir__','__doc__','__eq__','__
format__','__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__','__init_subclass__','__le__','__lt__','__module__','__ne__','__
new__','__reduce__','__reduce_ex__', '__repr__', '__setattr__', '__
sizeof__', '__str__', '__subclasshook__', '__weakref__', 'dersEkle',
'ogrenciEkle', 'ogrenciListesiYazdir','ogrenciler','ogrencininDersleri',
'ogrencininSinavPuanlari','sinavPuaniEkle']
```

Örnek 21

Sınıfınızdan bir nesne türeterek bir öğrenci ekleyebilirsiniz.

```
ogrenciNesnesi=Ogrenci('Murat Çalışkan')
Murat Çalışkan adlı kişi öğrencilere eklendi
```

Örnek 22

Öğrenci sayısına bakabilirsiniz.

```
print ('Öğrenci Sayısı:', len(ogrenciNesnesi.ogrenciler))
```

Örnek 23

Öğrencilere dersleri ve sınav puanlarını ekleyebilirsiniz.

```
ogrenciNesnesi.dersEkle('Türkçe')
ogrenciNesnesi.sinavPuaniEkle(95)
ogrenciNesnesi.dersEkle('Matematik')
ogrenciNesnesi.sinavPuaniEkle(100)
ogrenciNesnesi.dersEkle('Fen Bilgisi')
ogrenciNesnesi.sinavPuaniEkle(100)
```

Örnek 24

Öğrencinin derslerini ve sınav puanlarını listeleyebilirsiniz.

```
ogrenciNesnesi.ogrencininDersleri()
ogrenciNesnesi.ogrencininSinavPuanlari()
Murat Çalışkan adlı kişinin dersleri:
Türkçe
Matematik
Fen Bilgisi
Murat Çalışkan adlı kişinin sınav sonuçları:
95
100
100
```

Örnek 25

Şimdi bir öğrenci daha ekleyiniz ve öğrenci listesini yazdırınız.

```
ogrenciNesnesi.ogrenciEkle('Ahmet Birinci')
Ahmet Birinci adlı kişi öğrencilere eklendi.
len(ogrenciNesnesi.ogrenciler)
2
```

Örnek 26

Öğrenci listesini görebilirsiniz.

```
ogrenciNesnesi.ogrenciListesiYazdir()
Öğrenci listesi
Murat Çalışkan
Ahmet Birinci
```

16.3. Sınıf Metotları

Sınıf tanımlarken sınıftan bir nesne türetmeden fonksiyonlarına erişmek için "@classmethod" kullanılır.

Örnekte "Ogrenciler" sınıfı oluşturdunuz ve içinde öğrenci verilerinin bulunduğu bir liste var. Fonksiyon, bir sınıf metodu olarak tanımlanırken satır başına "@classmethod" eklenir ve "self" tanımında olduğu gibi fonksiyona da "cls" eklenir. Bu parametre adı değiştirilebilir ancak standart olarak bu şekilde kullanılmaktadır. Sınıf metotları sınıftan nesne türetmeden doğrudan sınıfın metotlarına erişme olanağı sağlar. Aşağıdaki sınıfta __init__ sınıf içindeki metotlar kullanılabilmektedir. Aynı zamanda bu metotlara sınıf üzerinden doğrudan erişilebilmektedir.

Örnek 27

```
class Ogrenciler():
    def __init__(self, sorgu=None, sirasi=None):
      self.ogrenciListe = [('Murat Çalışkan', 11, 131), ('Ahmet Birinci', 11, 124),
('Emre Hızlı', 12, 135), ('Murat Çalışkan', 12, 155)]
      if not sorgu and not sirasi:
        listem = self.ogrenciListe
      else:
        listem = [li for li in self.ogrenciListe if sorgu == li[sirasi]]
      for i in listem:
        print(*i, sep=', ')

    @classmethod #Bir sınıf metodu tanımlanıyor
    def ogrenciAdindanSorgula(cls, adi): # self gibi burada da cls parametresi
kullanılıyor
        cls(adi, 0)

    @classmethod
    def ogrenciSinifindanSorgula(cls, sinifi):
        cls(sinifi, 1)

    @classmethod
    def ogrenciNumarasindanSorgula(cls, numarasi):
        cls(numarasi, 2)
```

Örnek 28

Doğrudan sınıfın metotlarını çağırarak ve adını verdiğiniz bir öğrencinin bilgilerini listeleyiniz.

```
Ogrenciler.ogrenciAdindanSorgula('Murat Çalışkan')
Murat Çalışkan, 11, 131
Murat Çalışkan, 12, 155
```

Örnek 29

Belirli bir sınıftaki öğrencilerin listesini alınız.

```
Ogrenciler.ogrenciSinifindanSorgula(12)
Emre Hızlı, 12, 135
Murat Çalışkan, 12, 155
```

Örnek 30

Öğrenci numarasından sorgulama yapınız.

```
Ogrenciler.ogrenciNumarasindanSorgula(131)
Murat Çalışkan, 11, 131
```

Örnek 31

Tüm öğrencileri listelemek için Ogrenciler() argüman kullanmadan çağırmalısınız. Bunun için sınıfınızdan bir nesne türetiniz.

```
tumOgrenciListe=Ogrenciler()
Murat Çalışkan, 11, 131
Ahmet Birinci, 11, 124
Emre Hızlı, 12, 135
Murat Çalışkan, 12, 155
```

16.4. Kapsülleme

Sınıf içindeki metotlara ve özelliklere erişimleri kısıtlamak için kapsülleme yöntemi kullanılır. Bir sınıf tanımlanırken kapsülleme ile erişilmesi, değiştirilmesi istenmeyen veya sadece kısıtlı erişim verilmek istenilen özellikler ve/veya metotlar tanımlanabilir. Bunun için istenilen metodun veya özelliğin başına "__" eklenmelidir.

Örnek 32

Örnekte self.__TCNo=TCNo kullanımı öğrencinin kimlik numarasına dışarıdan erişimi engeller.

```
class Ogrenciler:
  def __init__(self,adi,yasi,TCNo):
    self.adi=adi
    self.yasi=yasi
    self.__TCNo=TCNo
  def ogrenciYaz(self):
    print('Öğrenci Bilgileri')
    print(self.adi)
    print(self.yasi)
    print(self.__TCNo)
ogrenci=Ogrenciler('Murat Çalışkan',17,54639876211)
ogrenci.ogrenciYaz()
Öğrenci Bilgileri
Murat Çalışkan
17
54639876211
```

Örnek 33

Öğrenci bilgilerine doğrudan erişmeyi deneyiniz.

```
print (ogrenci.adi)
print (ogrenci.yasi)
Murat Çalışkan
17
```

Örnek 34

Öğrencinin adı ve yaşı değişkenleri gizli olmadığı için sınıf dışından doğrudan ulaşılabilir. Öğrencinin kimlik numarasına erişmeye çalışınız.

```
print(ogrenci.__TCNo)
AttributeError                          Traceback (most recent call last)
<ipython-input-212-62da05ea5506> in <module>()
----> 1 print(ogrenci.__TCNo)

AttributeError: 'Ogrenciler' object has no attribute '__TCNo'
```

Örnek 35

Sınıf içinde özel olarak tanımlanmış bu tür özelliklere erişmek için "set" ve "get" metodları (diğer NTP dillerinde de farklı adlarla) bulunur. "get" özellikleri okumak "set" ise değiştirmek için kullanılır.

```
class Ogrenciler:
  def __init__(self,adi,yasi,TCNo):
    self.adi=adi
    self.yasi=yasi
    self.__TCNo=TCNo

  def ogrenciYaz(self):
    print('Öğrenci Bilgileri')
    print(self.adi)
    print(self.yasi)
    print(self.__TCNo)

  def get_TCNo(self):
    return str(self.__TCNo)[7:]
```

Yukarıda tanımlanan def get_TCNo(self): ile kimlik numarasının son 4 hanesi döndürülmüştür. Kapsülleme sınıfın içindeki özellikleri ve metotları korumayı sağlamaktadır. Kapsülleme aynı zamanda verinin güvenliğini sağlar.

Örnek 36

```
ogrenci=Ogrenciler('Sezai', 43, 12345678901)
print(ogrenci.get_TCNo())
8901
```

Örnek 37

"get" metodu ile yazma özelliği verebilir, "set" metodu ile de değişkene yeni değer atayabilirsiniz.

```
class Ogrenciler:
  def __init__(self,adi,yasi,TCNo):
    self.adi=adi
    self.yasi=yasi
    self.__TCNo=TCNo
   def ogrenciYaz(self):
    print('Öğrenci Bilgileri')
    print(self.adi)
    print(self.yasi)
    print(self.__TCNo)
    def get_TCNo(self):
    return str(self.__TCNo)[7:]

  def set_TCNo(self,TcNoDuzelt):
    self.__TCNo=TcNoDuzelt
```

Örnek 38

Yanlış girilen kimlik numarasının düzeltildiğin varsayınız. Öncelikle bir öğrenci tanımlayınız.

```
ogrenci=Ogrenciler('Kaan Emre', 33, 23142343655)
print('Öğrenci TC No son 4 hane: ', ogrenci.get_TCNo())
Öğrenci TC No son 4 hane:  3655
```

Örnek 39

Öğrencinin kimlik numarasını düzeltebilir ve tekrar yazdırabilirsiniz.

```
ogrenci.set_TCNo(12309768687)
print ('Öğrenci TC No son 4 hane: ', ogrenci.get_TCNo())
Öğrenci TC No son 4 hane:  8687
```

16.5. Kalıtım (Miras Alma)

Kalıtım bir sınıfın başka bir sınıfın metotlarını ve özelliklerini kendi bünyesine almasıdır. Bu metot ve özelliklerin devralındığı (Miras alma da denir.) sınıfa üst sınıf, devralan sınıfa ise alt sınıf denir. Alt sınıflar, üst sınıfın tüm metotlarını ve özelliklerini devralır, ancak farklı özellikler ve metotlar da ekleyebilir. En temel sınıf türü, genellikle diğer tüm sınıfların ebeveynleri olarak miras aldığı bir nesnedir. Örneğin bir "kedi" sınıfının metotları ve özellikleri neler olabilir? Her kedinin bir adı, yaşı ve cinsi vardır. Buna ek olarak her kedi cinsinin farklı özellikleri de vardır. Her kedi cinsi, üst sınıf olan kedi sınıfının alt sınıfıdır. Üst sınıflara "ebeveyn (parent) sınıf" alt sınıflara ise "çocuk (child) sınıf" denir.

Örnek 40

Bu örnekte bir kedi üst sınıfı tanımlayınız.

```
#Üst Sınıf
class Kedi:
    tur = 'Memeli'

    def __init__(self, adi, yasi):
      self.adi = adi
      self.yasi = yasi

    def ozellikler(self):
      return "{} {} yaşında".format(self.adi, self.yasi)

    def miyavla(self, sesi):
      return "{}  {} diye miyavlar".format(self.adi, sesi)
```

Örnek 41

Kedi üst sınıfından farklı türdeki kediler için alt sınıflar tanımlayınız.

```
# Kedi alt sınıfından miras alan alt sınıf
class PersKedisi(Kedi):
  def tuyleri(self, tuyuNasil):
    return "{} tüyleri {}".format(self.adi, tuyuNasil)

class VanKedisi(Kedi):
  def gozRengi(self, gozRengiNe):
    return "{} gözleri {}".format(self.adi, gozRengiNe)
```

Alt sınıftan "Pisi" adında ve "4" yaşında bir kedi türetiniz.

```
pisi = VanKedisi('Pisi', 4)
print(pisi.ozellikler())
print(pisi.gozRengi('Mavi ve kahverengi'))
Pisi 4 yaşında
Pisi gözleri Mavi ve kahverengi
```

Örnek 42

"PersKedisi" ve "VanKedisi" alt sınıflarını "Kedi" üst sınıfından metotları ve özellikleri devralarak oluşturdunuz. Bu iki alt sınıfa kendilerine özgü yeni özellikler ve metotlar tanımlayarak örneğe devam ediniz.

```
persia= PersKedisi('Persia', 4)
print(persia.ozellikler())
print(persia.miyavla('meaaaw'))
print(persia.tuyleri('Yumuşak'))
Persia 4 yaşında
Persia meaaaw diye miyavlar
Persia tüyleri Yumuşak
```

Örnek 43

Diğer alt sınıfın da kendine ait özelliklerini ve üst sınıftan devraldığı özellik ve metotları çağırabilirsiniz. Üst sınıftan devralınan metotları ve özellikleri geçersiz (overridding) kılabilirsiniz.

```
class VanKedisi(Kedi):
  def gozRengi(self, gozRengiNe):
    return "{} gözleri {}".format(self.adi, gozRengiNe)
  def ozellikler(self):
    return "Türü: Van kedisi adı:{} ve {} yaşında".format(self.adi, self.yasi)
    #overridding
pisi = VanKedisi('Pisi', 4)
print(pisi.ozellikler())
print(pisi.gozRengi('Mavi ve kahverengi'))
Türü: Van kedisi adı:Pisi ve 4 yaşında
Pisi gözleri Mavi ve kahverengi
```

Yukarıdaki örnekte özellikler metodunu geçersiz kılınarak üst sınıftan farklı bir değer döndürülmüştür.

16.6. Bölüm Sonu Örnekleri

1. Bir araba sınıfı tanımlayınız.

Özellikler:

marka, model, kapı sayısı, rengi, fiyatı, motor seri numarası (gizli özellik). Marka ve model dışındaki tüm özelliklerin varsayılan değeri yok (None), kapı sayısı varsayılan değeri=2 olarak tanımlanacaktır. Arabanın özelliklerini yazdıran bir metot tanımlayınız.

2. Bu araba sınıfından sarı renkli 2 kapılı 1993 model bir "Marka 00" marka araba türetiniz. Motor şase numarası 123456789 olsun. Daha sonra bu arabanın özelliklerini yazdırınız.

3. Motor şase numarasını okumak ve değiştirmek için sınıfı düzenleyiniz. Şase numarasını değiştirerek ekrana yazdırınız.

4. Araba sınıfından kalıtım yoluyla bir sınıf üretiniz ve gazVer metodunda ekrana 'Gaza basıldı' yazmasını sağlayınız. Bu alt sınıfa çelik jant özelliği ekleyiniz.

Cevaplar

1.
```
class Araba:
  __doc__='Araba Sınıfı'
  isikDurum=False
  def __init__(self,    marka, model, kapiSayisi=2, rengi=None, fiyat=None,
motorSeriNo=None):
    self.marka=marka
    self.model=model
    self.kapiSayisi=kapiSayisi
    self.rengi=rengi
    self.__fiyat=fiyat
    self.__motorSeriNo=motorSeriNo
  def arabaOzellikleri(self):
    print ('{} model  {} kapılı  {} renkli {} marka araç'.format(self.model,
self.kapiSayisi, self.rengi, self.marka))
```

2.
```
arabam=Araba('Marka 00', 1993,2,'sarı',motorSeriNo=123456789)
arabam.arabaOzellikleri()
1993 model  2 kapılı  sarı renkli Marka 00 marka araç
```

3.
```
class Araba:
  __doc__='Araba Sınıfı'
  isikDurum=False
  def __init__(self,    marka, model, kapiSayisi=2, rengi=None, fiyat=None,
motorSeriNo=None):
    self.marka=marka
    self.model=model
    self.kapiSayisi=kapiSayisi
    self.rengi=rengi
    self.__fiyat=fiyat
    self.__motorSeriNo=motorSeriNo
  def arabaOzellikleri(self):
    print ('{} model  {} kapılı  {} renkli {} marka araç motor seri no: {}'.
format(self.model,  self.kapiSayisi, self.rengi, self.marka, self.__motorSeriNo))
  def get_motorSeriNo(self):
    return self.__motorSeriNo
  def set_motorSeriNo(self, motorSeriNoDuzeltme):
    self.__motorSeriNo=motorSeriNoDuzeltme

arabam=Araba('Marka 123', 1981)
arabam.arabaOzellikleri()
arabam.set_motorSeriNo(123456) #yeni seri numara değerini verdik
print(arabam.get_motorSeriNo())
1981 model  2 kapılı  None renkli Marka 123 marka araç motor seri no: None
123456
```

4.
```
class ArabaSinifim(Araba):
  çelikJant=True
  def gazVer(self):
    print (self.marka, 'Durum: Gaza basıldı')

tomofil=ArabaSinifim('Marka 456', 1992)
tomofil.gazVer()
tomofil.arabaOzellikleri()
Marka 456 Durum: Gaza basıldı
1992 model  2 kapılı  None renkli Marka 456 marka araç motor seri no: None
```

KAYNAKÇA

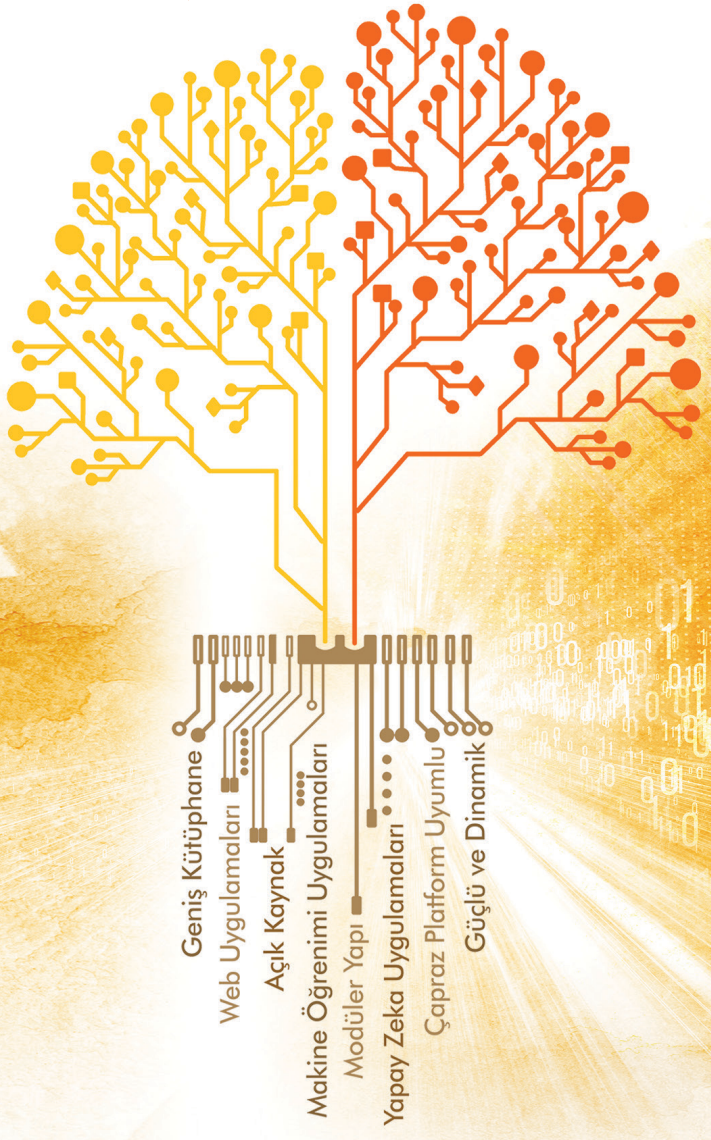
Bu kitabın yazım aşamasında aşağıdaki kaynaklardan yararlanılmıştır.

- https://python-istihza.yazbel.com/
- https://docs.python.org/3/
- https://www.netacad.com/ üzerinde bulunan Python Institute tarafından hazırlanan içerikler.
- **Python içerikleri** https://www.mobilhanem.com/python-egitimi/ 20.04.2020 tarihinde ulaşıldı.
- **Python örnekleri** https://www.yazilimkodlama.com/programlama/python-ornekleri/ 02.03.2018 tarihinde ulaşıldı.
- **Python içerikleri** https://www.yazilimbilimi.org/python-liste-veri-tipi/ 15.04.2020 tarihinde ulaşıldı.
- **Python içerikleri** http://www.veridefteri.com/2018/01/30/python-programlamaya-giris-liste-metodlari/ 16.04.2020 tarihinde ulaşıldı.
- **Python içerikleri** https://www.pythondersleri.com/2013/07/moduller.html 17.04.2020 tarihinde ulaşıldı.
- **Python içerikler** https://yazilimhanem.com/python-moduller/ 22.04.2020 tarihinde ulaşıldı.
- **Python içerikler** https://www.pythontr.com/makale/python-sozlukler-ve-kullanimi-42 23.04.2020 tarihinde ulaşıldı.
- **Python içerikler** https://www.pythondersleri.com/2013/05/sozlukler.html 24.04.2020 tarihinde ulaşıldı.
- **Python içerikler** https://tr.qwe.wiki/wiki/Logo_(programming_language)
- TDK, 2020 Türk Dil Kurumu https://sozluk.gov.tr/ adresinden 28.04.2020 tarihinde alınmıştır.

NOTLAR

NOTLAR

PYTHON EĞİTİMİ



Birleşmiş Milletler Eğitim, Bilim ve Kültür Kurumu
UNESCO Türkiye Millî Komisyonu

ÖĞRETMEN YETİŞTİRME VE GELİŞTİRME GENEL MÜDÜRLÜĞÜ

PYTHON EĞİTİMİ

Güçlü ve Dinamik

Modüler Yapı

Açık Kaynak

Web Uygulamaları

Yapay Zeka Uygulamaları

Çapraz Platform Uyumlu

Geniş Kütüphane

Makine Öğrenimi Uygulamaları

python eğitimi