3. EDİSYON

TEMEL PYTHON

Dr. Âdem ZENGİN

değişim®

TEMEL PYTHON

Dr. Âdem ZENGİN

3. Edisyon

2021

değişim®

TEMEL PYTHON / Dr. Âdem ZENGİN

TASARIM: İbrahim Cihan

ISBN: 978-625-8466-06-5

Çatalçeşme Sk. 52/2
Cağaloğlu / İstanbul
Tel : + 90 (212) 514 29 71
Faks : + 90 (212) 514 42 31

Postane Sk. 3 Sakarya
Tel : + 90 (264) 278 56 39 / 272 13 81
Faks : + 90 (264) 273 52 99
www.degisimkitap.com
www.degisimyayinlari.net
bilgi@degisimyayinlari.net

İÇİNDEKİLER

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيمِ

Rahman ve Rahim Olan Allah'ın Adıyla,

Âlemlerin Rabbi Allâh'a hamdolsun. Her şeyi bir ölçüye göre hikmetle yaratan ve yaşatan Allâh'a hamdolsun. Tüm bunların anlamını ve amacını bize bildirmek için aramızdan seçip kendisine elçi, bize de önder ve model kıldığı Muhammed'e (sav) salât ve selâm olsun. Onun getirdiği mesaj ve yaratılış kodlarımızdaki merak duygusu bizi Allâh'ın evrene koyduğu yasaları araştırma ve öğrenmeye sevketmektedir. Allâh'tan bizi en doğru, en güzel ve en iyi neticelere ulaştırmasını, bu yolu bize kolaylaştırmasını temenni ediyoruz. İnanıyoruz ki bu yolculukta, her yeni bilgiyle birlikte var edici Kudret'i daha yakından tanıyacağız. Böylece O'na duyduğumuz hayranlığımız, sevgimiz, saygımız artacak ve bir kez daha O'ndan başka ilâh olmadığına şâhit olacağız. Bunun yanında hiç şüphesiz Allâh'ın bir ikramı olarak hayatımızı kolaylaştıracak veya problemlerimizi çözecek bilgilere de ulaşmış olacağız.

Yaşadığımız çağda, çözülmeyi bekleyen problemlerimizin veya keşfetmek istediğimiz sistemlerin karmaşıklık düzeyi bilgisayarları etkin şekilde kullanmayı gerektirir oldu. Bilgisayar sistemlerini, sunulan hazır paket programlarla kullanmak genel ve yaygın ihtiyaçları karşılamaktadır. Bu programlar ticari amaçla üretildiği için genel ve yaygın ihtiyaçlara göre dizayn edilmekte böylece daha fazla müşteriye hitap etmeleri sağlanmaktadır. Ancak çoğu zaman keşfetmek istediğimiz sistemler veya çözmek istediğimiz problemler kendilerine özgü hususiyetler taşımaktadır. Hazır programların tüm bu hususiyetlere cevap verecek esneklikte olmaları mümkün değildir. Dolayısı ile bilgisayar programlama bilgisine sahip olmak, yeni keşiflere ve yeni çözümlere ulaşmak için büyük önem taşımaktadır. Yayımladığımız bu kitap ile bilgisayarlar programlama dillerinden biri olan Python programlama dilini temel seviyede ve basit bir anlatımla sunmaya çalıştık. Tüm okuyuculara Allah'tan zihin açıklığı diliyor, istifadelerinin yüksek olmasını temenni ediyorum.

Dr. Âdem ZENGİN
Sakarya - 2021

GİRİŞ

Kitap Hakkında

Bu kitap hiçbir programlama dilinde tecrübesi olmayan veya başka bir dil bildiği halde Python dilinde tecrübesi olmayan kişilere temel seviyede Python öğretmeyi hedeflemektedir. Halihazırda yayımlanmış Python kitaplarının çok kapsamlı, karmaşık ve hacimli olması temel seviyede kitap arayanların ihtiyacını karşılamamaktadır. Buradan hareketle temel seviyede kodlamaya adım atmak isteyen okuyucuların ihtiyacını karşılamak amacıyla bu kitap hazırlanmıştır.

Kitapta yazılı anlatımı yapılan her konunun kodlama örneği de verilmiştir. Her bölümün sonunda çözümlü sorular ve bir uygulama sorusu ve çözümü yer almaktadır. Böylece yeni öğrenilen bilgiler üzerinde pratik yapılmasına ve bilgilerin kalıcı olmasına imkân verilmiştir. Kitaptaki konu sıralaması basitten zora doğru ve en temel bilgiden üst bilgilere doğru basamak basamak ilerlemektedir. Dolayısıyla okuyucu, satırlarda veya sorularda karşılaştığı bilgilere yabancılık çekmeden öğrenme yolculuğunu sürdürecektir. Kitabı bitiren kişiler, temel Python bilgisiyle küçük problemlerin çözümü için basit programlar yazacak bir tecrübe

düzeyine erişeceklerdir. Kitabın sonuna gelen okuyucular, Python'u hangi alanda hangi amaçla kullanmak istiyorlarsa o alanda kullanılan Python kütüphanelerini araştırmalılar ve öğrenme yolculuklarına bu kütüphanelerin kullanımı ile devam etmeliler. Özetle bu kitap okuyucuyu Python kütüphanelerini kullanabilecek düzeye getirmeyi hedeflemektedir.

Yazar Hakkında

Âdem ZENGİN, lisans düzeyinde ilahiyat ve mühendislik eğitimi aldı. İngiltere'nin Leeds Üniversitesi'nde Matematik ve Fizik Bilimleri Fakültesi'nde yüksek lisans ve doktora yaptı. Doktora tezini "makromoleküllerin belli şartlar altındaki davranışlarının simülasyonu" konusunda yazdı. 20 yıllık bilgisayar programlama geçmişine sahip olan yazar, akademik yaşamında aktif olarak programlama yapmayı sürdürmekte ve programlama eğitimi vermektedir.

Programlama Hakkında

Programlama, bir programın kaynak kodunun yazılması eylemidir. Bir problemi bilgisayar ile çözmek için oluşturulmuş ayrıntılı plan veya prosedüre ise program denir. Daha spesifik olarak program, çözüme ulaşmak için gerekli olan net ve sıralı talimatlar dizisidir. Yazılım (*software*) ve donanım (*hardware*) ise sırasıyla bilgisayar programları ve bilgisayar ekipmanları için kullanılan kavramlardır.

Bilgisayarlar kullanıcılara genellikle bir program koleksiyonu ile birlikte sunulurlar. Bu program koleksiyonuna işletim sistemi adı verilir. İşletim sisteminin temel işlevi çeşitli işleri yerine getirmek için kullanıcıya

yardımcı olmak ve sistem optimizasyonunu sağlamaktır. İşletim sistemleri hiç şüphesiz programcılar için de çeşitli yardımcı programlar sunmaktadır. Bunlar arasında metin editörleri (*text editor*), derleyiciler (*compiler*) ve hata ayıklayıcılar (*debugger*) sayılabilir. Bazı işletim sistemleri bu programları önden içermektedir. Bazılarına ise bu programlar sonradan yüklenebilmektedir. Sonradan yüklenmek üzere bu programlar ayrı ayrı sunuldukları gibi birlikte paket program şeklinde de sunulmaktadırlar.

Bir programın yazılması için ilk adım yerine getireceği görevin formüle edilmesidir. Bu formülasyona algoritma denmektedir. Bu işlem zihinde, kâğıt üzerinde veya bir akış diyagramı çizim programında yapılabilecek bir işlemdir. Sonra algoritma uygun bir programlama diliyle programcı tarafından o dilin kurallarına göre yazıya yani kodlara dökülür. Yazma işlemi için metin editörü işlevi gören programlar kullanılır. Bir programlama diline aktarılan algoritma, derleyici adı verilen çeviriciler ile makina diline yani bilgisayarın çalıştırabileceği bir programa dönüştürülür. Bir başka açıdan bakıldığında programlama dilleri, bizim dilimiz ile makina dili arasında köprü görevi gören bir ara dildir. Derleyiciler ise bizimle makina arasında çeviri yapan mütercimlerdir.

Bir dilde yazılmış olan algoritma, çalışan bir programa dönüşürken birkaç aşamadan geçer. Bunlardan biri yukarıda bahsedilen derleme işlemidir. Bir diğeri de hata ayıklama işlemidir. Hata ayıklayıcılar, yazılan algoritmayı parça parça çalıştırır. Çeşitli durumları kontrol ederek programcıya bilgi sağlar. Böylece programcı programının sorunsuz bir şekilde çalıştığından emin olur.

Python Hakkında

Python, 1991 yılında Hollandalı programcı Guido van Rossum tarafından geliştirildi. İfadeleri gruplamak için parantez yerine girinti kullanmak gibi özelliklerle kullanımı kolay bir dil olarak tasarlandı. Python ayrıca karmaşık işlerin yalnızca birkaç ifade ile yürütülebilmesi için tasarlanmış oldukça kompakt bir dildir. 2010 yılına gelindiğinde Python, Java ve JavaScript ile birlikte en popüler programlama dillerinden biri haline geldi.

Python ücretsiz ve açık-kaynak bir programlama dilidir. Çok sayıda bilimsel kütüphaneye sahip oluşu ve bunlara sürekli yenilerinin ekleniyor olması popülaritesini artırmaktadır. Bu durum Python'a çok sayıda bilimsel kullanıcı kazandırmaktadır. Programlamaya yeni başlayanlar için diğer dillere göre daha kolay öğrenilmesi, tüm platformlarda çalışıyor olması ve iyi bir dokümantasyona sahip olması Python'u öne çıkaran diğer özellikler arasındadır.

Kodlama Araçları

Kodlama yapmak için temelde iki araç gereklidir. Bunlardan biri metin editörü diğeri derleyicidir. Derleyiciler genellikle hata ayıklama işlevine de sahiptirler. Metin editörleri kod yazımı için kullanılır. Programcılar için geliştirilmiş ücretsiz ve ücretli birçok metin editörü bulunmaktadır. Bu programlar kod yazımını kolaylaştıran çeşitli özelliklere sahip olabilmektedir. Derleyici ise daha önce de bahsedildiği gibi yazılan kodu bilgisayarın anlayacağı dile çeviren bir araçtır. Bilgisayardan gelen geri dönüşlerin ve hata bildirimlerinin insanın anlayacağı formatta ekrana verilmesi yine derleyicilerin rolleri arasındadır. Derleyiciler konsol veya

shell adı verilen bir arayüze sahiptir. Bu arayüz programcının derleyiciye komut vermesini sağlar. Ayrıca bilgisayardan gelen geri bildirimler bu arayüz ile programcıya iletilir.

Her metin editörü derleyici ile birlikte sunulmaz. Bazıları sadece kod yazımı için kullanılır. Bu durumda derleme ve programı çalıştırma işlemi ayrı bir pencerede ayrıca bir işlem olarak gerçekleştirilir. Bazı metin editörleri bir derleyici ile harici entegrasyona elverir ve böylece bu işlemler otomatize edilebilir. Metin editörü ile derleyicinin bir arada bulunduğu tümleşik programlar da vardır. Bu tür programlara *Integrated Development Environment* veya kısa adıyla IDE denmektedir.

Programlamaya yeni başlayanlar için Python'un sunduğu araçları kullanmalarını öneriyoruz. Başka dillerde programlama yapmış ve bir metin editörüne aşinalık kazanmış kişiler, alışageldikleri metin editörünü kullanmaya devam edebilirler. Kitapta verilen örnek kodlar Python'un 3.9 sürümü ile derlenmiş ve çalıştırılmış kodlardır. Siz okuyucularımız, kullandığınız derleyici sürümüne dikkat etmelisiniz. Bizim kitapta sunduğumuz örnek kodlar için Shell'den[1] aldığımız geri bildirimler ile sizin aldığınız geri bildirimler farklı çıkıyor ise bunun muhtemel sebeplerinden biri kullandığınız derleyicinin sürümüdür. Okuyucularımıza 3.8.0 ve üzeri Python sürümüne sahip olmalarını öneriyoruz.

[1] Shell, Python kodlarının derlenip çalıştırıldığı ve geri dönüşlerin alındığı penceredir. Konsol olarak da ifade edilmektedir.

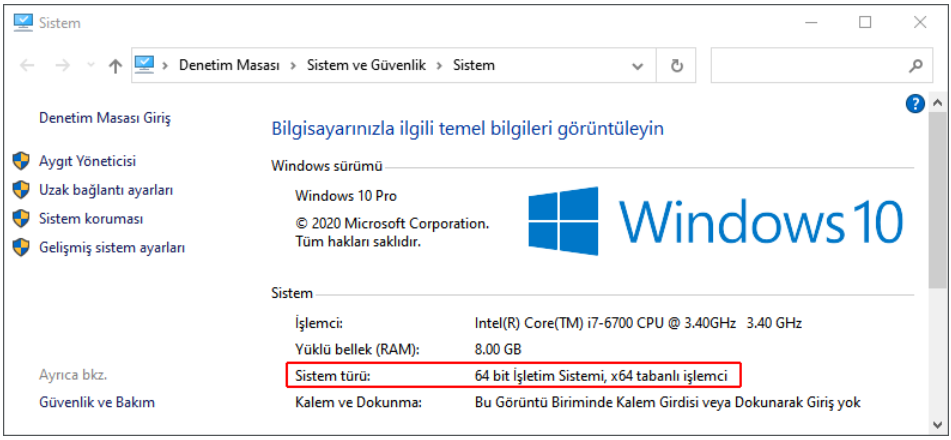
Python Kurulumu

Bu bölümde Python dilinde programlama yapmak için Python tarafından sunulan paket programın Microsoft Windows ve macOS işletim sistemlerine kurulumu gösterilecektir. Kitapta sunduğumuz örnek kodlar, Python'un IDLE (*Integrated Development Learning Environment*) programı ile yazılmış ve çalıştırılmış kodlardır. Kodların renklendirilmesinin (tema) ve alınan geri bildirimlerin tam uyumu için okuyucularımızın da bu araçları kullanmalarını tavsiye ediyoruz.

Microsoft Windows

Microsof Windows işletim sisteminde yüklü bir Python paketi olup olmadığını kontrol etmek için "Komut İstemi" uygulaması açılır. Açılan pencereye **`Python --version`** komutu girilir ve bir Python sürüm bilgisi alınıp alınmadığı kontrol edilir. Bilgisayar bu komuta, üç haneli 2.x.x veya 3.x.x formatında bir sürüm numarası ile cevap vermez ise bilgisayarda yüklü bir Python paketinin bulunmadığı anlaşılır. Bu durumda aşağıdaki adımları izleyerek Python yükleyebilirsiniz.

Windows 10 kullanıcıları Microsoft Store uygulamasına "Python" yazıp arama yaptıklarında çeşitli Python sürümlerini sonuçlar arasında göreceklerdir. Bunlar arasından bir sürümü seçerek yükleme işlemini yapabilirler. Microsoft Store uygulamasından yükleme yapamayanlar ise Windows için farklı Python sürümlerinin bulunduğu https://www.Python.org/downloads/windows resmi indirme sayfasına giderek sunulan paketlerden bilgisayarları için doğru seçimi indirmeleri ve yüklemeleri gerekir.

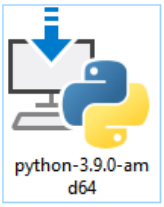


Şekil 1. *Windows sürümünün ve sistem türünün kontrol edildiği pencere.*

Bilgisayar için doğru Python seçimini yapmak için iki hususa dikkat etmek gerekir. Birinci husus, kullanılan Windows işletim sisteminin Windows 7 mi, Windows 8 mi yoksa Windows 10 mu olduğudur. İkinci husus ise sistemin 32-bit mi yoksa 64-bit mi olduğudur. Bu iki hususa dair bilgi edinmek için bilgisayarınızda sırasıyla **"Denetim Masası > Sistem ve Güvenlik > Sistem"** sayfası açılmalı ve **"Sistem türü"** bilgisi **Şekil 1**'de gösterildiği gibi kontrol edilmelidir.

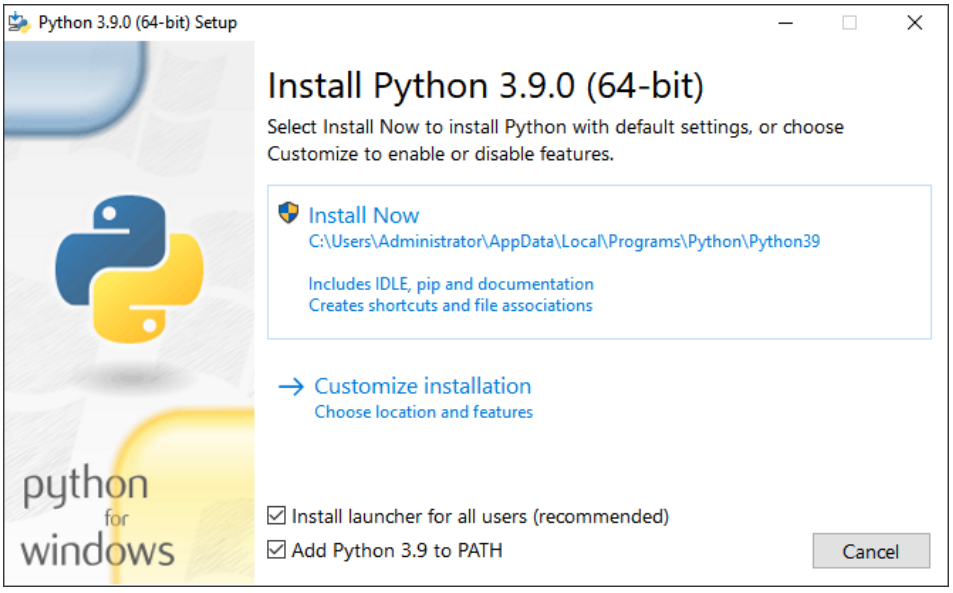
Yukarıda verilen indirme adresinde, kararlı sürümler "*Stable Releases*" başlığı altında verilmektedir. 64-bit Windows 7 sistemine sahip olanlar Python 3.8.x başlığı altındaki "*Windows x86-64 executable installer*" seçeneğini indirmeliler. 32-bit Windows 7 sistemine sahip olanlar Python 3.8.x başlığı altındaki "*Windows x86 executable installer*" seçeneğini indirmeliler. Windows 8 ve Windows 10 kullanıcıları, aynı adımları Python 3.9.0 veya üzeri sürümlere uygulayarak indirme işlemini gerçekleştirmeliler.

Doğru kurulum paketi indirildikten sonra pakete (**Şekil 2**) çift tıklayarak kuruluma geçilir ve aşağıdaki adımlar izlenerek kurulum tamamlanır.

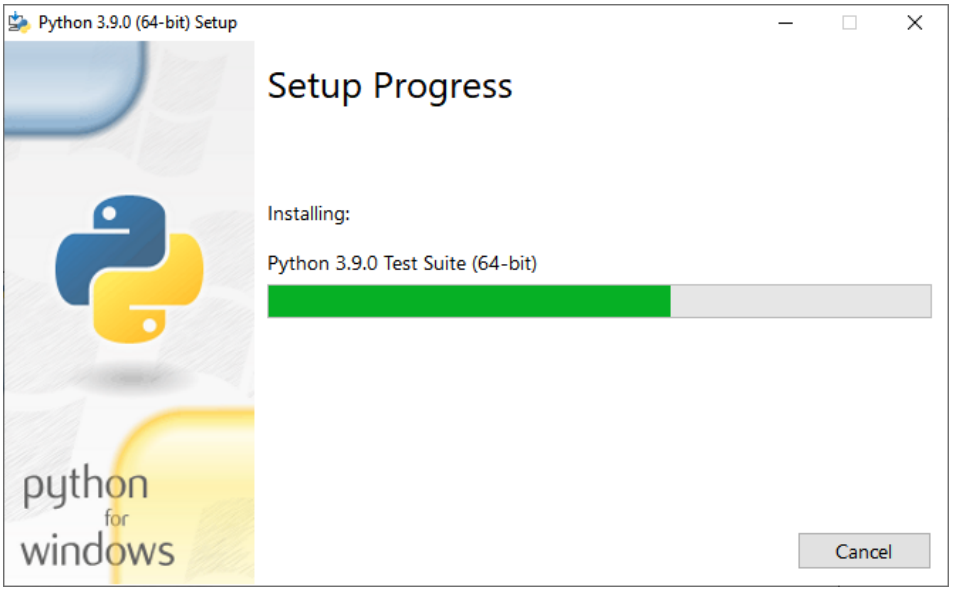


Şekil 2. *İndirilmiş örnek bir kurulum paketinin ikonu.*

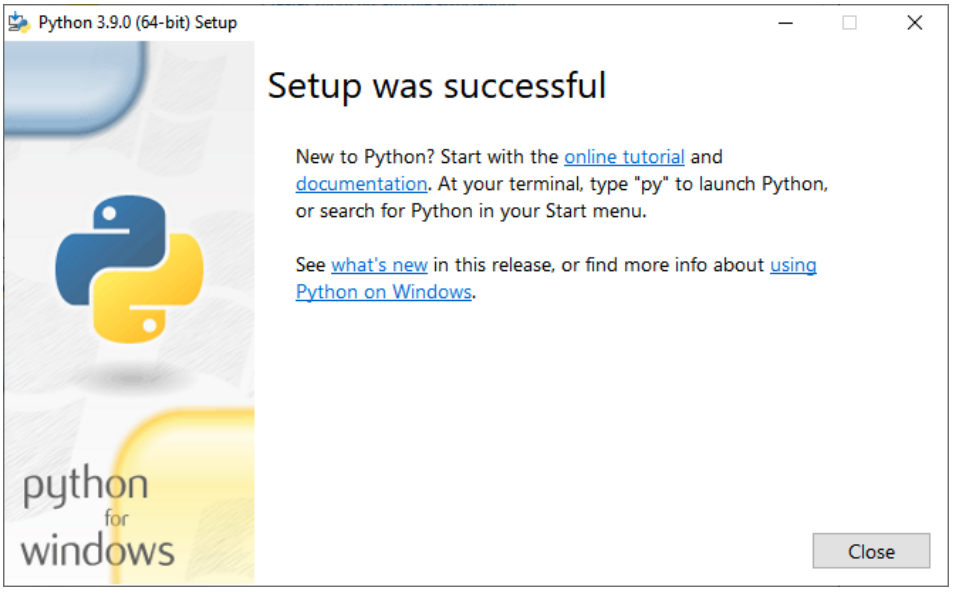
İlk pencerede (**Şekil 3**) "**Add Python 3.x to PATH**" seçeneğine tik atılır ve sonra "**Install Now**" ile kurulum başlatılır.



Şekil 3. *Python kurulumuna başlangıç penceresi.*

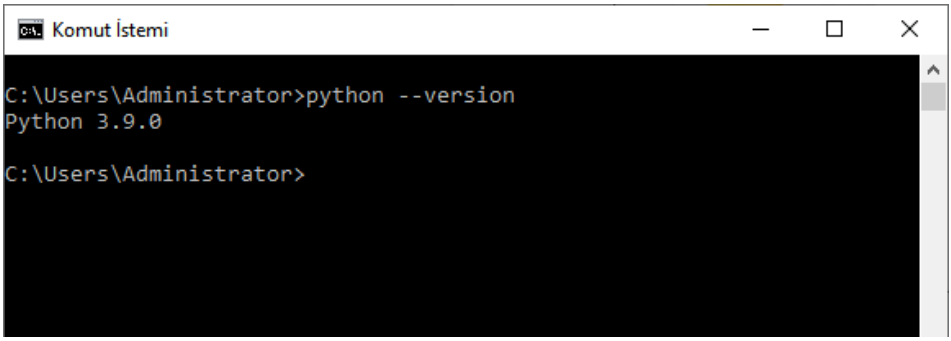


Şekil 4. *Python kurulumunun ilerleme penceresi.*



Şekil 5. *Python kurulumunun bitiş penceresi.*

Bu son pencere (**Şekil 5**) ile birlikte kurulum tamamlanmış olur. Şimdi en başta bahsedilen kurulum kontrolü tekrar yapılabilir. Bunun için "Komut İstemi" programı açılır ve **`Python --version`** komutu çalıştırılır. **Şekil 6**'da gösterildiği (Python 3.9.0) gibi bir sürüm numarası alınıyorsa Python sisteminize başarılı bir şekilde yüklenmiş demektir.

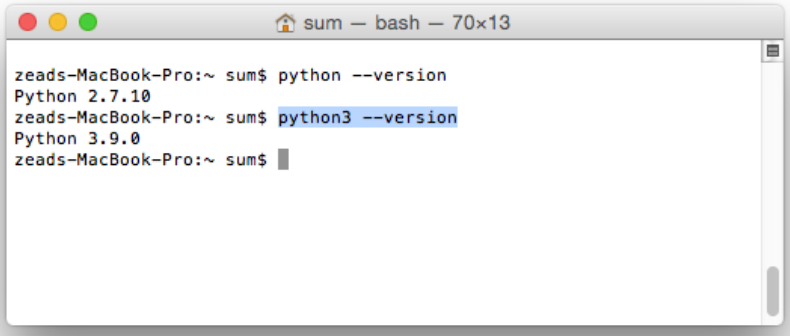


Şekil 6. *Komut İstemi penceresinden Python kurulum kontrolü.*

macOS

macOS'ta yüklü bir Python paketi olup olmadığını kontrol etmek için **"Terminal"** uygulaması açılır. Açılan pencereye **`Python --version`** komutu girilir ve bir Python sürüm bilgisi alınıp alınmadığı kontrol edilir. Bilgisayar bu komuta, üç haneli 2.x.x veya 3.x.x formatında bir sürüm numarası ile cevap vermez ise bilgisayarda yüklü bir Python paketinin bulunmadığı anlaşılır.

Bazı macOS sistemlerinde Terminal'e **`Python3 --version`** komutunun girilmesi gerekir. Çünkü **Şekil 7**'de de gösterildiği gibi salt **`Python`** komutu Python'nun 2.x.x sürümlerine hitap edebilmektedir.



Şekil 7. *Python mevcudiyetinin macOS'ta kontrolü.*

Terminal'den 3.8.0 veya daha yukarı bir sürüm numarası bilgisi alınırsa sistemde istediğimiz Python sürümünün yüklü olduğunu teyit etmiş oluruz. Alınamaz ise aşağıdaki adımlar izlenerek kurulum gerçekleştirilebilir. Bu kitabın yazıldığı tarih itibariyle Python'un güncel sürümü 3.9.0'dır.

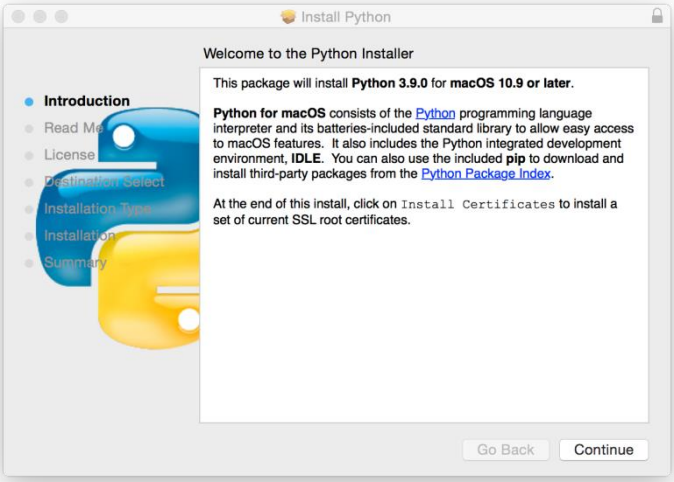
İlk adımda Python'un güncel sürümü, resmi sayfası Python.org'dan (**Şekil 8**) indirilir.

Şekil 8. *Python.org sayfasına bir macOS sisteminden girdiğimizde ve mouse'u "Downloads" sekmesine getirdiğimizde elde ettiğimiz sayfa görüntüsü.*

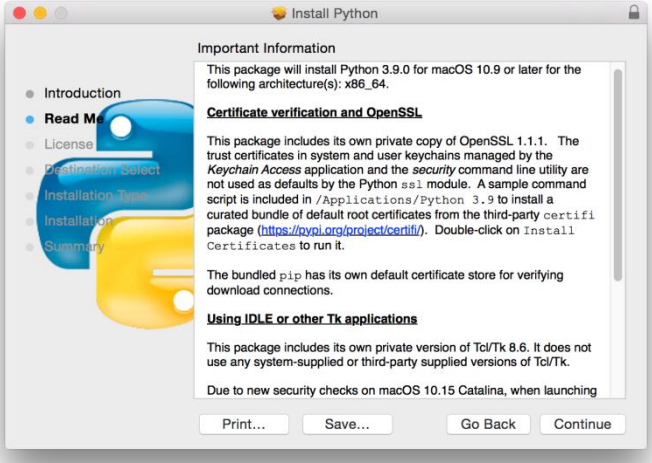
Bilgisayara indirilen pakete (**Şekil 9**) çift tıklanır ve aşağıdaki diğer adımlar izlenerek yükleme işlemi tamamlanır.

python-3.9.0-
macosx10.9.pkg

Şekil 9. *İndirilmiş örnek bir kurulum paketinin ikonu.*

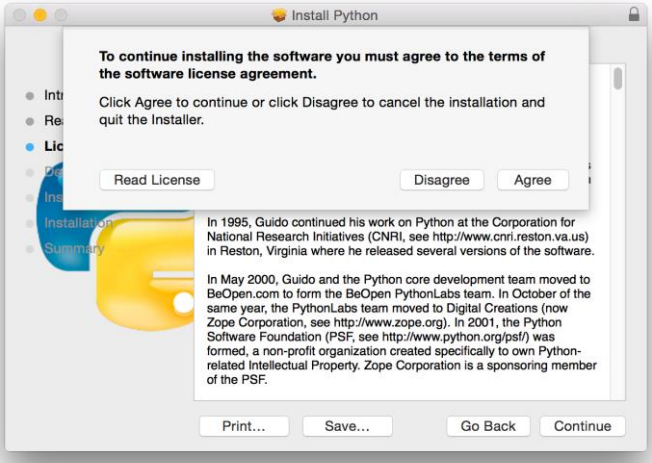


Şekil 10. *Python kurulumu: 1.Adım.*

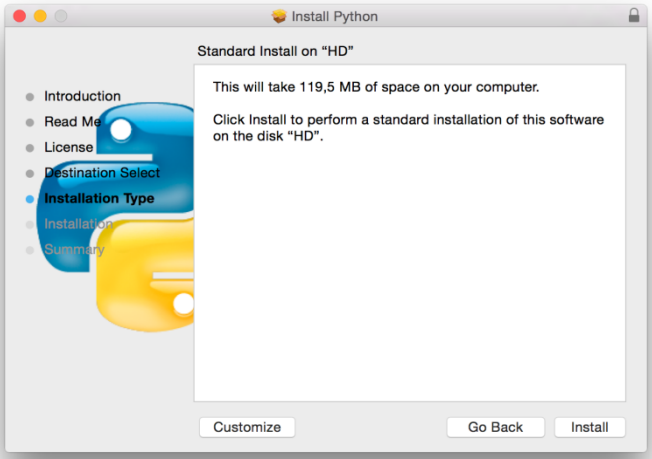


Şekil 11. *Python kurulumu: 2.Adım.*

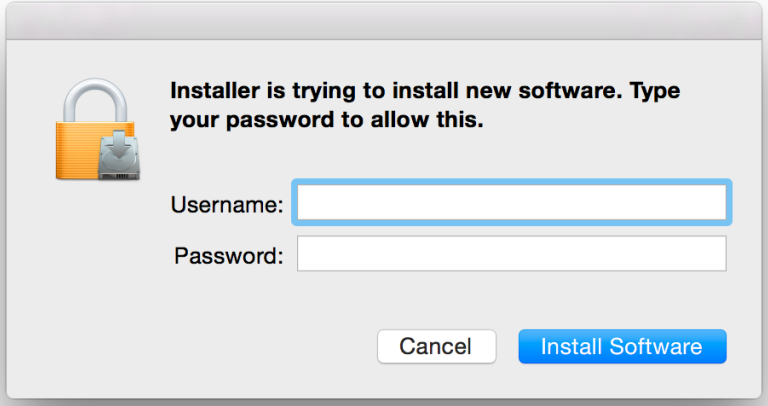
Şekil 12. *Python kurulumu: 3.Adım.*



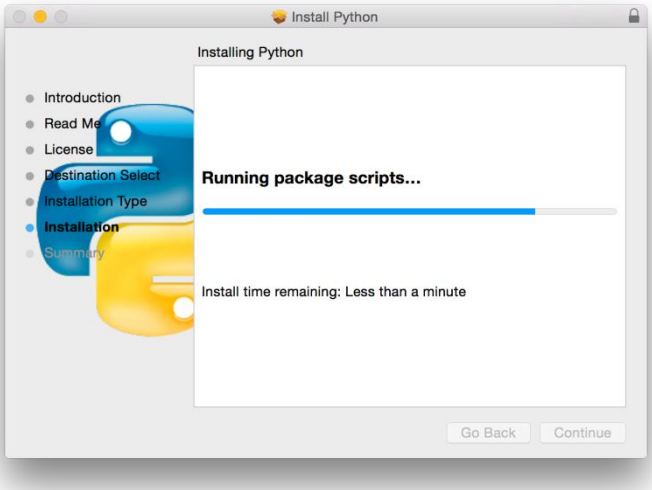
Şekil 13. *Python kurulumu: 4.Adım.*



Şekil 14. *Python kurulumu: 5.Adım.*



Şekil 15. *Python kurulumu: 6.Adım.*



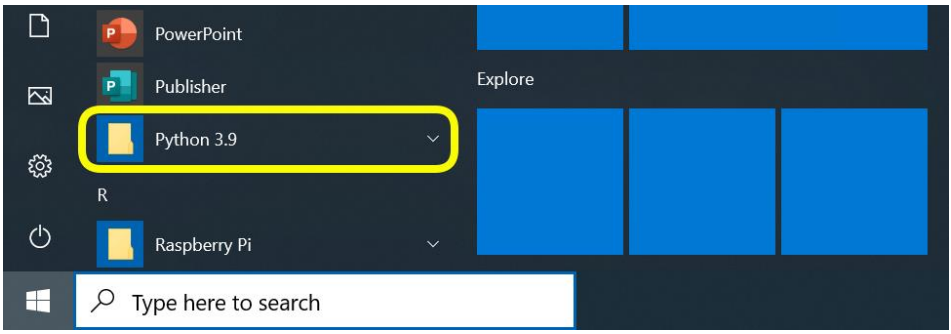
Şekil 16. *Python kurulumu: 7.Adım.*

Python IDLE

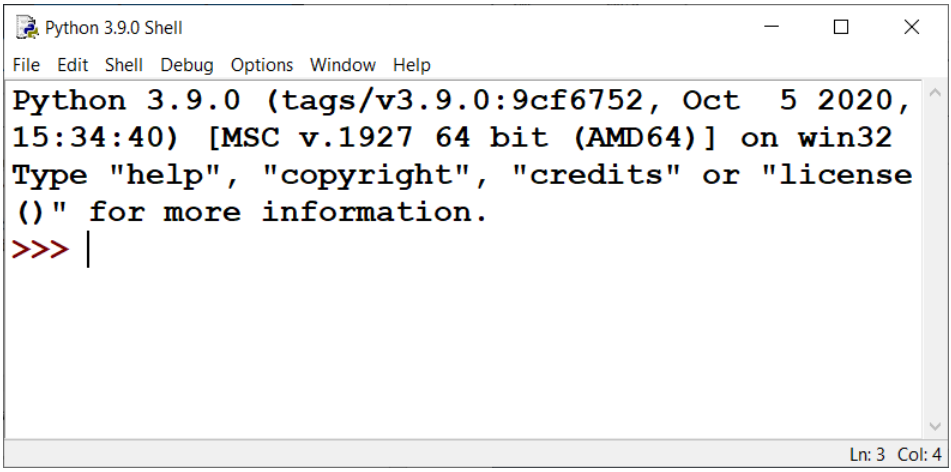
Bir önceki başlık altında Python IDLE'ın kurulumu gösterildi. Bu başlıkta kurulum tamamlandıktan sonraki ilk çalıştırma, Microsoft Windows ve macOS işletim sistemleri için, gösterilecektir.

Microsoft Windows İçin IDLE

Başarılı bir yüklemeden sonra programlar listesinde "P" harfine gidilerek Python klasörüne ve oradan da Python IDLE programına erişilebilir. Python klasörüne tıklandığında IDLE uygulaması görülecektedir. Ayrıca bu uygulamaya Windows'un arama araçlarıyla da ulaşılabilir. Arama satırına "Python" veya "idle" yazılırsa arama sonuçlarında IDLE'a rastlamak mümkün olur. IDLE uygulaması çalıştırıldığında **Şekil 18**'deki "Python Shell" penceresi ile karşılaşılır. Shell penceresi Python kodlarının derlenip çalıştırıldığı ve geri dönüşlerin alındığı penceredir. Bu pencerede her ne kadar kod yazmak mümkün olsa da programcılar kod yazımı için bu pencereyi kullanmazlar. Bu pencere belki sadece kısa kod parçalarını hızlıca denemek için kullanışlı olabilir.

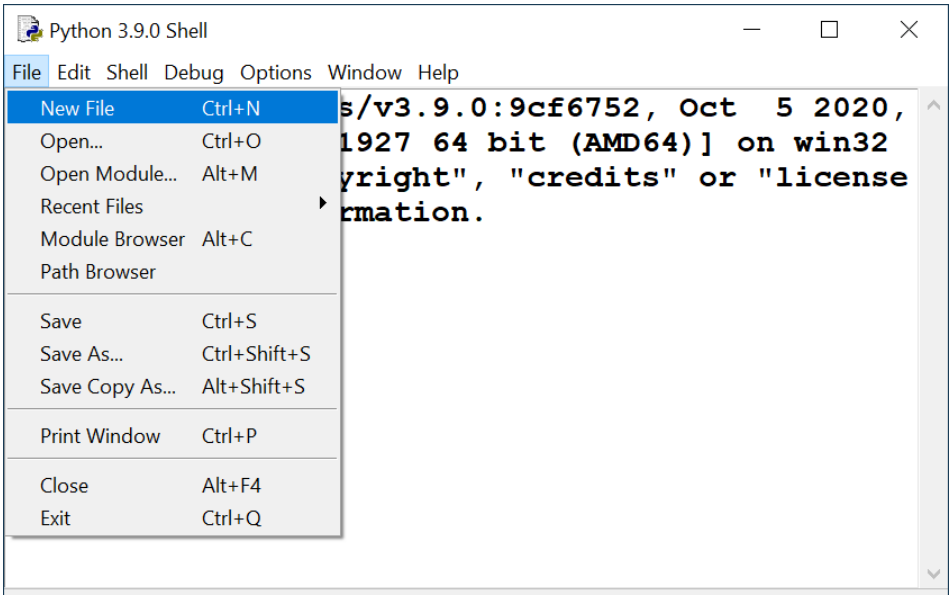


Şekil 17. *Windows programlar listesinde Python klasörünün gösterimi.*

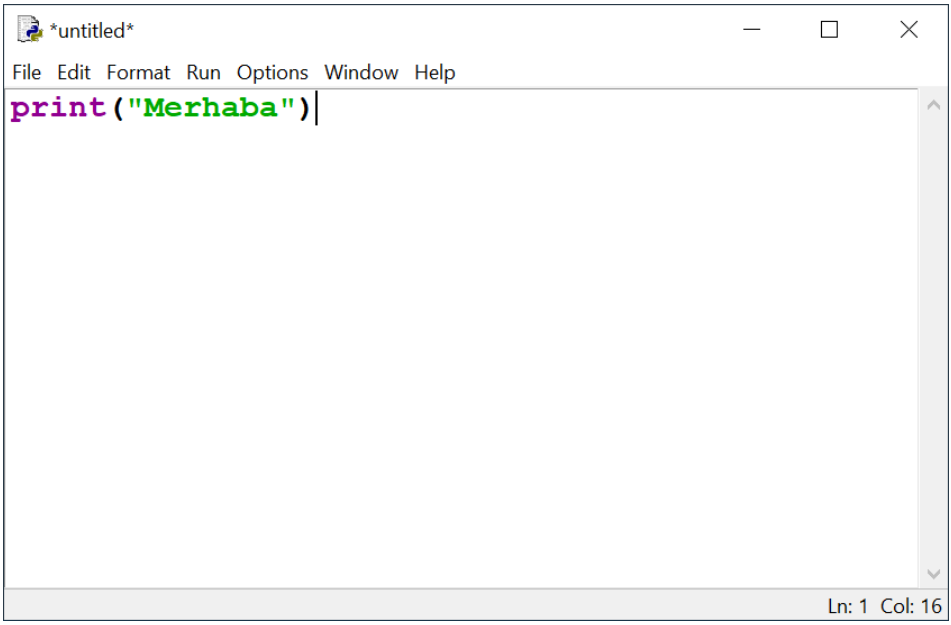


Şekil 18. *Windows'ta Python Shell.*

Python kodlarını yazmak için bu pencerenin (**Şekil 18**) üst kısmında yer alan "File" menüsünden "New File" seçeneğine tıklanır (**Şekil 19**).



Şekil 19. *Windows'ta Python Shell'den yeni kod dosyası oluşturma.*

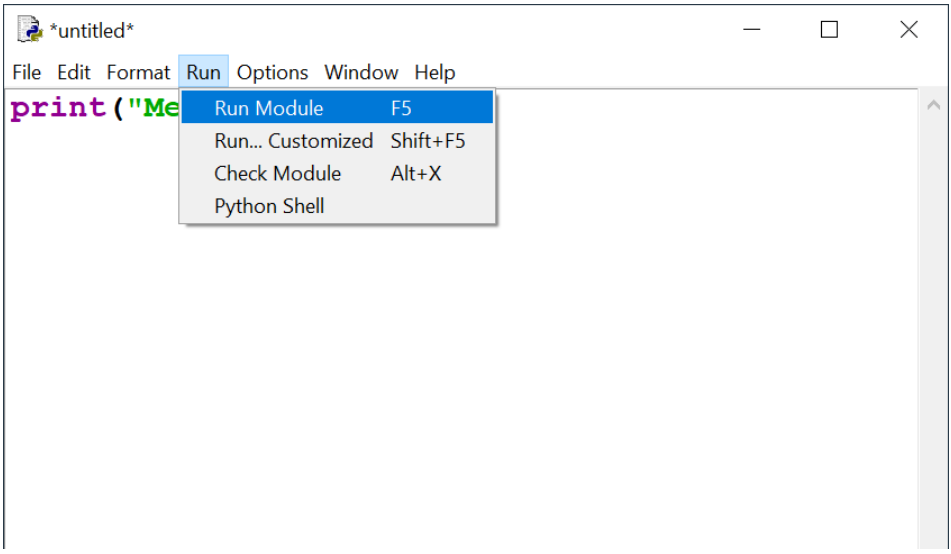


Şekil 20. *Windows'ta Python IDLE metin (kod) editörü.*

Böylece IDLE programının metin editörü (**Şekil 20**) açılmış olur. Python kodları bu pencerede yazılır. Fare imlecinin pozisyonu satır ve sütun olarak pencerenin sağ alt köşesinde gösterilir. **Şekil 20**'teki örnekte imlecin birinci satır ve 16. sütunda olduğu görülmektedir.

Yazılan programı çalıştırmak için yine pencerenin üst kısmındaki IDLE'ın "Run" menüsünden "Run Module" seçeneğine (**Şekil 21**) tıklanır. Yazılan kod metni diske kaydedilmemiş ise derleme ve çalıştırmaya geçmeden önce metni diske kaydetme işleminin yapılmasını bilgisayar kullanıcıdan ister. Bunun için açılan pencereden metin dosyası için bir konum seçilmesi ve dosya için bir isim girilmesi gerekir. Dosya kaydedildiğinde editör penceresinin ismi daha önce "untitled" iken değişir ve dosya yolu ve dosya ismini alır. Yani pencere başlığına bakıldığında hangi dosya ile

çalışıldığı görülebilir. Kaydetme işleminin tamamlanmasının ardından yazılan kod bilgisayar tarafından çalıştırılır.

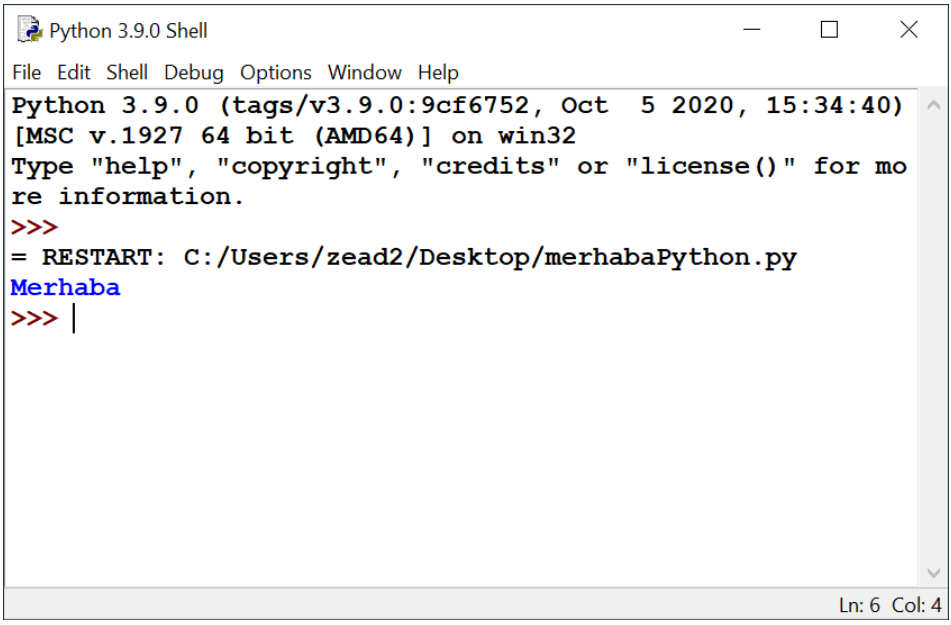


Şekil 21. *Python IDLE metin editöründe yazılan kodun çalıştırılması.*

Şekil 22'da görüldüğü gibi yazılan kod; derlenmiş, çalıştırılmış ve "Merhaba" çıktısı Shell penceresinde bir çıktı olarak alınmıştır. Çalıştırma işlemleri için kısayol tuşu olan F5'i kullanmak daha pratiktir. Aynı kod dosyası üzerinde bazı değişiklikler yapılır ve tekrar çalıştırılmak üzere F5'e basılırsa bu kez sadece daha önce oluşturulan dosyanın üzerine kayıt yapılmasının teyidi alınır ve yazılan kod çalıştırılır.

Bir kod editörünün satır numaralarını göstermesi her zaman tercih edilir. Çünkü Shell'den gelen hata bildirimlerinde satır bilgisi verilir. Yani kod içindeki hatanın ilk görüldüğü satır hangisi ise o satır numarası Shell'de gösterilir. IDLE'ın ön tanımlı (*default*) ayarlarında satır numaralarının

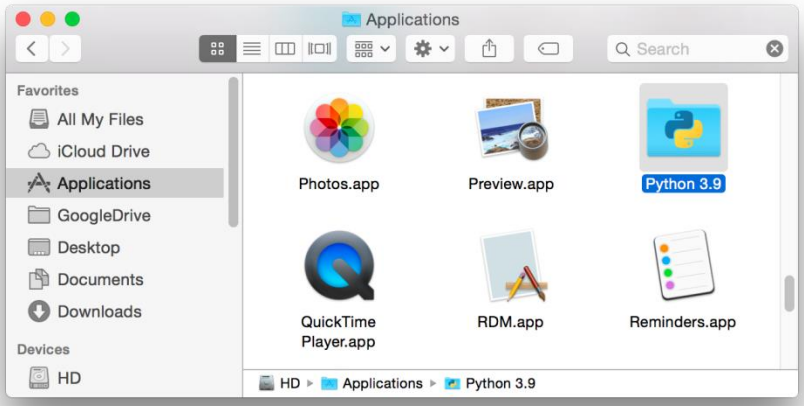
gösterimi kapalıdır. Açmak için menü satırında “Options” menüsündeki “Show Line Numbers” seçeneği seçilmelidir.



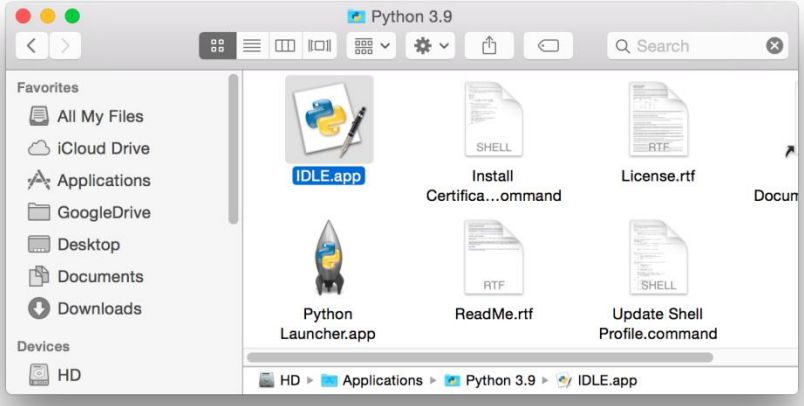
Şekil 22. *Çalıştırılan kodun Python Shell’deki çıktısı.*

macOS İçin IDLE

Python başarılı olarak bilgisayara yüklenmiş ise “Uygulamalar” klasöründe aşağıda gösterildiği gibi bir Python klasörü görülür.

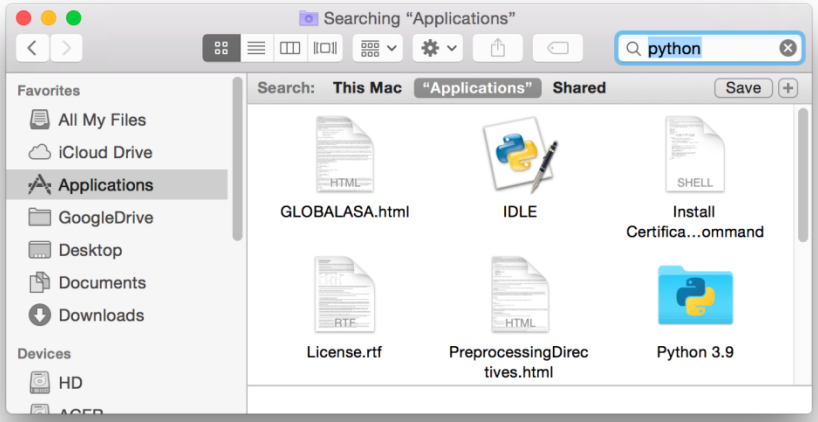


Şekil 23. *Python klasörünün macOS’ta gösterimi.*



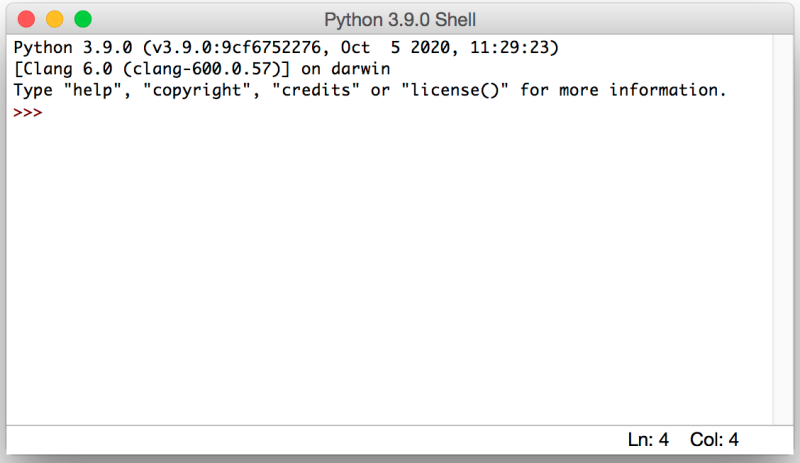
Şekil 24. *IDLE programının macOS’taki konumu.*

Python klasörü açıldığında IDLE uygulaması görülecektedir (Şekil 24). Ayrıca bu uygulamaya Mac'in arama araçlarıyla da ulaşılabilir (Şekil 25). Arama satırına "Python" yazılırsa arama sonuçlarında IDLE'a rastlamak mümkün olur.



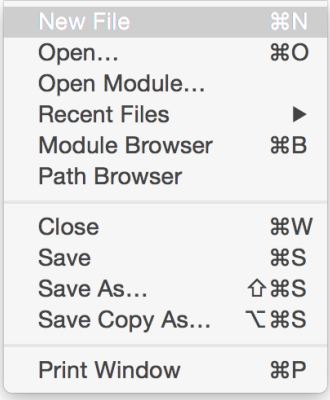
Şekil 25. *IDLE programının macOS'ta arama araçlarıyla bulunması.*

IDLE uygulaması çalıştırıldığında "Python Shell" penceresi ile karşılaşılır. Şekil 26'da IDLE programı çalıştırıldığında ekrana gelen pencerenin görünümü verilmiştir. Shell penceresi Python kodlarının derlenip çalıştırıldığı ve geri dönüşlerin alındığı penceredir.



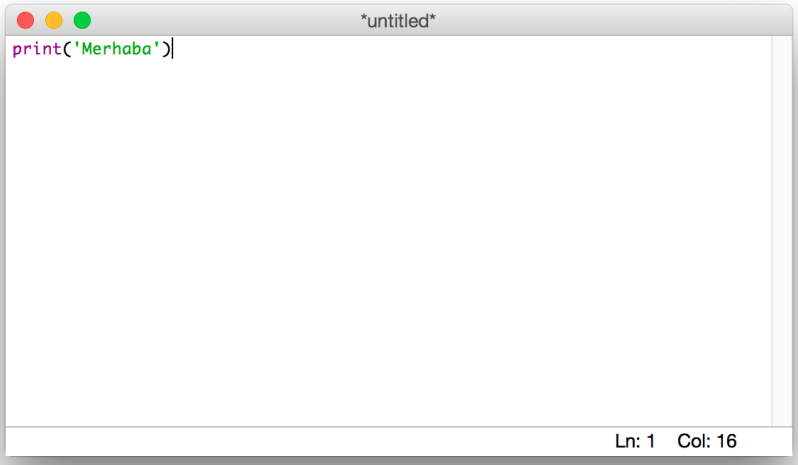
Şekil 26. *macOS'ta Python Shell.*

Bu pencerede her ne kadar kod yazmak mümkün olsa da programcılar kod yazımı için bu pencereyi kullanmazlar. Bu pencere belki sadece kısa kod parçalarını hızlıca denemek için kullanışlı olabilir.



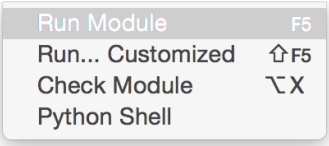
Şekil 27. *macOS'ta Python Shell'den yeni kod dosyası oluşturma.*

Python kodlarını yazmak için bu pencerenin en üst satırında yer alan “File” menüsünden “New File” seçeneğine tıklanır (**Şekil 27**). Böylece IDLE programının metin editörü (**Şekil 28**) açılmış olur. Python kodları bu pencerede yazılır.



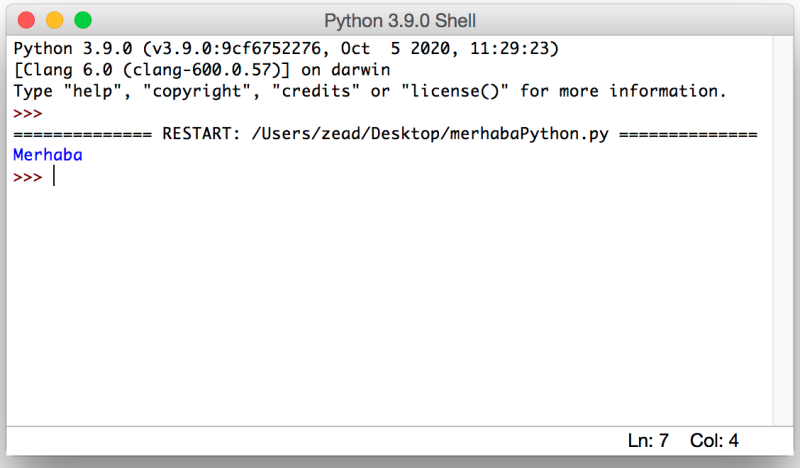
Şekil 28. *macOS’ta Python IDLE metin (kod) editörü.*

Fare imlecinin pozisyonu satır ve sütun numarası olarak pencerenin sağ alt köşesinde gösterilir. **Şekil 28**’deki örnekte imlecin birinci satır ve 16. sütunda olduğu görülmektedir.



Şekil 29. *Python IDLE metin editöründe yazılan kodun çalıştırılması.*

Yazılan programı çalıştırmak için yine ekranın en üst satırındaki IDLE'ın "Run" menüsünden "Run Module" seçeneğine (**Şekil 29**) tıklanır. Yazılan kod metni diske kaydedilmemiş ise derleme ve çalıştırmaya geçmeden önce metni diske kaydetme işleminin yapılmasını bilgisayar kullanıcıdan ister. Bunun için açılan pencereden metin dosyası için bir konum seçilmesi ve dosya için bir isim girilmesi gerekir. Dosya kaydedildiğinde editör penceresinin ismi daha önce "untitled" iken değişir ve dosya yolu ve dosya ismini alır. Yani pencere başlığına bakıldığında hangi dosya ile çalışıldığı görülebilir. Kaydetme işleminin tamamlanmasının ardından yazılan kod bilgisayar tarafından çalıştırılır.



Şekil 30. *Çalıştırılan kodun Python Shell'deki çıktısı.*

Şekil 30'da görüleceği gibi yazılan kod; derlenmiş, çalıştırılmış ve "Merhaba" çıktısı Shell penceresinde bir çıktı olarak alınmıştır. Çalıştırma işlemleri için kısayol tuşu olan F5'i kullanmak daha pratiktir.

Aynı kod dosyası üzerinde bazı değişiklikler yapılır ve tekrar çalıştırılmak üzere F5'e basılırsa bu kez sadece daha önce oluşturulan dosyanın üzerine kayıt yapılmasının teyidi alınır ve yazılan kod çalıştırılır.

Bir kod editörünün satır numaralarını göstermesi her zaman tercih edilir. Çünkü Shell'den gelen hata bildirimlerinde satır bilgisi verilir. Yani kod içindeki hatanın ilk görüldüğü satır hangisi ise o satır numarası Shell'de gösterilir. IDLE'ın ön tanımlı (*default*) ayarlarında satır numaralarının gösterimi kapalıdır. Açmak için menü satırında "Options" menüsündeki "Show Line Numbers" seçeneği seçilmelidir.

1. SENTAKS

Sentaks, söz dizimi anlamına gelen Fransızca kökenli bir kelimedir. Konuşma dillerinde nasıl ki dile ait yapılar ve kaideler varsa programlama dillerinde de aynı durum geçerlidir. Bu bölümde Python programlama diline ait sentaks bilgisi verilecektir.

1.1 İlk Kod

İlk kodlamayı yapmak için IDLE[2] programını açıyoruz ve "File" menüsü altındaki "New File" seçeneğine tıklıyoruz. Karşımıza çıkan yeni pencere bizim kodlama yapacağımız penceredir. Kod yazmadan önce bu pencerenin de "File" menüsüne gidip "Save" seçeneğine tıklıyoruz. Bu seçenek, yeni oluşturduğumuz boş Python metin belgesinin bilgisayara kaydedilmesini sağlamaktadır. "Save" e tıkladıktan sonra bilgisayar, bizden belge için bir isim ve bilgisayarda bir konum belirlememizi ister. Bunları yaptığımızda ".py" uzantısına sahip bir Python metin belgesi oluşturmuş oluruz. Artık o boş belge içine kodlama yapabiliriz.

[2] IDLE hakkında detaylı bilgi kitabın giriş bölümünde verilmiştir.

print()

Python diline ilk adımı, **`print()`** fonksiyonu ile atıyoruz. Bu fonksiyon; metin, sayı veya diğer yazdırılabilir bilgilerin shell'e[3] çıktısını almak için kullanılır. Bir başka ifade ile bilgisayara konuşmasını söylemek için **`print()`** fonksiyonu kullanılır. Parantezler arasına yerleştirilen veriyi bu fonksiyon ekrana yazdırır. Birden fazla veri aynı parantez içinde yazdırılabilir. Birden fazla veri olacaksa aralarına virgül konulması gerekir. Fonksiyon, her birini bir boşlukla ayrılmış olarak shell'e çıkarır. Hiçbir veri sağlanmadıysa, **`print()`** fonksiyonu boş bir satır çıkarır. Yazdırılacak veri, bir metin (*string*) ise tırnak içinde parantezler arasına yazılmalıdır. Yazılan kodu çalıştırmak ve ekranda çıktıyı görmek için IDLE programını kullanıyorsak F5'e basabiliriz.

```
1  print("bir not")
2  print()
3  print("1","2")
4  print(1,2,3)
```

```
>>>

bir not

1 2
1 2 3
```

Bu kitaptaki örnek kodlamalar yukarıdaki gibi bir tablo yapısında verilecektir. Yazılan kodlar tablonun üst kısmında yer almakta, shell'den

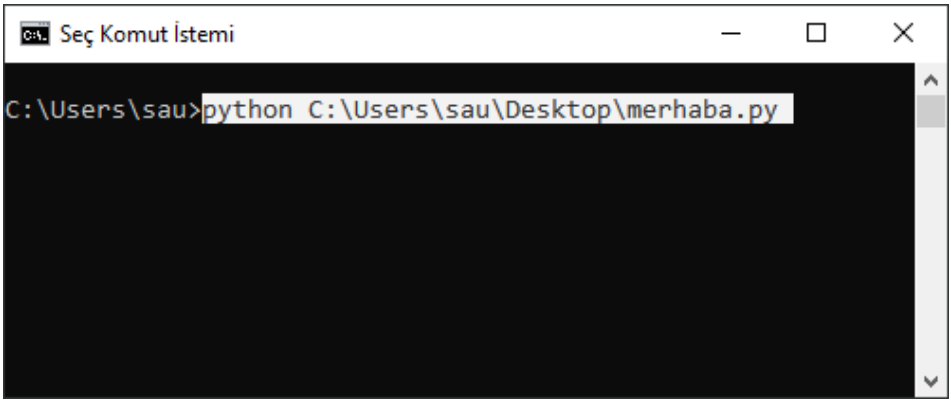
[3] Shell'e, konsola veya ekrana çıktı almak veya yazdırmak dediğimizde aslında hep aynı şeyi kastetmiş olacağız. Bu kasıt ise bir değeri veya veriyi, print() komutunu kullanarak programın çalıştığı ve geri bildirimlerin alındığı pencereye göndermektir.

elde edilen çıktılar ve geri bildirimler de tablonun alt kısmında yer almaktadır. Tablolardaki renklendirmeler, IDLE programının varsayılan renk şeması ile aynıdır.

Yazılmış bir kodu çalıştırmanın başka yolları da vardır. Python kodları IDLE haricinde Windows işletim sisteminin yerleşik uygulaması olan "Komut İstemi" ve onun macOS'taki muadili olan "Terminal" ile de çalıştırılabilmektedir. Elbette bunun için Python'un bilgisayarınıza giriş bölümünde anlatıldığı gibi yüklenmiş olması gerekir.

Windows Komut İstemi

"Komut İstemi" ile örneğin *merhaba.py* isimli belgeyi çalıştırmak istersek, **`python`** yazıp bir boşluk bırakıp *merhaba.py* belgesinin konumunu ve devamında belgemizin ismini uzantısıyla birlikte yazmamız gerekir. Bir belgenin konumunu görmek için üzerine sağ tıklayıp "Özellikler" bölümüne bakmamız yeterlidir. Orada konum bilgisi görülecektir. Ancak konum bilgisi uzun bir satır olduğu için yazımı zordur. Eğer Python belgesi sürüklenip Komut İstemi penceresine bırakılırsa belgenin konum bilgisi otomatik olarak girilmiş olur. Belgeyi çalıştırmadan önce belgenin son halini "File" > "Save" seçeneği ile kaydettiğimizden emin olmalıyız. Aşağıda örnek bir çalıştırma gösterilmiştir.



Şekil 31. *Windows Komut İstemi'nde Python belgesinin çalıştırılması.*

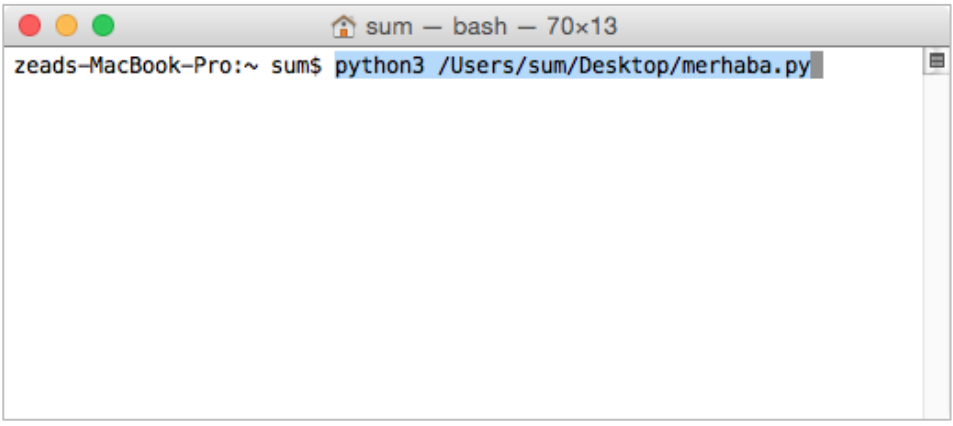
Eğer Python yüklemesini önceki sayfalarda gösterildiği gibi yaptıysanız, ".py" uzantılı Python belgelerine çift tıkladığınızda doğrudan Komut İstemi üzerinde çalıştırılırlar. Eğer programınızda bir bekleyişi gerektiren işlem yoksa Komut İstemi penceresi anlık açılır, program çalıştırılır ve pencere kapanır. Bu işlem basit kodlar için bir saniye bile sürmeyebilir. Dolayısı ile "acaba yazdığım program çalışmadı mı" gibi bir kuşkuya düşmenize gerek yoktur. Kodlarınızı bu şekilde çalıştırdığınızda (yani çift tıklama ile) hata bildirimlerini göremezsiniz. Bu tür bir çalıştırma, son hali verilmiş kod belgeleri için uygun olabilir.

Mac Terminal

Mac sistemlerinde komutlar Terminal adı verilen bir uygulama ile çalıştırılır. Bu uygulama sistemde yerleşik bir uygulamadır. Uygulamalar klasöründe yer almaktadır. Eğer giriş bölümünde gösterildiği gibi bir **`Python`** yüklemesi yaptıysanız, Terminal uygulamasında ".py" uzantılı **`Python`** metin belgeleri **Şekil 32**'de gösterildiği gibi çalıştırılabilir. Önce Terminal'e **`python3`** yazılır ve bir boşluk bırakılır.

Sonra ".py" uzantılı Python belgesi mouse ile tutulup Terminal penceresine sürüklenir ve bırakılır. Bırakma ile birlikte Python belgesinin dosya yolu pencerede görülür. Artık bu belge "enter" tuşuna basılarak çalıştırılabilir. Belgeyi çalıştırmadan önce belgenin son halini "File" > "Save" seçeneği ile kaydettiğimizden emin olmalıyız.

macOS'ta ".py" uzantılı bir Python belgesine çift tıklama yapıldığında Python IDLE programı ile belgenin açıldığı görülür.[4]



Şekil 32. *Mac Terminal'de Python belgesinin çalıştırılması.*

1.2 Yorumlar

Bir programda yazılan ancak bilgisayar tarafından çalıştırılmayan metne yorum denir. Python, **#**'dan sonraki her şeyi satırın sonuna kadar yorum olarak algılar. Yorum satırları IDLE'da kırmızı renkte gösterilir. Yorumların kullanılış amaçları aşağıdaki gibidir.

[4] Eğer Mac bilgisayarınızda Xcode programı varsa ".py" uzantılı Python kod belgeleri Xcode ile açılabilir. Bunu bir Python belgesine sağ tıklayıp ilgili menüye giderek düzeltebilirsiniz.

1. Bir şeyin ne amaçla yazıldığına ilişkin bağlam bilgisi verir. Bu aynı zamanda diğer kişilerin kodu daha hızlı anlamasına yardımcı olur.

```
# Bu değişken,'aynen' kelimesini saymak için
# kullanılacaktır.
aynen_sayacı = 0
```

```
# Bu kod, yarın yağmur yağma olasılığını
# hesaplar
yarın_yağmur()
```

2. Bir kod satırını yok saymaya ve o olmadan bir programın nasıl çalışacağını görmeye yarar.

```
# yarın_yağmur() = eski_işlem()
yarın_yağmur() = yeni_işlem()
```

1.3 Değişkenler

Program tarafından kullanılacak verileri saklamak için değişkenler kullanılır. Bu veriler sayı, yazı karakteri veya başka bir veri türü olabilir. Bir değişkenin türünü taşıdığı verinin türü belirler. Değişken isimleri tanımlanırken harfler, sayılar ve alt çizgi karakteri (_) kullanılabilir. Sayılar değişken isimlerinin ilk karakteri olarak kullanılamazlar. Türkçe karakterlerin (ı,ö,ç,ş,ğ) değişken isimlerinde yer alması Python'da problem çıkarmaz. Ancak bir metin editöründe yazılmış ve Türkçe karakterler içeren Python kodu bir başka metin editöründe açıldığında karakterlerin görüntülenmesinde ve ilgili kodun çalıştırılmasında problem

yaşanabilmektedir. Bu problem, kodun yazıldığı metin editörünün karakter kodlamada kullandığı sistem ile diğer metin editörünün kullandığı karakter çözümleme sisteminin uyuşmaması ile ilgilidir.

Eşittir işareti (=) bir değişkene değer atamak için kullanılır. İlk atama yapıldıktan sonra, bir değişkenin değeri gerektiğinde yeni değerlere güncellenebilir. Aşağıda değişken isimlendirmesine ve bazı değişken türlerine örnekler verilmiştir.

```
# Farklı türlerde değişkenler.
# Sırasıyla yazı karakteri, sayı ve boolean

kullanıcı_adı = "@adem"
kullanıcıKodu = 101
_onay1 = False

# Bir değişkenin değeri tanımlandıktan
# sonra değiştirilebilir.

skor1 = 100
skor2 = 120
skor2 = 140
print(skor2)
```

```
>>>

140
```

Python'da büyük-küçük harf duyarlılığı vardır. Yani A, a'ya eşit değildir. Aşağıda **a** değişkeni bulunamadığı için hata alınmıştır.

```
1  A = 50
2  print (a)
```

```
>>>

File <dosyaYolu>, line 2, in <module>
    print (a)
NameError: name 'a' is not defined
```

1.4 Veri Türleri

Python'da değişkenlere atanabilecek farklı veri türleri vardır. Veri türü bazen değişken türü anlamında da anlaşılabilir. Yani veri türü ile değişken türü birbirinin yerine kullanılabilir. Ancak aslolan veri türüdür. Çünkü bir değişkenin türünü, taşıdığı verinin türü belirler.

Python'da desteklenen bazı önemli veri veya değişken türleri aşağıdaki gibi sınıflandırılabilir.

1. Yazı karakteri : **`str`**
2. Sayı : **`int, float`**
3. Boolean : **`bool`**
4. Koleksiyon : **`list, tuple, set, dict`**

`str` kodu *string* ifadesinin kısaltılmışıdır ve karakter dizisi anlamına gelmektedir. **`int`** kodu *integer* ifadesinin kısaltılmışıdır ve tam sayı anlamına gelmektedir. **`float`** kodu, *floating point number* ifadesini simgeler ve ondalık sayı anlamına gelmektedir. Birden fazla değeri koleksiyon halinde saklayan değişken türlerinden olan **`list`**, liste;

tuple, demet; **set**, küme ve **dict** sözlük (*dictionary*) anlamlarına gelmektedir.

1.4.1 String

Stringler, tırnak işaretleri arasında yer alan belli sayıda karakterden (harfler, sayılar, boşluklar veya noktalama işaretleri) oluşur. String türündeki verilerin çift tırnak işareti ("_") veya tek tırnak işareti ('_') arasında kullanılması gerekir.

```
1  isim = "Adem"
2  soyisim = 'Zengin'
3  print(isim)
4  print(soyisim)
```

```
>>>
Adem
Zengin
```

1.4.2 Sayılar

Sayı sınıfının önemli alt türleri **int** ve **float** türleridir. **int**, kesirli kısım olmadan yazılabilen (ondalık kısmı olmayan) tam bir sayıdır. Bir tam sayı, ondalık kısım olmadığı sürece pozitif bir sayı, negatif bir sayı veya 0 sayısı olabilir. 0 sayısı bir tamsayı değerini temsil eder, ancak 0.0 ile yazılan aynı sayı bir **float** (ondalık) sayıyı temsil eder.

```
# Örnek tam(integer)sayılar

kitaplar = 4
defterler = 1
satılanlar = -2
tamirEdilen = 0

# Integer olmayan sayılar

elmaKg = 2.5
eldeKalan = 0.0
```

Python'da desteklenen sayı türlerinden biri de ondalıklı sayıları temsil eden **`float`** türüdür. **`float`** sayı, ondalık kısım içeren bir sayı değeridir. Kesirli büyüklüklere sahip sayıları temsil etmek için kullanılır. Örneğin, **`a=2/5`** bir tamsayı olarak temsil edilemez, bu nedenle **`a`** değişkenine **`0.4`** değerinde bir **`float`** değeri atanır.

```
# Ondalıklı sayılar
a = 0.4
pi = 3.14
biletParasi = 89.99
dolulukYüzde = 0.4
```

1.4.3 Boolean

Boolean, Python'da bir değişken (veri) türüdür. **`True`** ve **`False`** olmak üzere sadece iki değer alır. Integer, string ve float veri türlerinde olduğu gibi boolean değerleri de bir değişkene atanabilir. Bir ifadenin, kıyaslamanın veya denetimin doğru mu yanlış mı olduğunu kodlamak

için bu veri türü kullanılır. Kitabın “Kıyas ve Denetim” bölümünde bu konuda detay verilmiştir.

```
1  a = True
2  b = False
3
4  print(a)
```

```
>>>

True
```

1.4.4 Koleksiyonlar

Bir integer değişken tek bir değeri taşır. Aynı şekilde bir float veya boolean değişken de tek bir değeri taşır. Bir değişkenin birden fazla değer taşıması istendiğinde koleksiyon türlerinden biri kullanılmalıdır. Python’da **`list`** (liste), **`tuple`** (demet), **`set`** (küme) ve **`dict`** (sözlük) isimlerine sahip koleksiyon türleri vardır. Bu türler bazı koleksiyon özellikleri bakımından birbirinden farklılaşırlar. Koleksiyonlar, kitabın müstakil bir bölümünde detaylıca incelenmiştir. Burada sadece çalışma ortamına yazım (sentaks) olarak nasıl tanımlandıkları gösterilecektir.

```
# list
birListe = ["elma", "armut", "kiraz"]

# tuple
birDemet = ("elma", "armut", "kiraz")

# set
birKüme = {"elma", "armut", "kiraz"}

# dict
birSözlük = {"tür" : "elma", "kg" : 32}
```

1.5 Tür Sorgulama

Python'da **`type()`** fonksiyonu bir değişkenin türünün programatik olarak sorgulanmasını sağlar.

```
a = 3.14
b = {"elma", "armut", "kiraz"}
c = ("elma", "armut", "kiraz")

print(type(a))
print(type(b))
print(type(c))
```

```
>>>

<class 'float'>
<class 'set'>
<class 'tuple'>
```

1.6 Tür Dönüştürme

Python'da çeşitli sebeplerle bir türdeki değer başka bir türe dönüştürülmektedir. Örneğin **`int`** türündeki bir değer, **`float`** türüne veya **`str`** türüne dönüştürülebilmektedir. Bir değeri, **`str`** türüne dönüştürmek için **`str()`**, **`int`** türüne dönüştürmek için **`int()`** ve **`float`** türüne dönüştürmek için **`float()`** fonksiyonları kullanılmaktadır. **`float`** türünde bir rakam **`int`** türüne dönüştürülürse ondalıklı kısmı atılır. Örneğin 3.9 sayısı 3 olarak **`int`** türüne dönüşür. **`int`** türünde bir rakam, **`float`** türüne dönüşürse, yanına ondalık olarak sıfır (.0) alır. Bir **`str`** değerini **`float`** türüne dönüştürmek için o **`str`** değerinin sayılardan oluşması gerekir. Yine bir **`str`** değerini **`int`** türüne dönüştürmek için o **`str`** değerinin bir tam sayıyı ifade ediyor olması gerekir. Aksi hallerde hata alınır. Aşağıda bu dönüşümler için kullanılan fonksiyonların kullanımına dair örnekler verilmiştir.

```python
# Int-Str Dönüşümü
# Örnek Sebep:int ve str doğrudan eklemlenemez.
a = 3
b = str(a)
print(b+" TL")

# Str-Float Dönüşümü
# Örnek Sebep:str ile doğrudan matematik işlemi
# yapılmaz.
c = "2"
d = float(c)
print(d+3)

# Float-Int Dönüşümü
# Örnek Sebep:Aşağı yuvarlama ile tam sayı
# eldesi
e = 3.99
f = int(e)
print(f)
```

```
>>>

3 TL
5.0
3
```

1.7 Aritmetik İşlemler

Python; sayılar, değişkenler veya bunların farklı kombinasyonları ile gerçekleştirilen aritmetik işlemleri destekler. Birincil aritmetik operatörler şunlardır:

+ : toplama

- : çıkarma

***** : çarpma

/ : bölme

% : modül (kalanı döndürür)

****** : üs alma

```
# Aritmetik işlemler

sonuç1 = 4 + 6
sonuç2 = 5 - 2
sonuç3 = 6 * 5
sonuç4 = 80 / 8
sonuç5 = 30 % 4
sonuç6 = 5 ** 2

print (sonuç1)
print (sonuç2)
print (sonuç3)
print (sonuç4)
print (sonuç5)
print (sonuç6)
```

```
>>>

10
3
30
10.0
2
25
```

Modül hesaplaması, pay ve payda arasındaki bölümün kalanını verir. Örneğin 10%2 ifadesinin sonucu 0 olur. Çünkü 10 sayısı 2'ye tam

bölünür. 10%3 ifadesinin sonucu 1 olur. Çünkü 10 sayısı, 3'e tam bölünemez ve geriye 1 kalır.

1.8 String Eklemleme

Python, artı (+) operatörünü kullanarak stringlerin birleştirilmesini destekler. Artı (+) operatörü ayrıca matematiksel toplama işlemleri için de kullanılır. Birden çok string değişkeni artı (+) operatörü kullanılarak birleştirilebilir. Artı (+) operatörüne iletilen değerlerin tamamı string türünde ise birleştirme gerçekleştirilir. Farklı tür değerler iletilirse, bu durumda Python bir hata durumu bildirir.

```
1  # String eklemleme
2
3  birinci = "Hız Sınırı "
4  ikinci = "80"
5
6  sonuç1 = birinci + ikinci
7  sonuç2 = birinci + ikinci + "!"
8
9  print(sonuç1)
10 print(sonuç2)
11 print(birinci + ikinci + 10)
```

```
>>>

Hız Sınırı 80
Hız Sınırı 80!
Traceback (most recent call last):
  File "C:/Users/Administrator/Desktop/idle.py", line
11, in <module>
    print(birinci + ikinci + 10)
TypeError: can only concatenate str (not "int") to
str
```

1.9 Artı-Eşittir Kullanımı (+=)

Artı-eşittir operatörü (+=), mevcut bir değişkene bir değer eklemek ve yeni değeri aynı değişkene geri atamak için uygun bir yol sunar. Değişken ve değerin string türünde olması durumunda, bu operatör toplama yapmak yerine string eklemleme işlemini gerçekleştirir. Bu kullanım dört işlemin tümü (**+=** , **-=** , ***=** , **/=**) için geçerlidir.

```
1  # Artı-Eşittir Operatorü
2  sayaç1 = 2
3  sayaç1 += 3
4
5  # Bu kod aşağıdaki koda denktir.
6  sayaç2 = 2
7  sayaç2 = sayaç2 + 3
8
9  # String eklemleme için kullanımı
10 mesaj = "Mesajın ilk kısmı "
11 mesaj += "ve mesajın kalan kısmı"
12
13 print(sayaç1)
14 print(sayaç2)
15 print(mesaj)
```

```
>>>

5
5
Mesajın ilk kısmı ve mesajın kalan kısmı
```

1.10 Kullanıcı Veri Girişi

Kullanıcıların shell'e veri girmesi, bazı uygulamalarda ihtiyaç duyulan bir durumdur. Bu işlem için Python'da **input** fonksiyonu kullanılır. Bu fonksiyon kullanıcıya hangi bilgiyi istediğini shell'de kısa bir metin ile gösterir ve klavyeden bu bilgiyi girmesini ister. Kullanıcı bilgi girişi yapıp onaylayana kadar programın akışı durdurulur.

```
1 kullanıcıAdı = input("Kullanıcı adı:")
2 şifre = input("Şifre:")
3 print("Hoşgeldin "+ kullanıcıAdı)
```

```
>>>

Kullanıcı adı:Bir isim
Şifre:Bir şifre
Hoşgeldin Bir isim
```

input fonksiyonu ile alınan veri türünün string olduğu unutulmamalıdır. Kullanıcıdan bir rakam girmesi isteniyor ve bu rakamla matematiksel bir işlem yapılmak isteniyorsa mutlaka uygun bir sayı türüne (**int** veya **float** gibi) dönüştürülmelidir.

```
a = input("Bir tam sayı giriniz:")
# a'nın integer türüne dönüştürülmesi
a = int(a)
a_kare = a*a
a_küp = a*a*a

# Burada tekrar string'e dönüşüm yapılacak.
# Çünkü print parantezi içindeki verilerin
# aynı türden olmaları gerekir.
print("Karesi:" + str(a_kare))
print("Küpü:" + str(a_küp))
```

```
>>>

Bir tam sayı giriniz:3
Karesi:9
Küpü:27
```

1.11 Hatalar (Errors)

Python derleyicisi, kodunuzda bulunan hataları bildirecektir. Çoğu hata durumu için derleyici, hatanın tespit edildiği kod satırını görüntüler ve hatanın tespit edildiği kod kısmının altına bir şapka karakteri ^ yerleştirir. Kod editörü olarak *Python Idle* kullananlar hatalı yerin kırmızıya boyandığını görebilirler.

```
1 sayı =
2 print(sayı)
```

```
>>>

  File "<stdin>", line 1
    sayi =
         ^
SyntaxError: invalid syntax
```

1.11.1 SyntaxError

Kodun bir kısmı yanlış olduğunda Python derleyicisi tarafından bir *SyntaxError* bildirilir. *SyntaxError* ile örneğin eksik veya fazla parantez, yanlış operatör kullanımı, eksik veya fazla tırnak işareti gibi sebeplerle karşılaşılmaktadır.

```
1 miktar = 7 + 5 = 4
```

```
>>>

  File "<stdin>", line 1
SyntaxError: cannot assign to operator
```

1.11.2 NameError

Bilinmeyen bir değişken algılandığında, Python derleyicisi tarafından bir *NameError* bildirilir. Bu hatanın alınacağı bazı durumlar şöyledir:

- Bir değişkene değer atanmadan önce o değişkenin kullanılması
- Bir değişkenin adının ilk tanımlandığı şeklinden farklı olarak yazılması

Python derleyicisi, *NameError*'un tespit edildiği kod satırını görüntüler ve hatanın hangi değişken ile ilişkili olduğunu gösterir.

```
1  print(gelenMiktar)
```

```
>>>

  File "<stdin>", line 1, in <module>
NameError: name 'gelenMiktar' is not defined
```

1.11.3 ZeroDivisionError

Bölme işlemi gerçekleştirildiğinde ve payda olarak 0 değeri algılandığında Python derleyicisi bunu hata olarak bildirir. Matematikte, bir sayıyı sıfıra bölmenin tanımlı bir değeri yoktur. Bu nedenle Python bunu bir hata koşulu olarak ele alır ve bir ZeroDivisionError rapor eder. Bölmenin gerçekleştiği kod satırını görüntüler. Bu durum, payda olarak bir değişken kullanıldığında ve değeri 0 olarak ayarlandığında veya değiştirildiğinde de olabilir.

```
1  pay = 1
2  payda = 0
3  kötüBölmeİşlemi = pay / payda
```

```
>>>

  File "<stdin>", line 3, in <module>
ZeroDivisionError: division by zero
```

SORULAR

1. **3 ve 7 rakamlarını toplayan kod yarım yazılmıştır. Alt çizgi (_) ile işaretli boşluklara yazılması gereken kodları yazınız.**

```
1 _ = _
2 b = 3
3 print (a _ b)
```

2. **Aşağıdaki adımları sırasıyla kodlayınız.**
   a. a ve b değişkenlerine sırasıyla 23 ve 47 değerlerini atayınız.
   b. c isimli değişkeni a ve b'nin toplamı olarak tanımlayınız.
   c. c'nin değerini yazdırınız.

3. **Aşağıdaki değişken isminde bulunan hatalı karakterleri çıkararak yeniden yazınız.**

```
1 1_saksı-çiçeği.10 = "10 Türk Lirası"
```

4. Aşağıdaki seçeneklerden hangisi A değişkenine 100 tamsayı değerini atamaktadır?

A. A = 100

B. A = "100"

C. a = 100

D. a = "100"

E. a = 100.0

5. KUŞ isimli değişkene SERÇE değeri verilmek isteniyor. Aşağıdaki kodlardan hangisi bunu sağlamaktadır?

A. KUŞ = "SERÇE"

B. SERÇE = "KUS"

C. KUŞ = "SERCE"

D. KUS = SERCE

E. SERCE == KUS

6. Python'da yorumlar hangi karakter kullanılarak yazılır?

A. Çift tırnak (" ")

B. Tek tırnak (' ')

C. #

D. =

E. %

7. Verilen Python kodundaki a değişkeninin türü nedir?

```
1 a = "2.5"
```

8. Verilen Python kodundaki a değişkeninin türü nedir?

```
1 a = ["kalem", "silgi", "defter"]
```

9. Verilen Python kodundaki a değişkeninin türü nedir?

```
1 a = ("kalem", "silgi", "defter")
```

10. Verilen Python kodundaki a değişkeninin türü nedir?

```
1 a = False
```

11. Verilen Python kodundaki a değişkeninin türü nedir?

```
1 a = {"tür" : "lale", "boy" : 36}
```

12. Aşağıda bir değer, başka bir türe dönüştürülmüştür. Dönüşüm sonunda elde edilen yeni değerin çıktısı nasıl olur?

```
1  a = 7.51
2  b = int(a)
3  print(b)
```

```
>>>

?
```

13. Verilen Python kodunda hangi satır hata verir?

```
1  a = int(3.49)
2  b = float("2")
3  c = float(2)
4  d = int("2.5")
5  e = str(3.57)
```

14. Bir küçük işletme sahibi, sattığı ürünler için indirim kampanyası düzenleyecektir. Fiyatlandırmalarda hammadde ücretine 50 TL işçilik ücreti eklemektedir. İndirimi hammadde ücreti üzerinden %20 oranında yapmayı planlamaktadır. İşletmeci, etiket fiyatını girip, indirimli fiyatı çıktı olarak alacağı bir programa ihtiyaç duymaktadır. Bu Programı yazınız

CEVAP ANAHTARI

1.

```
a = 7
b = 3
print (a + b)
```

2.

```
a = 23
b = 47
c = a + b
print (c)
```

3.

```
_saksıçiçeği10 = "10 Türk Lirası"
```

4. Doğru cevap “A” seçeneğidir.

5. Doğru cevap “A” seçeneğidir.

6. Doğru cevap “C” seçeneğidir.

7. str

8. list

9. tuple

10. boolean

11. dict

12.

```
1  a = 3.59
2  b = int(a)
3  print(b)
```

```
>>>

  File "<stdin>", line 3, in <module>
ZeroDivisionError: division by zero
```

13. 4.satır

14.

```
# Kullanıcıdan etiket fiyatı al.
etiket = input("Etiket fiyatı giriniz:")

# String türündeki fiyatı float türüne çevir.
etiket = float(etiket)

# İşçilik tutarını etiket fiyatından düş.
hammadde = etiket - 50

# %20 indirim uygula.
indirimli = hammadde - hammadde*0.2
# ya da indirimli = hammadde*0.8

# İşçilik tutarını ekle.
sonFiyat = indirimli + 50

# Yeni fiyatı ekrana yazdır.
print(sonFiyat)
```

2. FONKSİYONLAR

Bazı görevlerin bir program içinde birden çok kez gerçekleştirilmesi gerekir. Bu şekilde tekrar eden görevleri fonksiyonlar ile yerine getirmek, kodlamada hem yazım hem organizasyon açısından kolaylık sağlar. Bir fonksiyon yapısı çeşitli bileşenlerden meydana gelir. Fonksiyonlar **`def`** anahtar kelimesi ile tanımlanırlar. Değişkenlere isim verilirken uyulması gereken kurallar, fonksiyonlara isim verirken de geçerlidir. Fonksiyon tanımları, fonksiyona veri girişi sağlayan parametreleri de içerebilir. Parametreler fonksiyon isminden sonra gelen parantezler içine yazılırlar. Parantez kapatıldıktan sonra iki nokta koyulur ve alt satırda fonksiyonun görev tanımına geçilir. Fonksiyonlar, **`return`** anahtar kelimesi ile bir değer döndürebilir, dönüt verebilir. Fonksiyon yapılarının parametre ve **`return`** içermesi bir zorunluluk değildir. Aşağıda örnek bir fonksiyon yapısı gösterilmiştir.

```
# Parametresi x olan bir fonksiyon tanımla

def fonksiyon1(x):
    return x + 10
```

2.1 Fonksiyon Çağırma

Python, önceden tanımlanmış bir fonksiyonu çağırmak yani onu kullanmak için basit bir yazım kullanır. Bir fonksiyonu çağırmak için önce adı sonra da parantez içinde varsa parametreleri yazılır. Parametresi olmayan fonksiyonlar için ise içi boş parantez yazılır.

```
def fonksiyon1(x):
    return x + 10

# Önceden tanımlanmış bir fonksiyonu çağırma
print(fonksiyon1(3))
```

```
>>>

13
```

2.2 Fonksiyon Girintisi

Python, kod bloklarını tanımlamak için girinti kullanır. Aynı blok içindeki kod, aynı seviyede girintilenmelidir. Python fonksiyonu, bir tür kod bloğudur. Bir fonksiyon tanımının altındaki tüm kodun, fonksiyonun bir parçası olduğunu bilgisayara bildirmek için kodu girintili yazarız. Diğer programlama dillerinde kod bloklarının nerede başlayıp nerede bittiği parantez işaretleri ile belirlenir. Python, parantez sistemi yerine girintileme sistemi kullanır. Bir fonksiyon içinde, **`for`** ve **`if`** blokları

kullanıldığında bu bloklar için ilave girinti verilmelidir. Böylece hangi kodun hangi bloka ait olduğu bilgisayara bildirilmiş olur. Aşağıda farklı derinliklerde yapılan girintilemeler gösterilmiştir.

```
# Girintileme kod bloklarının sınırlarını
# belirlemek için kullanılır

def testFonksiyonu(sayı):
    # bu girintinin altı tamamen fonksiyona
    # aittir
    print("fonksiyon içine girildi")
    toplam = 0
    for x in range(sayı):
        # Burada bir kademe daha girinti
        # verildi. Çünkü 'for' döngüsü de ayrı
        # bir kod bloğudur.
        toplam += x
    return toplam
print("fonksiyon dışına çıkıldı")
```

2.3 Fonksiyon Parametreleri

Bazen fonksiyonlar, içerdiği kod bloğuna veri sağlamak için girdi gerektirir. Bu girdiler, parametreler kullanılarak tanımlanır.

Parametreler, fonksiyon tanımında tanımlanan değişkenlerdir. Parametrelerin değerleri fonksiyon çağrıldığı sırada atanır. Bir başka deyişle bir fonksiyon parametreleri ile birlikte çağrıldığında parametrelerin değerleri de atanmış olur.

Fonksiyonlar parametresiz tanımlanabildiği gibi, birden fazla sayıda parametreli olacak şekilde de tanımlanabilir. Birden çok parametresi olan

bir fonksiyonu tanımlamak için parametre isimleri parantez içinde virgülle ayrılmış olarak birbiri ardına yerleştirilir.

```
# Verilen iki sayının karelerinin toplamını
# veren fonksiyon

def kareToplam(sayı1,sayı2):
    toplam = (sayı1**2) + (sayı2**2)
    return (toplam)

print(kareToplam(2,3)) # Veya

çıktı = kareToplam(2,3)
print(çıktı)
```

```
>>>

13
13
```

2.4 Fonksiyon Argümanları

Python'daki parametreler birer değişkendir. Belirli isimler verilen bu değişkenlere farklı değerler atanarak fonksiyon çağrılabilir. İşte burada değişkenlere parametre denirken fonksiyon çağrılırken değişkenler için kullanılan değerlere de argüman denir. Yukarıdaki kodu ele aldığımızda **`sayı1`** ve **`sayı2`**, **`kareToplam`** fonksiyonunun parametreleridir. 2 ve 3 değerleri ise fonksiyon çağrılırken kullanılan argümanlardır. Fonksiyonlar çağrılırken kullanılması gereken argüman sayısına dikkat edilmelidir. Kullanılan argümanların bir eksik veya bir fazla olması hataya neden olur.

2.5 Fonksiyon Anahtar Argümanları

Python fonksiyonları, varsayılan değerlere sahip olabilen adlandırılmış argümanlar ile tanımlanabilir. Böyle tanımlanmış bir fonksiyon çağrılırken argüman alanına girilecek değerler argüman isimleriyle birlikte girilebilir. Bunun mümkün olması için fonksiyon tanımlanırken argümanlara isim vermek ve varsayılan değerlerini atamak gerekir. Bu tür argümanlara anahtar argümanlar denir.

Anahtar argüman kullanıldığında argümanlar belli sıraya göre parantez içine yazılmak zorunda değildir. Anahtar argümanlı olarak tanımlanmış bir fonksiyon çağrıldığında eğer argümanlardan biri eksik ise varsayılan değer kullanılır.

```
1  def hacimHesap(uznlk=1, genişlik=1, derinlik=1):
2      print("Uzunluk = " + str(uznlk))
3      print("Genişlik = " + str(genişlik))
4      print("Derinlik = " + str(derinlik))
5      print()
6      return uznlk * genişlik * derinlik
7
8  hacimHesap(1, 2, 3)
9  hacimHesap(uznlk=5, derinlik=2, genişlik=4)
10 hacimHesap(2, derinlik=3, genişlik=4)
```

```
>>>

Uzunluk = 1
Genişlik = 2
Derinlik = 3

Uzunluk = 5
Genişlik = 4
```

```
Derinlik = 2

Uzunluk = 2
Genişlik = 4
Derinlik = 3
```

2.6 Fonksiyondan Geri Dönüş Almak

Python fonksiyonundan bir geri dönüş yani bir cevap almak için **`return`** anahtar sözcüğü kullanılır. Bir fonksiyondan alınan dönüş, daha sonra programda kullanılabilecek bir değişkene atanabilir. Örnekteki **`şubatKaçGün`** fonksiyonu **`yıl`** parametresinin alacağı değere göre bir string döndürmektedir.

```
1  def şubatKaçGün(yıl):
2    if yıl % 4 == 0:
3      return "Şubat " + str(yıl) + "'de 29 gündür"
4    else:
5      return "Şubat " + str(yıl) + "'de 28 gündür"
6
7  kontrol_yılı = 2020
8  çıktı = şubatKaçGün(kontrol_yılı)
9  print(çıktı)
```

```
>>>

Şubat 2020'de 29 gündür
```

2.7 Birden Fazla Geri Dönüş Almak

Python fonksiyonları, tek bir **`return`** ifadesi kullanarak birden çok değer döndürebilir. Döndürülmesi gereken tüm değerler, **`return`** anahtar sözcüğünden sonra listelenir ve virgülle ayrılır. Örnekte **`karesiniAl`** fonksiyonu **`x_kare`**, **`y_kare`** ve **`z_kare`**'yi döndürür.

```
1  def karesiniAl(x, y, z):
2      x_kare = x * x
3      y_kare = y * y
4      z_kare = z * z
5      # 3 değişkenin değerini döndür:
6      return x_kare, y_kare, z_kare
7
8  sonuç1, sonuç2, sonuç3 = karesiniAl(3, 4, 5)
9
10 print(sonuç1)
11 print(sonuç2)
12 print(sonuç3)
```

```
>>>

9
16
25
```

2.8 Yerel & Global Değişkenler

Fonksiyonların dışında tanımlanmış değişkenlere global değişken, içinde tanımlanmış değişkenlere de yerel değişkenler denir. Global değişkenlere fonksiyon içinden erişmek mümkündür. Örnekte, **`a`** değişkeni global bir değişkendir. Çünkü **`ekranaYaz`** fonksiyonunun dışında tanımlanmıştır. Dolayısı ile **`a`**'nın değerini yazdıracak olan fonksiyon **`a`**'ya erişebilecektir.

```
# a global değişkeninin tanımlanması
a = "Merhaba"

def ekranaYaz():
    # Global değişkenin bir fonksiyon içinden
    # çağrılması
    print(a)

# Fonksiyonun çalıştırılarak
# a değişkenine erişimin test edilmesi
ekranaYaz()
```

```
>>>

Merhaba
```

Fonksiyon içinde tanımlanmış yerel değişken, fonksiyon dışından çağrılırsa “değişken tanımlanmadı” hatası alınır.

```
# Fonksiyon içinde yerel değişkenin tanımlanması
def yeniBir():
    a = 5

# Fonksiyon dışından yerel değişkenin çağrılması
print(a)
```

```
>>>
Traceback (most recent call last):
  File "C:\Users\Adem\Desktop\yerel.py", line 6, in
<module>
    print(a)
NameError: name 'a' is not defined
```

Fonksiyon içinde global bir değişken tanımlanmak isteniyorsa **global** anahtar kelimesi kullanılabilir. Bu durumda yukarıdaki hata alınmaz. Yani bir değişken eğer **global** anahtar kelimesi ile tanımlanmışsa fonksiyon dışından da çağrılabilir. Ancak unutulmamalıdır ki, bunun gerçekleşebilmesi için fonksiyonun en az bir kez çalıştırılması gerekir.

```
# Fonksiyon içinde global değişkenin
# tanımlanması
def yeniBir():
    global a
    a = 5

# Fonksiyonun bir kez çalıştırılması
yeniBir()

# Fonksiyon dışından değişkenin çağrılması
print(a)
```

```
>>>

5
```

Global değişkenlerin değeri bir fonksiyon aracılığıyla değiştirilemez. Fonksiyon içinde yapılan güncellemeler fonksiyon içinde kalır.

```
# a global değişkeninin tanımlanması
a = 5
# Fonksiyonun tanımlanması
def birFonksiyon():
    a = 2
    print(a)

# Fonksiyonun çalıştırılması
birFonksiyon()

# a global değişkeninin fonksiyon dışı
# yazdırılması
print(a)
```

```
>>>

2
5
```

Fonksiyon içinde yapılan güncellemenin kalıcı olması isteniyorsa yine **`global`** anahtar kelimesi kullanılmalıdır.

```
# a global değişkeninin tanımlanması
a = 5
# Fonksiyonun tanımlanması
def birFonksiyon():
    global a
    a = 2
    print(a)

# Fonksiyonun çalıştırılması
birFonksiyon()

# a global değişkeninin fonksiyon dışı
# yazdırılması
print(a)
```

```
>>>

2
2
```

2.9 Yerel Değişken Olarak Parametreler

Fonksiyon parametreleri fonksiyon içinde kalan yerel değişkenler gibi davranırlar. Fonksiyon çağrıldığında değer alırlar ve tıpkı yerel değişkenlerde olduğu gibi fonksiyon bloğunun dışına çıkıldığında işlevsel değillerdir.

```python
def bir_fonksiyon(sayı):
    print(sayı)

# 7 rakamının fonksiyona iletilmesi
bir_fonksiyon(7)

# Fonksiyon dışı kullanım hata verir
print(sayı)
```

```
>>>

7
Traceback (most recent call last):
  File "/dosya.py", line 8, in <module>
    print(sayı)
NameError: name 'sayı' is not defined
```

SORULAR

1. Çağrıldığında ekrana “Merhaba” yazdıracak **`selam`** isimli bir fonksiyon oluşturunuz.

2. **`selam`** isimli fonksiyonu çalıştıracak kodu yazınız.

3. **`selam`** isimli fonksiyonu, **`isim`** parametresi taşıyacak şekilde yeniden tanımlayınız ve ekrana “Merhaba **`isim`**” şeklinde çıktı vermesini sağlayacak şekilde içeriğini yeniden düzenleyiniz.

4. Verilen bir sayıya önce 20 ekleyip sonra bu sayıyı 8’e bölerek geri döndüren bir fonksiyonu yazınız.

5. Öğrencilerini sınav yapan bir öğretmen, sınav sorularından bir tanesini iptal etmiştir. 25 soru sorduğu sınavda iptal edilen sorunun puanını, her öğrencinin aldığı puana eklemek istemektedir. Bu sınavda yanlış cevaplar doğruları götürmemektedir. Öğretmenin elinde öğrencilerin doğru cevap sayıları bulunmaktadır. Bu durumda bir öğrencinin aldığı nihai puanı hesaplayıp döndürecek bir Python fonksiyonu nasıl yazılır?

CEVAP ANAHTARI

1.

```
1 def selam():
2   print("Merhaba")
```

2.

```
1 def selam():
2   print("Merhaba")
3 selam()
```

3.

```
1 def selam(isim):
2   print("Merhaba" + isim)
```

4.

```
1 def işlem(sayı):
2   return((sayı + 20)/8)
```

5.

```
# Fonksiyonun tanımlanması ve yazılması
def sonNot1(öğrenciDoğruSayısı):
  return(öğrenciDoğruSayısı*4+4)

# Veya

def sonNot2(öğrenciDoğruSayısı):
  return((öğrenciDoğruSayısı+1)*4)

# Fonksiyonun çağrılması ve kullanılması
print(sonNot1(23))
print(sonNot2(23))
```

3. KIYAS ve DENETİM

3.1 Boolean Değerleri

Boolean bir veri, **`True`** ve **`False`** olmak üzere sadece bu iki değerden biri olabilir. Bir önermenin, kıyaslamanın veya denetimin doğru mu yanlış mı olduğunu kodlamak için bu veri türü kullanılır.

```
1  a = 3>2
2  b = 2>3
3  c = True
4
5  print(a)
6  print(b)
7  print(c)
```

```
>>>

True
False
True
```

3.2 Eşitlik Operatörü

Eşitlik operatörü (==) iki değerin, değişkenin veya ifadenin aynılığını belirlemek için kullanılır. Karşılaştırılan şeyler aynıysa, operatör **`True`** döndürür, aksi takdirde **`False`** döndürür. Operatör, karşılaştırma yaparken veri türünü de hesaba katar. Bu nedenle **`"2"`** string değeri, **`2`** integer değeri ile aynı kabul edilmez.

```
1  # Değerlerin, değişkenlerin ve ifadelerin
2  # eşitlik denetimi
3
4  # Değer denetimi
5  if 'Evet' == 'Evet':
6      print('Değerler aynıdır.')
7
8  # İfade denetimi
9  if (2 > 1) == (5 < 10):
10     print('İfadeler aynıdır.')
11
12 c = '2'
13 d = 2
14
15 # Değişken denetimi
16 if c == d:
17     print('Değişkenler aynıdır.')
18 else:
19     print('Değişkenler aynı değildir.')
```

```
>>>

Değerler aynıdır.
İfadeler aynıdır.
Değişkenler aynı değildir.
```

3.3 Eşitsizlik Operatörü

Eşitsizlik operatörü (**!=**), iki değerin, değişkenin veya ifadenin farklılığını belirlemek için kullanılır. Karşılaştırılan şeyler farklı ise, operatör **`True`** döndürür, aksi takdirde **`False`** döndürür. Bu operatör de, karşılaştırma yaparken veri tipini hesaba katar. Bu sebeple integer olan **2** ile string olan **"2"** karşılaştırıldığında, bu iki şey birbirinden farklıdır anlamında, **`True`** dönütü alınır.

```
 1  # Değerlerin, değişkenlerin ve ifadelerin
 2  # eşitsizlik denetimi
 3  # Değer denetimi
 4
 5  if 'Evet' != 'Hayır':
 6      print('Değerler farklıdır.')
 7
 8  # İfade denetimi
 9  if (2 > 1) != (15 < 10):
10      print('İfadeler farklıdır.')
11  c = '2'
12  d = 2
13
14  # Değişken denetimi
15  if c != d:
16      print('Değişkenler farklıdır.')
17  else:
18
19      print('Değişkenler farklı değildir.')
```

```
>>>

Değerler farklıdır.
İfadeler farklıdır.
Değişkenler farklıdır.
```

3.4 Kıyaslama Operatörleri

Python'da kıyaslama operatörleri iki değeri veya ifadeyi karşılaştırır. En yaygın olanları bunlardır:

<	küçük
>	büyük
<=	küçük veya eşit
>=	büyük veya eşit

Karşılaştırılan değerler veya ifadeler arasındaki ilişki doğru ise **`True`**, değilse **`False`** dönütü alınır.

```
1  a = 2
2  b = 3
3
4  print ("a < b: "+str(a < b))
5  print ("a > b: "+str(a > b))
6  print ("a >= b: "+str(a >= b))
7  print ("a <= b: "+str(a <= b))
```

```
>>>

a < b: True
a > b: False
a >= b: False
a <= b: True
```

3.5 `and` Operatörü

Python'da **`and`** operatörü, iki boolean değer, iki boolean değişken veya iki boolean ifade arasında bir bileşke denetim gerçekleştirir. Kıyaslanan

şeylerin tümü **`True`** ise operatör **`True`** dönütü verir. Ancak bunlardan en az birisi **`False`** ise operatör **`False`** dönütü verir.

```
a = True
b = False

# değer denetimi
print("1: "+ str(True and True))
print("2: "+ str(True and False))
print("3: "+ str(False and False))

# ifade denetimi
print("4: "+ str(1 == 1 and 1 < 2))
print("5: "+ str(1 < 2 and 3 < 1))

# değişken denetimi
print("6: "+ str(a and b))
```

```
>>>

1: True
2: False
3: False
4: True
5: False
6: False
```

3.6 `or` Operatörü

Python'da **`or`** operatörü, iki boolean değer, iki boolean değişken veya iki boolean ifade arasında bir bileşke denetim gerçekleştirir. Kıyaslanan şeylerin en az biri **`True`** ise operatör **`True`** dönütü verir. Ancak bunlardan hepsi **`False`** ise operatör **`False`** dönütü verir.

```
a = True
b = False

# değer denetimi
print("1: "+ str(True or True))
print("2: "+ str(True or False))

# ifade denetimi
print("3: "+ str(1 == 1 or 1 < 2))
print("4: "+ str(1 < 2 or 3 < 1))

# değişken denetimi
print("5: "+ str(a or b))
```

```
>>>

1: True
2: True
3: False
4: True
5: True
6: True
```

3.7 `not` Operatörü

Python'da **`not`** operatörü boolean bir değeri tersine çevirmek için kullanılır. Örneğin gerçek değer **`True`** ise onu **`False`** değerine çevirir.

```
a = True
b = False

# değer denetimi
print("1: "+ str(not True))
print("2: "+ str(not False))

# ifade denetimi
print("3: "+ str(not 1 < 2))

# değişken denetimi
print("4: "+ str(not b))
```

```
>>>

1: False
2: True
3: False
4: True
```

3.8 `if` İfadesi

Python'da **`if`** ifadesi, bir boolean ifadesinin değerlendirilmesine dayalı olarak kodun yürütülüp yürütülmeyeceğini denetlemek için kullanılır. **`if`** ifadesi **`True`** olarak değerlendirilirse, ifadenin ardından gelen girintili kod çalıştırılır. **`if`** ifadesi **`False`** olarak değerlendirilirse, **`if`** ifadesini izleyen girintili kod atlanır ve program **`if`** ifadesiyle aynı seviyede girintilenen sonraki kod satırını yürütür

```
# if İfadesi

test_değer = 100

if test_değer > 50:
    print("1. Bu girintideki kod çalıştırıldı!")

if test_değer > 120:
    print("2. Bu girintideki kod çalıştırıldı!")

print("Program buradan çalışmaya devam etti.")
```

```
>>>

1. Bu girintideki kod çalıştırıldı!
Program buradan çalışmaya devam etti.
```

3.9 `else` İfadesi

Python'da **`else`** ifadesi, **`if`** ifadesindeki sonuç **`False`** olarak değerlendirildiğinde yürütülecek alternatif kodu sağlar. **`if`** bloğundaki girintili kod, sonuç **`True`** olarak değerlendirilirse yürütülür. **`else`** bloğundaki girintili kod yalnızca **`if`** ifadesi **`False`** olduğunda çalıştırılır. Program **`else`** bloğunun sonunun geldiğini girinti seviyesinin azalmasıyla anlar. Dolayısı ile **`else`** bloğu için yazılan kod bittiğinde girinti seviyesinin **`if`** satırı ile aynı seviyeye getirilmesi gerekir.

```
# else İfadesi

test_değer = 50

if test_değer < 1:
    print("Değer < 1")
else:
    print("Değer >= 1")

test_dizesi = "GEÇERLİ"

if test_dizesi == "GEÇERLİ":
    print("Test onaylandı!")
else:
    print("Test onaylanmadı!")
```

```
>>>

Değer >= 1
Test onaylandı!
```

3.10 `elif` İfadesi

Python'da **`elif`** ifadesi, ilk **`if`** ifadesinden sonra devam eden kontrollerin yapılmasına izin verir. **`elif`** ve **`else`** birbirinden farklıdır. **`if`** ve **`elif`** ifadelerinden sonra başka kontrol ifadesi getirilebilir. Ancak **`else`** ifadesi **`if`** bloğunun son kontrol ifadesidir, kendinden sonra başka kontrol ifadesi getirilmez.

Bir **`elif`** ifadesi **`True`** çıkarsa, **`elif`**'in sahip olduğu girintili kod bloğu çalıştırılır. İfade **`False`** çıkarsa, program eğer varsa **`else`** ifadesine yönelir. Birden çok **`elif`** ifadesi, bir dizi kontrol gerçekleştirmek için bir

baş **if** ifadesinin ardından kullanılabilir. Bir **elif** ifadesi **True** çıkarsa, o **if** bloğundaki diğer **elif** ifadeleri çalıştırılmaz.

```
# elif İfadesi

çeşit = "bulgur"

if çeşit == "pirinc":
    print("Pirinc mevcut.")
elif çeşit == "makarna":
    print("Makarna mevcut.")
elif çeşit == "bulgur":
    print("Bulgur mevcut.")
else:
    print("Bilinmiyor!")
```

```
>>>

Bulgur mevcut.
```

3.11 Hata Denetimi

Bir **try** ve **except** bloğu kod bloğundaki hatayı işlemek için kullanılabilir. Hata yaratabilecek kod **try** bloğuna yazılabilir. Yürütme sırasında, bu kod bloğu bir hataya neden olursa, **try** bloğunun geri kalanı yürütmeyi durdurur ve **except** kod bloğu çalıştırılır. Böylece meydana gelen hatanın programı tümüyle durdurmasının önüne geçilir.

```
1  def şubatKaçGün(yıl):
2      yirmidokuz_mu = False
3      if yıl % 4 == 0:
4          yirmidokuz_mu = True
5  try:
6      şubatKaçGün(2018)
7      print(yirmidokuz_mu)
8  except:
9      print('Hata tespit edildi!')
```

```
>>>

Hata tespit edildi!
```

SORULAR

1. Aşağıdaki hatalı yazılmış kodun hatasını açıklayınız.

```
a = 4
b = 2
if a > b:
print("a b'den büyüktür")
```

2. Aşağıdaki kod parçasında ? işaretli yere yazılması gereken kod nedir?

```
if 3 ? 4:
  print("3 ve 4 eşit değildir.")
```

3. Aşağıdaki kod parçasında ? işaretli yere yazılması gereken kod nedir?

```
if 3 == 3 ? 4 == 3:
  print("Önermelerden en az biri doğrudur.")
```

4. İçerisinde **a** ve **b** değişkenlerinin olduğu ve **a**'nın değeri **b**'nin değerinden büyük ise **a**'nın **b**'den çıkarıldığı Python kodu nasıl yazılır?

5. İçerisinde **a** ve **b** değişkenlerinin olduğu ve a'nın değeri b'nin değerine eşit değil ise **a**'nın **b**'den çıkarıldığı Python kodu nasıl yazılır?

6. İçerisinde **a** ve **b** değişkenlerinin olduğu ve **a**'nın değeri **b**'nin değerine eşit ise "Tebrikler!" çıktısını veren, değilse "Tekrar Deneyin!" çıktısını veren Python kodu nasıl yazılır?

7. İçerisinde **a** ve **b** değişkenlerinin olduğu bir Python kodu olsun. Eğer **a** ve **b** eşit ise ekrana "1" yazan, yok eğer **a**, **b**'den büyük ise "2" yazan, bunlardan hiçbiri değilse ekrana "3" yazan bir Python kodunu yazınız.

8. İçerisinde **a**, **b**, **c** ve **d** değişkenlerinin olduğu bir Python kodu olsun. Aynı anda **a**'nın **b**'ye eşit olduğu ve **c**'nin de **d**'ye eşit olduğu durumda ekrana "Örtüşme bulundu!" yazan bir Python kodunu yazınız.

9. Bu soruda bir oyun yazmanız isteniyor. Oyun, bir sayı tahmin etme oyunudur. Programın kaynak kodunda tahmin edilecek sayı için bir değişken tanımlayın. Bu değişkene **gizliSayı** ismini verelim. Oyunu yazan kişi değişkene bir sayı ataması yapar. Yarışmacıların görevi bu sayıyı tahmin etmektir. Program çalıştırıldığında yarışmacıdan ekrana bir tam sayı girmesini ister. Program yarışmacının girdiği sayıyı **gizliSayı** ile kıyaslar. Yarışmacı burada 3 farklı durum ile karşılaşabilir. Girdiği sayı, **gizliSayı**'ya eşit olabilir. Bu durumda ekranda "Tebrikler!" yazdığını görür. Girdiği sayı

gizliSayı'dan küçük olabilir. Bu durumda ekranda "Yukarı" yazdığını görür. Tersi durumda ise ekranda "Aşağı" yazdığını görür. Yarışmacı doğru tahmin yapana kadar programı tekrar tekrar çalıştırır. Doğru sonuca kaç denemede ulaştığı bir kâğıda not edilir. Sonra diğer yarışmacılar oyunu oynar. Onların da kaç denemede doğru sonuca ulaştıkları not edilir. En az denemede **gizliSayı**'yı bulan yarışmacı oyunu kazanır.

CEVAP ANAHTARI

1. 4. satırda girintileme yapılmamıştır.

2. `!=`

3. `or`

4.

```
a = 50
b = 10
if a > b:
  a - b
```

5.

```
a = 50
b = 10
if a != b:
  a - b
```

6.

```
a = 50
b = 10
if a == b:
  print("Tebrikler!")
else:
  print("Tekrar Deneyin!")
```

7.

```
a = 50
b = 10
if a == b:
  print("1")
elif a > b:
  print("2")
else:
  print("3")
```

8.

```
a = 50
b = 50
c = 20
d = 20

if a == b and c == d:
  print("Örtüşme bulundu!")
```

9.

```
# BU BİR SAYI TAHMİN ETME OYUNUDUR.

# 1.Gizli sayı tanımlanır.
gizliSayı = 583

# 2.Yarışmacının tahmini istenir.
tahmin = input("Bir tam sayı giriniz:")

# 3.Alınan sayı integer formatına dönüştürülür.
tahmin = int(tahmin)

# 4.Kıyaslama işlemi yapılır ve sonuçlar
# yazdırılır.
if tahmin > gizliSayı:
    print("Aşağı")
elif tahmin < gizliSayı:
    print("Yukarı")
else:
    print("Tebrikler!")
```

4. KOLEKSİYONLAR

Koleksiyonlar, bir değişkenin birden fazla değer taşıması gerektiğinde kullanılan veri türlerdir. Python programlama dilinde başlıca dört koleksiyon türü vardır:

1. *List* (liste), öğeleri sıralı, indeksli ve değiştirilebilir bir koleksiyondur. Aynı öğeden birden fazla içermeye izin verir.
2. *Tuple* (demet), öğeleri sıralı, indeksli ancak değiştirilemeyen bir koleksiyondur. Aynı öğeden birden fazla içermeye izin verir.
3. *Set* (küme), öğeleri sıralı olmayan, indeksli olmayan ve değiştirilemeyen bir koleksiyondur. Aynı öğeden birden fazla içermeye izin vermez.
4. *Dictionary* (sözlük), öğeleri sıralı[5], anahtar indeksli ve değiştirilebilir bir koleksiyondur. Aynı öğeden (anahtardan) birden fazla içermeye izin vermez.

[5] Python 3.7'den itibaren Dictionary koleksiyon türü, sıralı (ordered) bir yapıya kavuşmuştur.

```
# list
birListe = ["elma", "armut", "kiraz", "kiraz"]

# tuple
birDemet = ("elma", "armut", "kiraz", "kiraz")

# set
birKüme = {"elma", "armut", "kiraz"}

# dict
birSözlük = {"tür" : "elma", "kg" : 32}
```

Yukarıda verilen koleksiyon türleri için sıralılık, indekslilik ve değiştirilebilirlik özelliklerinden bahsedildi. Bir koleksiyon türünün sıralı olması demek, sahip olduğu öğelerin belli bir sıraya göre dizilmesi ve bu sıralamanın bilgisayar tarafından korunması demektir. Bu sıralama, koleksiyon türü tanımlandığı anda belleğe kaydedilir. Tanımlanma sırasında soldan sağa doğru yazılan sıra geçerli olan sıradır. Değiştirilebilir koleksiyonlarda daha sonra istenirse bu sıralama değiştirilebilir. *List*, *tuple* ve *dictionary* koleksiyonları sıralı özellikte iken *set* koleksiyonları sıralı değildir. Koleksiyon, alt alta çekmeceleri olan bir dolaba benzetilirse sıralı olmayan bir koleksiyonda siz ilk çekmeceye gözlüğünüzü koyduğunuzda ve sonraki bir zamanda gelip ilk çekmeceyi açtığınızda gözlüğünüzü orada bulamama ihtimaliniz yüksektir. Gözlüğünüz orada değilse mutlaka diğer çekmecelerden birindedir. Özetle sıralı koleksiyonlarda, hem değerlerin kendileri hem de koleksiyondaki yerleri bellekte tutulur. Sıralı olmayan koleksiyonlarda ise değerlerin sadece kendileri bellekte tutulur.

Python'da indeksli koleksiyonlardan *list* ve *tuple* türlerinde koleksiyonun taşıdığı değerlere erişmek için indeks numaraları kullanılır. İndeks 0'dan başlar ve tam sayılar ile ilerler. Negatif rakamların kullanılması da mümkündür. Negatif indeks numaraları sondan başa doğru erişim sağlar. Örneğin -1. öğe sondan 1. öğedir. Anahtar indeksli olan *dictionary* türünde ise rakamların yerini anahtarlar alır. Her bir öğenin bir anahtarı (*key*) bir de değeri (*value*) vardır. Bir değere erişilmek istendiğinde onun anahtarı kullanılır. Koleksiyon öğelerine erişimin nasıl sağlandığına dair bilgiler ve örnek kodlar ilgili konu başlıkları altında verilmiştir.

Değiştirilebilirlik, koleksiyon türlerinin bir başka önemli özelliğidir. Python koleksiyonlarından *list* ve *dictionary* değiştirilebilir iken *tuple* ve *set* değiştirilemez koleksiyonlardır. Değiştirilebilir olmak ilk bakışta olumlu bir özellik gibi görünse de bazı durumlarda kullanımı dezavantajlı olabilmektedir. Örneğin belli sayıda öğeden oluşan ve ekleme-çıkarma yapılmasını istemediğiniz bir koleksiyona zaman zaman ihtiyaç duyulabilir. Buna IP adresi, kredi kartı numarası, bazı aktivasyon kodları gibi örnekler verilebilir. Değiştirilemezlik bir yönüyle de güvenlik faktörü ile ilişkili bir özelliktir. Olduğu şekliyle kalmasını sağlamak istediğiniz bir koleksiyonu güvenceye almak için onu *tuple* veya *set* türünde tanımlayabilirsiniz.

Bir koleksiyon türü seçerken, o türün özelliklerini bilmek ve ona göre seçmek gerekir. Tasarlanan bir veri koleksiyonu için doğru türün seçilmesi kullanışlılık, verimlilik veya güvenlik yönünden avantaj sağlayabilir. Örneğin bir programcı, koleksiyon öğelerine indeks numarası ile ulaşmak istiyorsa *list* veya *tuple* türlerini tercih etmelidir. Eğer koleksiyona zaman içinde yeni öğeler eklemek veya mevcutları

koleksiyondan çıkarmak gibi bir amaç güdüyorsa bu durumda *tuple* uygun bir tercih olmayacaktır.

4.1 LİSTELER

Python'da listeler, bir veri koleksiyonunun kolay kullanımına izin veren sıralı, indeksli ve değiştirilebilir öğe koleksiyonlarıdır. Liste öğeleri virgülle ayrılır ve tüm öğeler köşeli parantez `[ ]` arasına yerleştirilir. Virgül ile sonraki değer arasına boşluk koymak iyi bir uygulamadır. Listedeki değerlerin benzersiz olması gerekmez (aynı değer tekrarlanabilir). Boş listeler, köşeli parantez içinde herhangi bir değer içermez.

```
1  şifre = [6, 1, 6, 9, 3]
2  print(şifre)
3
4  # Boş bir liste oluşturma
5  liste1 = []
```

```
>>>

[6, 1, 6, 9, 3]
```

4.1.1 Listelerde Veri Türleri

Python'da listeler, aynı köşeli parantez içinde birden çok farklı veri türü içerebilen çok yönlü bir veri türüdür. Bir listedeki olası veri türleri, sayılar (integer, float), yazı karakterleri (strings), diğer nesneler ve hatta diğer listeler olabilir.

```
sayılar = [1, 2, 3, 4, 10]
isimler = ['Ali', 'Zeynep', 'Mahir']
karısık = ['Fırat', 1, 2]
listelerListesi = [['Lale', 1], ['Gül', 8]]
```

Diğer programlama dillerindeki *array* olarak ifade edilen koleksiyonları Python'da listeler karşılar. Yani Python'da *array* yerine listeler kullanılır.

4.1.2 Liste Öğelerine Erişim

Liste öğelerine erişim için indeks numarası kullanılır. İndekslenen koleksiyonlarda indeks 0'dan başlar. Ayrıca negatif indeks numaraları da öğelere erişim için pratik bir yöntem olarak kullanılabilmektedir. Bu durumda 0. öğe listenin ilk öğesidir ve -1. öğe ise sondan 1. öğedir.

```
şifre = [6, 1, 6, 9, 3]
print(şifre[0])
print(şifre[-1])
```

```
>>>

6
3
```

Eğer belli bir indeks aralığındaki tüm öğeler çağrılmak isteniyorsa bu, iki nokta (:) kullanımı ile gerçekleştirilir. İki nokta (:), her iki yanına değer alabildiği gibi, sadece sağına veya sadece soluna da değer alabilir. İki yanına [a:b] gibi değerler aldığında a'dan b'ye kadar (a dahil b hariç) olan öğelerin tümünü çağırır. Sadece soluna [a:] gibi tek bir değer aldığında a'dan listenin sonuna kadarki (a dahil) tüm öğeleri çağırır.

Sadece sağına [:b] gibi tek bir değer aldığında listenin başından b'ye kadar (b hariç) tüm öğeleri çağırır.

```
şifre = ["sıfır", "bir", "iki", "üç", "dört"]
print(şifre[1:4])
print(şifre[2:])
print(şifre[:2])
```

```
>>>

['bir', 'iki', 'üç']
['iki', 'üç', 'dört']
['sıfır', 'bir']
```

4.1.3 Değer Değiştirme

Listedeki bir öğenin değerini değiştirmek için öğe indeks numarası ile çağrılır ve yeni değer ataması yapılır.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
isimler[1]="Selim"
print(isimler)
```

```
>>>

['Ali', 'Selim', 'Mahir']
```

4.1.4 Öğeleri Döndürme

Bir listedeki öğeleri döndürmek için **`for`** döngüsü kullanılabilir.

```
1 isimler = ['Ali', 'Zeynep', 'Mahir']
2 for x in isimler:
3     print(x)
```

```
>>>
Ali
Zeynep
Mahir
```

4.1.5 Öğe Sorgulama

Bir listede bir öğenin var olup olmadığı, **`if`** ve **`in`** anahtar kelimeleri ile sorgulanabilir.

```
1 isimler = ['Ali', 'Zeynep', 'Mahir']
2 if "Mahir" in isimler:
3     print("Mahir listede mevcuttur.")
```

```
>>>

Mahir listede mevcuttur.
```

4.1.6 Öğe Sayısı

Bir listenin sahip olduğu öğelerin toplam sayısı **`len()`** fonksiyonuyla elde edilebilir.

```
1 isimler = ['Ali', 'Zeynep', 'Mahir']
2 print(len(isimler))
```

```
>>>

3
```

4.1.7 Liste İnşası

Liste inşası programatik olarak **`list()`** fonksiyonu ile yapılmaktadır. Koleksiyon türlerinden list türünde muhafaza edilmek istenen yinelenebilir (*iterable*) bir veri, **`list()`** fonksiyonuna parametre olarak gönderilirse kolaylıkla bir liste elde edilebilir. String verisi de yinelenebilir bir veridir. Bu fonksiyona parametre olarak verildiğinde karakterler tek tek listenin öğesi haline gelir.

```
1  liste1 = list(("elma", "armut", "kiraz"))
2  print(liste1)
3
4  liste2 = list(range(10))
5  print(liste2)
6
7  liste3 = list("ADEM")
8  print(liste3)
```

```
>>>

['elma', 'armut', 'kiraz']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['A', 'D', 'E', 'M']
```

4.1.8 İki Listeyi Birleştirme

İki listeyi birleştirmek için artı (+) işareti kullanılır.

```
1  liste1 = ['Ali', 'Zeynep', 'Mahir']
2  liste2 = [70,40,15]
3
4  büyükListe = liste1 + liste2
5  print(büyükListe)
```

```
>>>

['Ali', 'Zeynep', 'Mahir', 70, 40, 15]
```

4.1.9 Çok Boyutlu Listeler

Diğer programlama dillerinde kullanılan *array* koleksiyonları çok boyutlu olabilmektedir. Python'da *array* yerine kullanılan listeleri de çok boyutlu olarak tanımlamak mümkündür. Bu, liste-içinde-liste şeklinde tanımlanarak yapılır. Matematikte kullanılan matrisleri de bu şekilde tanımlamak mümkündür. Matrisler belli sayıda satır ve sütundan oluşan sayılar tablosudur. Bu tablolar iki boyutlu (**i** ve **j** veya **x** ve **y**) koleksiyonlardır. Aşağıda 3 sütun ve 2 satırdan oluşan bir matris örneği ve bunun Python'da liste olarak nasıl tanımlandığı gösterilmiştir.

$a_{i,j}$	j=0	j=1	j=2
i=0	32	8	45
i=1	11	68	72

```
1 a = [[32, 8, 45], [11, 68, 72]]
2 print(a[1][2])
```

```
>>>

72
```

Her ne kadar matrisler gibi çok boyutlu koleksiyonları listeler ile tanımlamak mümkün olsa da bu kullanım yaygın bir kullanım değildir. Çok boyutlu koleksiyonlar (çok boyutlu arrayler) ile çalışmak durumunda olanlar için, bir topluluk tarafından açık-kaynak olarak geliştirilen NumPy kütüphanesi daha elverişlidir. NumPy, Nümerik Python için kullanılan bir kısaltmadır ve matrislerle kullanılabilecek matematiksel fonksiyonları içerir. NumPy kütüphanesi Python'un çekirdek paketinde yer almaz. Bu kütüphanenin kullanılabilmesi için önce sisteme bir Python paket yöneticisi ile yüklenmesi gerekir. Kütüphanenin yüklenmesi ve kullanımı, bu kitabın kapsamı dışındadır. NumPy hakkında daha fazla bilgi için https://numpy.org/ adresine başvurulabilir.

4.1.10 Liste Metotları

append()

Python'da, **`append()`** metodunu kullanarak bir listenin sonuna değer ekleyebilirsiniz. Bu işlem, aktarılan nesneyi listenin en sonuna yeni bir öğe olarak yerleştirir.

```
1 sepet = ['süt', 'yumurta']
2 sepet.append('peynir')
3 print(sepet)
```

```
>>>

['süt', 'yumurta', 'peynir']
```

insert()

Python'da **`insert()`** metodu, bir listenin istenilen indeks pozisyonuna yeni değer eklemek için kullanılır. Eklenen yeni değer, kendinden sonra gelen öğelerin pozisyonlarında bir numara kaymaya sebep olur.

```
1 sepet = ['süt', 'yumurta', 'peynir']
2 sepet.insert(1,'ekmek')
3 print(sepet)
```

```
>>>

['süt', 'ekmek', 'yumurta', 'peynir']
```

count()

Python'da **`count()`** metodu, argüman olarak aldığı arama terimini bir listede arar ve eşleşen öğelerin sayısını verir.

```
1 çanta = ['klm', 'silgi', 'dftr', 'ktp', 'klm']
2 kalemSayısı = çanta.count('klm')
3 print(kalemSayısı)
```

```
>>>

2
```

sort()

Python'da **`sort()`** metodu, çağrıldığı listenin içeriğini sıralayacaktır. Sayısal listeler küçükten büyüğe doğru sıralanır. String türü listeler ise alfabetik sıraya göre sıralanır. Bu metot bir listeye uygulandığında listenin orjinal sıralamasını değiştirir ve herhangi bir dönüt vermez. Eğer listenin orjinali korunmak isteniyorsa bu durumda sıralama için **`sorted()`** fonksiyonunun kullanılması uygun olur.

```
1  listemiz = [4, 2, 1, 3]
2  listemiz.sort()
3  print(listemiz)
4
5  korunanListemiz = [62, 12, 76, 33]
6  siralanmis = sorted(korunanListemiz)
7  print(siralanmis)
8  print(korunanListemiz)
```

```
>>>

[1, 2, 3, 4]
[12, 33, 62, 76]
[62, 12, 76, 33]
```

reverse()

Python'da **`reverse()`** metodu, çağrıldığı listenin sıralamasını terse çevirir.

```
listemiz = [4, 2, 1, 3]

# Küçükten büyüğe sıralama
listemiz.sort()
print(listemiz)

# Sıralamayı terse çevirme
listemiz.reverse()
print(listemiz)
```

```
>>>

[1, 2, 3, 4]
[4, 3, 2, 1]
```

index()

Python'da **`index()`** metodu, verilen liste öğesinin ilk kez görüldüğü pozisyonu verir.

```
meyveler=['elma','armt','kiraz','şftli','kiraz']
x = meyveler.index("kiraz")
print(x)

sayılar = [4, 32, 64, 32, 16, 32]
y = sayılar.index(32)
print(y)
```

```
>>>

2
1
```

pop()

Python'da **pop()** metodu, verilen pozisyondaki liste öğesini siler ve o öğenin değerini döndürür. Eğer pozisyon verilmemişse işlemi listenin son öğesi için gerçekleştirir. Çünkü **pop()** metodunun aldığı parametrenin varsayılan değeri -1'dir. -1 listelerde son öğenin pozisyonudur.

```
1  meyveler = ['elma','armut','kiraz']
2  x = meyveler.pop(1)
3  print(x)
4  print(meyveler)
```

```
>>>

armut
['elma', 'kiraz']
```

remove()

Python'da **remove()** metodu, verilen öğeyi listede bulduğu ilk pozisyondan siler ve herhangi bir değer döndürmez. Bu metot bir öğe verilmeden kullanılırsa veya verilen öğe listede yok ise program hata verir.

```
1  meyveler = ['elma','armut','kiraz','armut']
2  meyveler.remove('armut')
3  print(meyveler)
```

```
>>>

['elma', 'kiraz', 'armut']
```

clear()

Python'da **clear()** metodu, listedeki tüm öğeleri siler.

```
meyveler = ['elma','armut','kiraz','armut']
meyveler.clear()
print(meyveler)
```

```
>>>

[]
```

copy()

Python'da **copy()** metodu, bir listenin kopyasını oluşturmak için kullanılır. Bir liste **liste2=liste1** şeklinde kopyalanamaz. Bu komut sadece **liste2**'yi **liste1**'in referansı yapar. Yani **liste1**'de yapılan değişiklikler otomatik olarak **liste2**'ye de yansır.

```
# Orjinal liste
meyveler = ['elma','armut','kiraz','armut']

# copy() ile kopyalama
meyveler2 = meyveler.copy()

# eşitlik ile kopyalama denemesi
meyveler3 = meyveler

# Orjinalden elma öğesini silme
meyveler.remove('elma')

# Silme (remove) işleminin iki kopya liste
# üzerindeki etkisini görmek için her ikisini de
# yazdır
print(meyveler2)
print(meyveler3)
```

```
>>>

['elma', 'armut', 'kiraz', 'armut']
['armut', 'kiraz', 'armut']
```

4.2 DEMETLER

Demetler öğeleri sıralı, indeksli ve değiştirilemeyen koleksiyon türüdür. Bu tür aynı öğeden birden fazla içermeye izin verir. Yazımları standart parantez `()` ile olur.

```
# tuple
birDemet = ("elma", "armut", "kiraz", "kiraz")

print(birDemet)
```

```
>>>

('elma', 'armut', 'kiraz', 'kiraz')
```

4.2.1 Demet Öğelerine Erişim

Demet öğelerine erişim için indeks numarası kullanılır. İndekslenen diğer koleksiyonlarda olduğu gibi indeks 0'dan başlar. Ayrıca negatif indeks numaraları da öğelere erişim için pratik bir yöntem olarak kullanılabilmektedir. Bu durumda 0. öğe listenin ilk öğesidir ve -1. öğe ise sondan 1. öğedir. Erişilmek istenen öğenin indeks numarası köşeli parantez içinde verilir.

```
# tuple
birDemet = ("elma", "armut", "kiraz", "kiraz")

a = birDemet[0]
b = birDemet[-1]

print(a)
print(b)
```

```
>>>

elma
kiraz
```

Eğer belli bir indeks aralığındaki öğeler çağrılmak isteniyorsa bu, listelerde olduğu gibi iki nokta (:) kullanımı ile gerçekleştirilir. İki nokta (:), her iki yanına değer alabildiği gibi, sadece sağına veya sadece soluna da değer alabilir. İki yanına [a:b] gibi değerler aldığında a'dan b'ye kadar (a dahil b hariç) olan öğelerin tümünü çağırır. Sadece soluna [a:] gibi tek bir değer aldığında a'dan listenin sonuna kadarki (a dahil) tüm öğeleri çağırır. Sadece sağına [:b] gibi tek bir değer aldığında listenin başından b'ye kadar (b hariç) tüm öğeleri çağırır. Bir demetten belli bir indeks aralığındaki öğeler çağrıldığında gelen dönüt, yeni bir demeti oluşturur.

```
# Orjinal Demet
şifre = ("sıfır", "bir", "iki", "üç", "dört")

# Yeni Demetler
x = şifre[1:4]
y = şifre[2:]
z = şifre[:2]

print(x)
print(y)
print(z)
```

```
>>>

('bir', 'iki', 'üç')
('iki', 'üç', 'dört')
('sıfır', 'bir')
```

4.2.2 Değer Değiştirme

Demetlerin değiştirilemez özellikte olmaları sebebiyle demet öğelerinin taşıdıkları değerler değiştirilemez. Ayrıca bir demete yeni bir öğe

eklenemez ve mevcut bir öğe demetten çıkarılamaz. Bir demet, listeye dönüştürülürse üzerinde istenilen değişiklikler yapılabilir. Demetten listeye dönüşüm **`list()`** fonksiyonuyla yapılır. Üzerinde istenilen değişiklik yapılan liste daha sonra **`tuple()`** fonksiyonuyla orjinal demet ile aynı isimde bir demete dönüştürülür. Böylece orjinal demet üzerinde değişiklik gerçekleşmiş olur.

```
# orjinal demet
birDemet = ("elma", "armut", "kiraz", "kiraz")
print(birDemet)

# demetten listeye dönüşüm ve değişiklik
birListe = list(birDemet)
birListe[2] = "dut"

# listeden geri demete dönüşüm
birDemet = tuple(birListe)

print(birDemet)
```

```
>>>

('elma', 'armut', 'kiraz', 'kiraz')
('elma', 'armut', 'dut', 'kiraz')
```

4.2.3 Öğeleri Döndürme

Bir demetteki öğeleri döndürmek için **`for`** döngüsü ve **`in`** anahtar kelimesi kullanılabilir.

```
1 isimler = ('Ali', 'Zeynep', 'Mahir')
2 for x in isimler:
3     print(x)
```

```
>>>

Ali
Zeynep
Mahir
```

4.2.4 Öğe Sorgulama

Bir demette bir öğenin var olup olmadığı, **`if`** ve **`in`** anahtar kelimeleri ile sorgulanabilir.

```
1 isimler = ('Ali', 'Zeynep', 'Mahir')
2 if "Mahir" in isimler:
3     print("Mahir demette mevcuttur.")
```

```
>>>

Mahir demette mevcuttur.
```

4.2.5 Öğe Sayısı

Bir demetin sahip olduğu öğelerin toplam sayısı **`len()`** fonksiyonuyla elde edilebilir.

```
1 isimler = ('Ali', 'Zeynep', 'Mahir')
2 print(len(isimler))
```

```
>>>

3
```

4.2.6 Tek Öğeli Demet

Demetler tek bir öğeye sahip olabilirler. Ancak tek öğeli demet tanımlarken aşağıdaki gibi öğeden sonra virgül kullanılması gerekmektedir. Aksi takdirde tanımlanan şey demet değil string olur.

```
1 tekDemet = ('Ali',)
2 birString = ('Ali')
3
4 print(tekDemet)
5 print(birString)
```

```
>>>

('Ali',)
Ali
```

4.2.7 İki Demeti Birleştirme

İki demeti birleştirmek için artı (+) işareti kullanılır.

```
1  demet1 = ('Ali', 'Zeynep', 'Mahir')
2  demet2 = (70,40,15)
3
4  büyükDemet = demet1 + demet2
5  print(büyükDemet)
```

```
>>>

('Ali', 'Zeynep', 'Mahir', 70, 40, 15)
```

4.2.8 Demet İnşası

Demet inşası programatik olarak **`tuple()`** fonksiyonu ile yapılmaktadır. Koleksiyon türlerinden demet türünde muhafaza edilmek istenen yinelenebilir (*iterable*) bir veri, **`tuple()`** fonksiyonuna parametre olarak gönderilirse kolaylıkla bir demet elde edilebilir. String verisi de yinelenebilir bir veridir. Bu fonksiyona parametre olarak verildiğinde karakterler tek tek demetin öğesi haline gelir.

```
demet1 = tuple(("elma", "armut", "kiraz"))
print(demet1)

demet2 = tuple(range(10))
print(demet2)

demet3 = tuple("ADEM")
print(demet3)
```

```
>>>

('elma', 'armut', 'kiraz')
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
('A', 'D', 'E', 'M')
```

4.2.9 Demet Metodları

count()

Python'da **count()** metodu, bağımsız değişken olarak aldığı arama terimini bir demette arar ve eşleşen öğelerin sayısını verir.

```
çanta = ('klm', 'silgi', 'dftr', 'ktp', 'klm')
kalemSayısı = çanta.count('klm')
print(kalemSayısı)
```

```
>>>

2
```

index()

Python'da **index()** metodu, verilen demet öğesinin ilk kez görüldüğü pozisyonu verir.

```
meyveler=('elma','armt','kiraz','şftali','kiraz')
x = meyveler.index("kiraz")
print(x)
```

```
>>>

2
```

4.3 KÜMELER

Kümeler, öğeleri sıralı olmayan, indekslenmeyen ve değiştirilemeyen koleksiyon türüdür. Bu tür aynı öğeden birden fazla içermeye izin vermez. Yazımları küme parantezi `{}` ile olur.

```
# set
birKüme = {"elma", "armut", "kiraz"}

print(birKüme)
```

```
>>>

{'elma', 'armut', 'kiraz'}
```

4.3.1 Küme Öğelerine Erişim

Küme öğelerine indeks numaraları ile erişilmez. Erişilmeye çalışılırsa aşağıdaki gibi bir hata alınır.

```
# set
birKüme = {"elma", "armut", "kiraz"}

print(birKüme[0])
```

```
>>>

  File "C:\test.py", line 4, in <module>
    print(birKüme[0])
TypeError: 'set' object is not subscriptable
```

4.3.2 Öğeleri Döndürme

Bir kümedeki öğeleri döndürmek için **for** döngüsü ve **in** anahtar kelimesi kullanılabilir. Sizin konsoldan aldığınız sıralama bizimki ile aynı olmayabilir. Çünkü küme koleksiyonları en başta söylediğimiz gibi sıralı özellikte değildir. Her yeni döndürmede öğeleri farklı sıralamada önünüze getirebilir.

```
isimler = {'Ali', 'Zeynep', 'Mahir'}
for x in isimler:
    print(x)
```

```
>>>

Zeynep
Ali
Mahir
```

4.3.3 Öğe Sorgulama

Bir kümede bir öğenin var olup olmadığı, **if** ve **in** anahtar kelimeleri ile sorgulanabilir.

```
isimler = {'Ali', 'Zeynep', 'Mahir'}
if "Mahir" in isimler:
    print("Mahir kümede mevcuttur.")
```

```
>>>

Mahir demette mevcuttur.
```

Sorgulama sadece in anahtar kelimesi ile de yapılabilir.

```
isimler = {'Ali', 'Zeynep', 'Mahir'}
print("Mahir" in isimler)
```

```
>>>

True
```

4.3.4 Öğe Ekleme

Küme öğeleri değiştirilemez ancak kümelere yeni öğeler eklenebilir. Bunun için **`add()`** metodu kullanılır. Birden fazla ekleme yapılacaksa **`update()`** metodu daha kullanışlıdır. Aşağıdaki örnek kodlarda sizin **`print()`** ile alacağınız sıralamalar yukarıda açıkladığımız gerekçe sebebiyle farklı olabilir.

```
1 isimler = {'Ali', 'Zeynep', 'Mahir'}
2 isimler.add('Amine')
3 print(isimler)
```

```
>>>

{'Mahir', 'Ali', 'Zeynep', 'Amine'}
```

```
1 isimler = {'Ali', 'Zeynep', 'Mahir'}
2 isimler.update(['Amine', 'Furkan', 'Elif'])
3 print(isimler)
```

```
>>>

{'Amine', 'Elif', 'Zeynep', 'Mahir', 'Ali', 'Furkan'}
```

4.3.5 Öğe Silme

Kümeden öğe silmenin farklı yöntemleri vardır. Örneğin, **`remove()`** metodu, değeri bilinen öğeyi kümeden silmek için kullanılabilir. Yanlış değer girilirse hata verir. **`discard()`** metodunun kullanımı da aynıdır. Ancak bu metot aranan değer bulunamadığında hata vermez. Eğer tüm öğeler kümeden silinmek isteniyorsa **`clear()`** metodu kullanılmalıdır. Bunların kullanımına ilişkin örnek kodlar aşağıda verilmiştir. **`del`** anahtar kelimesi ise küme değişkenini ortadan kaldırır.

```
isimler = {'Ali', 'Zeynep', 'Mahir'}

isimler.remove('Ali')
print(isimler)

isimler.discard('Zeynep')
print(isimler)

isimler.clear()
print(isimler)

print('-------------------------------------')

del isimler
print(isimler)
```

```
>>>

{'Zeynep', 'Mahir'}
{'Mahir'}
set()
-------------------------------------
  File "C:\Users\sau\Desktop\test.py", line 15, in
<module>
    print(isimler)
NameError: name 'isimler' is not defined
```

4.3.6 Öğe Sayısı

Bir kümenin sahip olduğu öğelerin toplam sayısı **`len()`** fonksiyonuyla elde edilebilir.

```
isimler = {'Ali', 'Zeynep', 'Mahir'}
print(len(isimler))
```

```
>>>

3
```

4.3.7 İki Kümeyi Birleştirme

İki kümeyi birleştirmek için **union()** veya **update()** metotları kullanılabilir. **union()** metodu birleşmeden doğan büyük kümeyi yeni bir değişken olarak geri döndürür. Yani diğer iki küçük kümenin orjinal halleri korunmuş olur. Ancak **update()** metodunda kümelerden biri diğerinin üzerine eklenir. Üzerine eklemleme yapılan kümenin orjinal hali kaybedilmiş olur. Her iki metot ile yapılan birleştirme işleminde mükerrer öğeler dışarıda bırakılır. Yani bir kümedeki tüm öğelerin birbirinden farklı olması sağlanır.

```
isimler1 = {'Ali', 'Zeynep', 'Mahir'}
isimler2 = {'Mehmet', 'Ali'}
isimler3 = isimler1.union(isimler2)

print(isimler3)
print(isimler1)
isimler1.update(isimler2)
print(isimler1)
```

```
>>>

{'Ali', 'Mehmet', 'Zeynep', 'Mahir'}
{'Zeynep', 'Ali', 'Mahir'}
{'Ali', 'Mehmet', 'Zeynep', 'Mahir'}
```

4.3.8 Küme İnşası

Küme inşası programatik olarak **`set()`** fonksiyonu ile yapılmaktadır. Koleksiyon türlerinden küme türünde muhafaza edilmek istenen yinelenebilir (*iterable*) bir veri, **`set()`** fonksiyonuna parametre olarak gönderilirse kolaylıkla bir küme elde edilebilir. String verisi de yinelenebilir bir veridir. Bu fonksiyona parametre olarak verildiğinde karakterler tek tek kümenin öğesi haline gelir. Mükerrer öğeler **`set()`** fonksiyonu ile küme inşası sırasında otomatik olarak dışarıda bırakılır.

```
1  küme1 = set(("elma", "armut", "elma"))
2  print(küme1)
3
4  küme2= set(range(10))
5  print(küme2)
6
7  küme3= set("DEDE")
8  print(küme3)
```

```
>>>

{'elma', 'armut'}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{'D', 'E'}
```

4.3.9 Küme Metodları

copy()

`copy()` metodu verilen bir kümenin bağımsız kopyasını oluşturur.

```
1 küme1 = {"elma", "armut"}
2 küme2 = küme1.copy()
3 küme1.remove("elma")
4
5 print(küme1)
6 print(küme2)
```

```
>>>

{'armut'}
{'elma', 'armut'}
```

difference()

A kümesinin B kümesinden farkını verir.

```
1 A = {"elma", "armut"}
2 B = {"elma", "dut"}
3 C = A.difference(B)
4
5 print(C)
6 print(A)
```

```
>>>

{'armut'}
{'armut', 'elma'}
```

Hem A hem B kümesinde bulunan öğeler, A kümesinden çıkarılmak istenirse **difference_update()** metodu kullanılabilir.

```
A = {"elma", "armut"}
B = {"elma", "dut"}
A.difference_update(B)

print(A)
```

```
>>>

{'armut'}
```

intersection()

A ve B kümesinin kesişimini verir.

```
A = {"elma", "armut"}
B = {"elma", "dut"}
C = A.intersection(B)

print(C)
print(A)
```

```
>>>

{'elma'}
{'armut', 'elma'}
```

A kümesi B kümesi ile olan farklılıklarından arındırılmak istenirse **intersection_update()** metodu kullanılabilir.

```
A = {"elma", "armut"}
B = {"elma", "dut"}
A.intersection_update(B)

print(A)
```

```
>>>

{'elma'}
```

symmetric_difference()

A kümesinin B kümesinden farklı olan öğeleri ile B kümesinin A kümesinden farklı olan öğelerini bir araya toplar ve geri döndürür.

```
A = {"elma", "armut"}
B = {"elma", "dut"}
C = A.symmetric_difference(B)

print(C)
print(A)
```

```
>>>

{'dut', 'armut'}
{'armut', 'elma'}
```

Eğer A kümesi, yukarıda elde edilen farklar kümesine dönüştürülmek istenirse **`symmetric_difference_update()`** metodu kullanılabilir.

```
A = {"elma", "armut"}
B = {"elma", "dut"}
A.symmetric_difference_update(B)

print(A)
```

```
>>>

{'armut', 'dut'}
```

isdisjoint()

A ve B kümelerinin ayrık kümeler olup olmadığını sorgular. A ve B kümeleri hiç ortak öğe bulundurmuyorsa **`True`** değeri geri döndürür.

```
A = {"elma", "armut"}
B = {"kiraz", "dut"}
ayrıkmı = A.isdisjoint(B)
print(ayrıkmı)
```

```
>>>
True
```

issubset()

A kümesinin B kümesinin alt kümesi olup olmadığını sorgular. Eğer A kümesinin tüm öğeleri B kümesinde mevcut ise **`True`** değerini döndürür.

```
A = {"elma", "armut"}
B = {"elma", "armut", "kiraz", "dut"}
altkümemi= A.issubset(B)
print(altkümemi)
```

```
>>>

True
```

issuperset()

A kümesi B kümesinin kapsayan kümesi olup olmadığını sorgular. Eğer B kümesinin tüm öğeleri A kümesinde mevcut ise **`True`** değerini döndürür.

```
A = {"elma", "armut"}
B = {"elma"}
altkümemi= A.issuperset(B)
print(altkümemi)
```

```
>>>

True
```

4.4 SÖZLÜKLER

Python'da sözlükler (*dictionaries*) bir koleksiyon türüdür. Sözlük koleksiyonları; sıralı, değiştirilebilir ve anahtar indeksli öğeleri barındırır. Her öğe bir anahtara (*key*) ve bir de değere (*value*) sahiptir. Sözlükler küme parantezleri **`{ }`** kullanılarak aşağıdaki örnekte görüleceği gibi tanımlanırlar.

```
personel = {
   "isim": "Adem",
   "soyisim": "Zengin",
   "yaş": 35
}
print(personel)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin', 'yaş': 35}
```

Sözlükte tanımlı öğelerin ilk unsuru anahtar (*key*), ikinci unsuru değer (*value*) olarak isimlendirilir. Anahtarlar string veri türündedir ve tırnak işareti içinde tanımlanırlar. Değerler ise herhangi bir veri türünde olabilir.

4.4.1 Sözlük Öğelerine Erişim

Bir sözlük içindeki bir öğenin değerine onun anahtarı kullanılarak erişilir. Bunun için sözlük adının yanına bir köşeli parantez `[]` açılır ve içine erişilmek istenen öğenin anahtarı yazılır. Aynı işlem `get()` metodu kullanılarak da yapılabilir.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

a = personel["yaş"]
b = personel.get("yaş")

print(a)
print(b)
```

```
>>>

35
35
```

4.4.2 Değer Değiştirme

Sözlükteki bir öğenin değeri, anahtarı kullanılarak değiştirilebilir.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

personel["yaş"] = 40

print(personel)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin', 'yaş': 40}
```

4.4.3 Öğeleri Döndürme

Sözlük öğeleri **`for`** döngüsü kullanılarak döndürülebilir. Sözlük öğelerine ait anahtar (*key*) ve değer (*value*) çiftlerini aşağıda gösterildiği şekilde ayrı ayrı döndürmek mümkündür.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

# Anahtarları döndürme
for a in personel:
  print(a)

print("-------")

# Değerleri döndürme
for a in personel:
  print(personel[a])
```

```
>>>

isim
soyisim
yaş
-------
Adem
Zengin
35
```

Değerleri döndürmenin bir başka yolu **`values()`** metodunu kullanmaktır. Diğer taraftan **`items()`** metodu hem anahtarların hem de

değerlerin döndürülmesini basitleştiren bir metot olarak kullanılabilir. Bu iki metodun kullanımı aşağıdaki örnekte görülmektedir.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

# values() metodu ile değerleri döndürme
for a in personel.values():
  print(a)

print("-------")

# items() metodu ile hem anahtar hem değer
# döndürme
for a, b in personel.items():
  print(a, b)
```

```
>>>

Adem
Zengin
35
-------
isim Adem
soyisim Zengin
yaş 35
```

4.4.4 Anahtar Sorgulama

Bir sözlükte bir anahtarın olup olmadığı “**in**” kodu ile sorgulanmaktadır.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

if "yaş" in personel:
  print("Evet, 'yaş' diye bir anahtar mevcut!")
```

```
>>>

Evet, 'yaş' diye bir anahtar mevcut!
```

4.4.5 Öğe Ekleme

Bir sözlüğe yeni öğe eklemek için yeni bir anahtar tanımlanmalı ve o anahtara bir değer atanmalıdır.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35
}

personel["ünvan"] = "Doktor"
print(personel)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin', 'yaş': 35,
'ünvan': 'Doktor'}
```

4.4.6 Öğe Silme

Sözlükten öğe silmenin farklı yöntemleri vardır. Örneğin, **`pop()`** metodu veya **`del`** kodu, anahtarı bilinen öğeyi sözlükten silmek için kullanılabilir. **`popitem()`** metodu ise son eklenen öğeyi sözlükten siler. Eğer tüm öğeler sözlükten silinmek isteniyorsa **`clear()`** metodu kullanılmalıdır. Bunların kullanımına ilişkin örnek kodlar aşağıda verilmiştir.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin",
  "yaş": 35,
  "ünvan": "Doktor"
}

personel.pop("ünvan")
print(personel)

del personel["yaş"]
print(personel)

personel.popitem()
print(personel)

personel.clear()
print(personel)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin', 'yaş': 35}
{'isim': 'Adem', 'soyisim': 'Zengin'}
{'isim': 'Adem'}
{}
```

4.4.7 Sözlük Kopyalama

Sözlükler atama yöntemiyle, yani eşitlik ifadesi (=) kullanılarak kopyalanamaz. Örneğin **`personel2 = personel`** şeklinde bir kullanım ortaya iki bağımsız sözlük çıkarmaz. Bu işlem sadece **`personel`** isimli sözlüğün bir yansımasını ortaya çıkarır. Bunun anlamı, **`personel`** isimli sözlükte yapılacak her değişiklik otomatik olarak **`personel2`**'de de görülecektir. Bir sözlükten iki bağımsız sözlük elde etmek için **`copy()`** metodu veya **`dict()`** fonksiyonu kullanılmalıdır.

```
personel = {
  "isim": "Adem",
  "soyisim": "Zengin"
}

personel2 = personel.copy()
print(personel2)

print("-------------------------------------")

personel3 = dict(personel)
print(personel3)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin'}
-------------------------------------
{'isim': 'Adem', 'soyisim': 'Zengin'}
```

4.4.8 İç İçe Sözlükler

Bir sözlük kendi içinde başka sözlükleri barındırabilir. Bu tür sözlüklere iç içe sözlükler (*nested dictionaries*) denir.

```
personeller = {
  "p1" : {
    "isim" : "Adem",
    "yaş" : 35
  },
  "p2" : {
    "isim" : "Ahmet",
    "yaş" : 27
  }
}

print(personeller)
```

```
>>>

{'p1': {'isim': 'Adem', 'yaş': 35}, 'p2': {'isim':
'Ahmet', 'yaş': 27}}
```

Hali hazırda mevcut olan sözlükler de tek bir sözlük içinde toplanabilir.

```
p1 = {
"isim" : "Adem",
"yaş" : 35
}

p2 = {
"isim" : "Ahmet",
"yaş" : 27
}

personeller = {"per1":p1,
               "per2":p2}

print(personeller)
```

```
>>>

{'per1': {'isim': 'Adem', 'yaş': 35}, 'per2':
{'isim': 'Ahmet', 'yaş': 27}}
```

4.4.9 Sözlük İnşası

Daha önce sözlük kopyalamada kullanılan **dict()** fonksiyonu aynı zamanda yeni bir sözlük inşası için de kullanılabilmektedir.

```
# Anahtarların string olarak verilmediğine
# dikkat ediniz.
personel = dict(isim="Adem",
                soyisim="Zengin",
                yaş=35)

print(personel)
```

```
>>>

{'isim': 'Adem', 'soyisim': 'Zengin', 'yaş': 35}
```

SORULAR

1. Aşağıdaki kod parçasında ? işaretli yere gelmesi gereken kod nedir?

```
isimler = ['Ali', 'Zeynep', 'Mahir']
if "Mahir" ? isimler:
    print("Mahir listede mevcuttur.")
```

2. Aşağıdaki Python kodunda verilen liste öğelerinden ikincisini ekrana yazdıracak komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

3. Aşağıdaki Python kodunda verilen liste öğelerinden birincisini “Ahmet” değeri ile değiştirecek komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

4. Aşağıdaki isimler listesinde **`append`** metodunu kullanarak “Ayşe” ismini listenin sonuna ekleyecek kodu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

5. Aşağıdaki isimler listesinde “Zeynep” ve “Mahir” arasına “Seda” ismini **`insert`** metodunu kullanarak ekleyiniz.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

6. Aşağıdaki isimler listesinden “Zeynep” ismini **`remove`** metodu ile silecek komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
```

7. Aşağıdaki isimler listesinin son öğesini negatif indeks numarası kullanarak ekrana yazdıracak komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
```

8. Aşağıdaki isimler listesinin ikinci, üçüncü ve dördüncü öğelerini bir indeks aralığı (ör. 6:9) kullanarak ekrana yazdıran komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
```

9. Aşağıdaki isimler listesinin kaç öğeye sahip olduğunu ekrana yazdıran komutu yazınız.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
```

10. Verilen demetteki birinci öğeyi ekrana yazdıracak doğru kodlamayı yapınız.

```
birDemet = ('elma', 'armut', 'dut')
```

11. Verilen demette kaç tane öğe olduğunu ekrana yazdıracak doğru kodlamayı yapınız.

```
birDemet = ('elma', 'armut', 'dut')
```

12. Verilen demetin son öğesini negatif indeks kullanarak ekrana yazdıracak doğru kodlamayı yapınız.

```
1 birDemet = ('elma', 'armut', 'dut')
```

13. Verilen demetin öğelerinden ikinci, üçüncü ve dördüncü öğeleri indeks aralığı kullanarak ekrana yazdıracak doğru kodlamayı yapınız.

```
1 birDemet = ('elma','kiwi','dut','muz','turp')
```

14. Verilen kümede "kivi" öğesi olup olmadığını sorgulayan ve sonucu **`True`** veya **`False`** olarak ekrana yazdıracak doğru kodlamayı yapınız.

```
1 birKüme = {'elma','kivi','dut','muz','turp'}
```

15. Meyveler kümesine "erik" öğesi ekleyecek ve tüm kümeyi ekrana yazdıracak doğru kodlamayı yapınız.

```
1 meyveler = {'elma','kivi','dut','muz'}
```

16. Meyveler kümesine "erik", "turp", "kayısı" öğelerini tek seferde ekleyecek ve tüm kümeyi ekrana yazdıracak doğru kodlamayı yapınız.

```
meyveler = {'elma','kivi','dut','muz'}
```

17. Meyveler kümesinden “dut” öğesini silecek ve tüm kümeyi ekrana yazdıracak doğru kodlamayı yapınız.

```
meyveler = {'elma','kivi','dut','muz'}
```

18. Verilen sözlükteki “hobi” isimli anahtarın taşıdığı değeri get metodunu kullanarak ekrana yazdıran komutu giriniz.

```
üye = {
    "yaş":45,
    "cinsiyet":"erkek",
    "hobi":"kitap"
    }
```

19. Verilen sözlükteki “hobi” isimli anahtarın taşıdığı değeri “bisiklet” ile değiştiren komutu giriniz.

```
üye = {
    "yaş":45,
    "cinsiyet":"erkek",
    "hobi":"kitap"
```

```
5       }
```

20. Verilen sözlüğe yeni bir anahtar-değer (ör. eğitim:lisans) ikilisi ekleyiniz.

```
1  üye = {
2       "yaş":45,
3       "cinsiyet":"erkek",
4       "hobi":"kitap"
5       }
```

21. Verilen sözlükten “yaş” öğesini pop metodu ile siliniz.

```
1  üye = {
2       "yaş":45,
3       "cinsiyet":"erkek",
4       "hobi":"kitap"
5       }
```

22. Verilen sözlük öğelerinin tamamını `clear` metodu ile siliniz.

```
1  üye = {
2       "yaş":45,
3       "cinsiyet":"erkek",
4       "hobi":"kitap"
5       }
```

23. 10 kelimelik bir İngilizce-Türkçe sözlük programı yazınız. Bu program çalıştırıldığında kullanıcıdan bir kelime girmesini istesin. Girilen kelime öncelikle sözlükte mevcut mu değil mi kontrol edilsin. Mevcut değilse “Kelime bulunamadı!” uyarısı ekrana yazdırılsın. Mevcutsa kelimenin sözlükteki karşılığı ekrana yazdırılsın.

CEVAP ANAHTARI

1. in

2.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
print(isimler[1])
```

3.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
isimler[0] = 'Ahmet'
```

4.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
isimler.append('Ayşe')
```

5.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
isimler.insert(2,'Seda')
```

6.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
isimler.remove('Zeynep')
```

7.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
print(isimler[-1])
```

8.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
print(isimler[1:4])
```

9.

```
isimler = ['Ali', 'Zeynep', 'Seda', 'Mahir']
print(len(isimler))
```

10.

```
birDemet = ('elma', 'armut', 'dut')
print(birDemet[0])
```

11.

```
birDemet = ('elma', 'armut', 'dut')
print(len(birDemet))
```

12.

```
birDemet = ('elma', 'armut', 'dut')
print(birDemet[-1])
```

13.

```
birDemet = ('elma','kivi','dut','muz','turp')
print(birDemet[1:4])
```

14.

```
birKüme = {'elma','kivi','dut','muz','turp'}
print('kivi' in birKüme)
```

15.

```
meyveler = {'elma','kivi','dut','muz'}
meyveler.add('erik')
print(meyveler)
```

16.

```
meyveler = {'elma','kivi','dut','muz'}
meyveler.update(['erik','turp','kayısı'])
print(meyveler)
```

17.

```
meyveler = {'elma','kivi','dut','muz'}
meyveler.remove('dut')
print(meyveler)
```

18.

```
üye = {
     "yaş":45,
     "cinsiyet":"erkek",
     "hobi":"kitap"
     }
print(üye.get("hobi"))
```

19.

```
üye = {
     "yaş":45,
     "cinsiyet":"erkek",
     "hobi":"kitap"
     }
üye["hobi"]="bisiklet"
```

20.

```
üye = {
     "yaş":45,
     "cinsiyet":"erkek",
     "hobi":"kitap"
     }
üye["eğitim"]="lisans"
```

21.

```
üye = {
     "yaş":45,
     "cinsiyet":"erkek",
     "hobi":"kitap"
     }
üye.pop("yaş")
```

22.

```
üye = {
     "yaş":45,
     "cinsiyet":"erkek",
     "hobi":"kitap"
     }
üye.clear()
```

23.

```
# BİR SÖZLÜK PROGRAMI

# Sözlük verisi
sözlük = {
  "go": "gitmek",
  "come": "gelmek",
  "face": "yüz",
  "hot": "sıcak",
  "fear": "korku",
  "speed": "hız",
  "nice": "hoş",
  "hard": "zor",
  "deep": "derin",
  "cow": "inek",
}

# Kullanıcıdan kelime girmesi istenir.
ara = input("Bir kelime giriniz:")

# Kelimenin sözlükteki varlığı denetlenir.
if ara in sözlük:
    print(sözlük[ara])
else:
    print("Kelime Bulunamadı!")
```

5. DÖNGÜLER

Python iki tür döngü komutuna sahiptir. Bunlar **`for`** ve **`while`** komutlarıdır.

5.1 `for` Döngüsü

Python'da **`for`** döngüsü, bir koleksiyon (*list*, *tuple*, *set* veya *dictionary*) içindeki öğeleri veya bir string içindeki karakterleri döndürmek için kullanılabilir. Öğeleri veya karakterleri döndürmedeki amaç, üzerlerinde bir dizi eylem gerçekleştirmektir. Döngüdeki çevrim sayısı koleksiyonlarda öğe sayısına, stringlerde karakter sayısına eşittir. **`for`** döngüsü yazarken, her eylemi uygun şekilde girintilemeyi unutmamak gerekir. Aksi takdirde bir girinti hatası (**`IndentationError`**) oluşur.

Aşağıda verilen **`for`** döngüsü örneğinde 3 öğesi bulunan bir koleksiyon döndürülmektedir. 3 çevrimin gerçekleştiği bu döngüde birinci çevrimde Ali ismi yazdırılır, ikinci çevrimde Zeynep ve son çevrimde de Mahir ismi yazdırılmış olur.

```
1 isimler = ['Ali', 'Zeynep', 'Mahir']
2 for x in isimler:
3     print(x)
```

```
>>>

Ali
Zeynep
Mahir
```

for döngüsü string için kullanıldığında stringin her karakteri bir öğe konumunda olur ve 0'dan başlayıp ilerleyen indeks ile harfler bir bir çağrılır.

```
1 for x in "zeynep":
2     print(x)
```

```
>>>

z
e
y
n
e
p
```

5.1.1 `break` İfadesi

Bir döngüde **break** ifadesi çalıştırıldığında döngüden çıkış yapılmış olur. Yapılan girintilemeye göre döngüden sonra gelen işleme geçilmiş olur. Bu ifade, belli bir şart sağlandığında döngüden çıkış için kullanılır.

```
sayılar = [0, 512, 6, -8, 3]

for x in sayılar:
  if (x < 0):
    print("Negatif sayı bulundu!")
    break
  print(x)
print("Buradan devam...")
```

```
>>>

0
512
6
Negatif sayı bulundu!
Buradan devam...
```

5.1.2 `continue` İfadesi

Bir döngüde **`continue`** ifadesi çalıştırıldığında kendinden sonraki kod çalıştırılmadan aynı döngü içinde bir sonraki çevrime geçilir. Aşağıda sayılar listesinin öğeleri yazdırılmaktadır. Döngü negatif bir sayıya denk geldiğinde **`continue`** komutu çalışmakta ve **`continue`**'dan sonra yazılan kodlar çalıştırılmadan **`sayılar`** listesindeki bir sonraki öğeye geçilmektedir. Dolayısı ile listedeki -8 öğesi çıktılar arasında yer almamıştır.

```
sayılar = [0, 512, 6, -8, 3]

for x in sayılar:
  if (x < 0):
    continue
  print(x)
```

```
>>>

0
512
6
3
```

5.1.3 `range()` Fonksiyonu

Bir dizi kodu belirli sayıda döndürüp tekrar tekrar çalıştırmak için **`for`** döngüsü **`range()`** fonksiyonu ile birlikte kullanılabilir. **`range()`** fonksiyonu, varsayılan olarak 0'dan başlayan ve 1 artarak (varsayılan olarak) ilerleyen bir sayı dizisi verir ve belirlenmiş bir sayı ile de biter.

```
for x in range(3):
    print(x)
```

```
>>>

0
1
2
```

Varsayılan olarak 0'dan başlayan **`range()`** fonksiyonu istenirse farklı bir değerden başlatılabilir. Bunun için fonksiyona bir argüman daha girmek

gerekir. **range(a,b)** şeklinde yazılan fonksiyon **a**'dan başlar (**a** dahil) ve **b**'ye kadar (**b** hariç) 1 artarak ilerler. Bu fonksiyondaki varsayılan artış değeri de istenirse değiştirilebilir. Bu durumda da fonksiyon, 3 argümanlı olarak kullanılır. Örneğin, **range(a,b,c)** fonksiyonu **a**'dan başlar ve **b**'ye kadar **c** artarak ilerler.

```
1  for x in range(2,8,2):
2    print(x)
```

```
>>>
2
4
6
```

5.1.4 pass İfadesi

Python'da bir **for** girintisi oluşturulduktan sonra içi boş bırakılamaz. Eğer herhangi bir sebeple boş geçilmek isteniyorsa bu durumda girinti içine **pass** komutu girilerek döngü boş geçilebilir. **pass** komutu benzer şekilde fonksiyonlar için de aynı amaçla kullanılabilir.

```
1  sayılar = [0, 512, 6, -8, 3]
2
3  for x in sayılar:
4      pass
5
6  print("Buradan devam...")
```

```
>>>

Buradan devam...
```

5.1.5 İç İçe Döngüler

Python'da bir döngü içine başka döngüler yerleştirmek mümkündür. Bir **`for`** girintisi içinde yeni bir **`for`** girintisi oluşturulduğunda içi içe iki döngü elde edilmiş olur. İçteki döngü dıştaki döngünün her bir çevrimi için tam bir döngü yapar. İç içe döngüler örneğin liste-içinde-listelerin öğelerine erişmek için kullanılabilir.

```
1  ödevGrupları = [["Sümeyye", "Meryem"],
2                  ["Emre", "Yunus"],
3                  ["Akif", "Mehmet", "Burak"]]
4
5  # Dış döngü, grupları sırasıyla grup değişkenine
6  # atayacaktır.
7  for grup in ödevGrupları:
8    # İç döngü, sırasıyla gruplardaki isimleri
9    # isim değişkenine atayacaktır.
10   for isim in grup:
11     print(isim)
```

```
>>>

Sümeyye
Meryem
Emre
Yunus
Akif
Mehmet
Burak
```

5.2 `while` Döngüleri

Python'da **`while`** döngüleri verilen koşul **`True`** değerini sağladıkça devam eder. **`while`** girintisi içindeki kod eğer koşul sağlanmamışsa çalıştırılmaz. Yani önce koşul denetlenir. **`True`** değeri alınırsa kod çalıştırılır. **`True`** değeri alınmaz ise program **`while`** girintisinden sonraki kodlara yönelir. **`for`** döngüsü başlığı altında verilen **`break`** ve **`continue`** komutları **`while`** döngüleri için de aynı işleve sahiptirler.

```
1  a = 1
2  while a < 4:
3    print(a)
4    a += 1
```

```
>>>

1
2
3
```

5.3 Sonsuz Döngü

Sonsuz bir döngü, asla sona ermeyen bir döngüdür. Döngünün koşulları sona ermesini engellediğinde sonsuz döngüler oluşur. Bunun nedeni, döngü içindeki koşullu ifadedeki bir yazım hatası veya yanlış mantık olabilir. Böyle bir durumla karşılaşıldığında, Python programını sonlandırmak için Ctrl ve C tuşlarına birlikte basılır.

SORULAR

1. Verilen listedeki öğeleri sırayla ekrana yazdıracak **`for`** döngüsü nasıl yazılır?

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

2. Verilen listedeki öğeleri, "Zeynep" i atlayarak sırayla ekrana yazdıracak **`for`** döngüsü nasıl yazılır?

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

3. Python'da range fonksiyonunu kullanarak ekrana 10 kez "Dikkat!" mesajını yazdırınız.

4. Verilen listedeki öğeleri sırasıyla yazdırma işlemi yapılırken sıra "Zeynep" öğesine geldiğinde "Zeynep" ekrana yazdırılmadan döngüden çıkılacak şekilde bir kodlama yapınız.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
```

5. Sıfırdan 10'a kadar rakamları (10 hariç) ekrana yazdıracak bir while döngüsü oluşturunuz.

6. 5. sorudaki döngüde sayı 5'e eşit olduğunda döngü sonlanacak şekilde kodu yeniden düzenleyiniz.

7. 5. sorudaki döngüde sayı 5'e eşit olduğunda 5'i yazdırmadan atlayıp diğer sayıya geçecek şekilde kodu yeniden düzenleyiniz.

8. Bu soruda KIYAS ve DENETİM bölümünde yazdığınız oyunun daha gelişmişini yazmanız isteniyor. Oyun, yine bir sayı tahmin etme oyunudur. Programın kaynak kodunda tahmin edilecek sayı için bir değişken tanımlayın. Bu değişkene **`gizliSayı`** ismini verelim. Oyunu yazan kişi değişkene bir sayı ataması yapar. Yarışmacıların görevi bu sayıyı tahmin etmektir. Program çalıştırıldığında yarışmacıdan ekrana bir tam sayı girmesini ister. Program yarışmacının girdiği sayıyı **`gizliSayı`** ile kıyaslar. Yarışmacı burada 3 farklı durum ile karşılaşabilir. Girdiği sayı, **`gizliSayı`**'ya eşit olabilir. Bu durumda ekranda "Tebrikler!" yazdığını görür. Girdiği sayı **`gizliSayı`**'dan küçük olabilir. Bu durumda ekranda "Yukarı" yazdığını görür. Tersi durumda ise ekranda "Aşağı" yazdığını görür. Yarışmacı doğru tahmin yapana kadar program sonlanmaz. Yarışmacı yanlış cevap verdiğinde "Aşağı" ya da "Yukarı" çıktısı verilir ve tekrar yeni bir tahminde bulunması istenir. Her denemede ekrana kaçıncı denemeyi yaptığının bilgisi de yazdırılır. Örneğin "DENEME : 3" gibi. Yarışmacı doğru cevabı verdiğinde ekrana "Tebrikler!" ve "X denemede buldunuz" çıktısı yazdırılır. Doğru sonuca kaç denemede ulaştığı bir kağıda not edilir. Sonra başka yarışmacılar oyunu oynar. Onların da kaç denemede doğru sonuca ulaştıkları not edilir. En az denemede **`gizliSayı`**'yı bulan yarışmacı oyunu kazanır.

CEVAP ANAHTARI

1.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
for a in isimler:
    print(a)
```

2.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
for a in isimler:
    if a == "Zeynep":
        continue
    print(a)
```

3.

```
for a in range(10):
    print("Dikkat!")
```

4.

```
isimler = ['Ali', 'Zeynep', 'Mahir']
for a in isimler:
    if a == "Zeynep":
        break
    print(a)
```

5.

```
a = 0
while a < 10:
    print(a)
    a += 1
```

6.

```
a = 0
while a < 10:
    print(a)
    if a == 5:
        break
    a += 1
```

7.

```
a = 0
while a < 10:
    if a == 5:
        a += 1
        continue
    print(a)
    a += 1
```

8.

```
# BU BİR SAYI TAHMİN ETME OYUNUDUR.

# Gizli sayı tanımlanır.
gizliSayı = 34

# Diğer gerekli değişkenler tanımlanır.
sonuç = False
deneme = 0

# Döngünün kaç çevrim olacağı bilinmediği için
# while döngüsü kullanılır.
while sonuç==False:
    deneme += 1
    print ("DENEME : " + str(deneme))

    # Yarışmacının tahmini istenir.
    tahmin = input("Bir tam sayı giriniz:")

    # Alınan sayı integer formatına
    # dönüştürülür.
    tahmin = int(tahmin)

    # Kıyaslama işlemi yapılır ve sonuçlar
    # yazdırılır.
    if tahmin > gizliSayı:
        sonuç = False
        print("Aşağı")
    elif tahmin < gizliSayı:
        sonuç = False
        print("Yukarı")
    else:
        sonuç = True
        print("Tebrikler!")
        print(str(deneme)+ " denemede buldunuz")
```

6. STRING

Bilgisayar biliminde, yazı karakteri dizilerine *string* denmektedir. Stringler herhangi bir uzunlukta olabilir ve harf, sayı, sembol, boşluk, sekme (tab), yeni satır gibi karakterler içerebilir. String verisi program içinde çift tırnak veya tek tırnak arasına alınır. String türü değişkenler veya string değerleri **`print()`** fonksiyonu ile Python Shell'e (konsol) yazdırılabilir.

```
1  a = "Merhaba"
2  print(a)
3  print('Arkadaşlar')
```

```
>>>

Merhaba
Arkadaşlar
```

6.1 Çok Satırlı Stringler

Stringler kimi zaman oldukça uzun olabilmektedir. Tek satır olarak programa girilmek istenen stringler tek tırnak veya çift tırnak arasında girilebilir. Ancak çok satırlı bir string girilmek istenirse ve kod editöründe çok satırlı olarak görülmek isteniyorsa bu durumda üç tırnaklı format

kullanılmalıdır. Bu format tek tırnak ile (**'''metin'''**) uygulanabildiği gibi çift tırnak ile de (**"""metin"""**) uygulanabilmektedir.

```
a = """
Arkadaş! Yurduma alçakları uğratma, sakın;
Siper et gövdeni, dursun bu hayasızca akın.
Doğacaktır sana va'dettigi günler Hakk'ın...
Kim bilir, belki yarın, belki yarından da yakın.
"""

print(a)
```

```
>>>

Arkadaş! Yurduma alçakları uğratma, sakın;
Siper et gövdeni, dursun bu hayasızca akın.
Doğacaktır sana va'dettigi günler Hakk'ın...
Kim bilir, belki yarın, belki yarından da yakın.
```

Bir stringin birden çok satıra bölünmesi ters eğik çizgi (\) kullanımı ile de mümkündür. Ancak bu yöntem aşağıda görüldüğü gibi shell'de çok satırlı bir çıktı oluşturmaz.

```
a = "Arkadaş! Yurduma alçakları uğratma, sakın;\
Siper et gövdeni, dursun bu hayasızca akın.\
Doğacaktır sana va'dettigi günler Hakk'ın...\
Kim bilir, belki yarın, belki yarından da yakın."

print(a)
```

```
>>>

Arkadaş! Yurduma alçakları uğratma, sakın;Siper et
gövdeni, dursun bu hayasızca akın.Doğacaktır sana
va'dettigi günler Hakk'ın...Kim bilir, belki yarın,
belki yarından da yakın.
```

6.2 String İndeksi

String, listeler gibi indeksli ve yinelenebilir (*iterable*) veri türüdür. Liste öğelerine erişim için kullanılan kodlar string karakterlerine erişmek için de kullanılmaktadır. Tek bir karaktere erişmek için köşeli parantez kullanılır. Örneğin **`isim[3]`** ile **`isim`** değişkeninde yüklü stringin 3. sıradaki karakterine erişilir. Sıralamanın sıfırdan başladığı unutulmamalıdır.

Listelerde olduğu gibi indeks numarası olarak negatif rakamlar da kullanılabilmektedir. Pozitif rakamlar baştan sona doğru bir indeksleme yaparken, negatif rakamlar sondan başa doğru bir indeksleme yaparlar. Örneğin **`isim[-1]`** kullanımı ile **`isim`** değişkeninde yüklü stringin sondan 1. sıradaki karakterine erişilir.

Eğer bir stringin bir kesitine erişilmek isteniyorsa bu durumda köşeli parantez, iki nokta ile ayrılan iki değer alır. Örneğin **`isim[2:4]`** kullanımı, **`isim`** değişkeninde yüklü stringin 2. karakterinden (2 dahil) 4.

karakterine (4 hariç) kadarlık kesitini verir. Eğer iki noktanın sol tarafına rakam girilmez ise bu “en baştan” anlamına gelir. Eğer sağ tarafına rakam girilmez ise bu da “sonuna kadar” anlamına gelir. Örneğin **`isim[:4]`** kullanımı en baştan 4. karaktere kadar (4 hariç) olan kesiti verir. Örneğin **`isim[2:]`** kullanımı 2. karakterden başlayıp (2 dahil) stringin sonuna kadar olan kesiti verir.

```
isim = 'Amine'

print(isim[0])
print(isim[3])
print(isim[-1])
print(isim[2:4])
print(isim[:4])
print(isim[2:])
```

```
>>>

A
n
e
in
Amin
ine
```

6.3 String Eklemleme

Python, birden fazla stringin birbirine eklemlenerek tek bir string oluşturmasını artı (+) işaretiyle sağlar. Başarılı bir eklemleme için (+) işaretine iletilen değişkenlerin tamamı string türünde olmalıdır. Farklı tür değişkenler iletilirse, bu durumda Python bir hata durumu bildirir.

```
# String eklemleme

birinci = "Dikkat "
ikinci = "Et"

sonuç1 = birinci + ikinci
sonuç2 = birinci + ikinci + "!"

print(sonuç1)
print(sonuç2)
```

```
>>>

Dikkat Et
Dikkat Et!
```

6.4 String Uzunluğu

String uzunluğundan kasıt bir stringin kaç karakterden oluştuğudur. String uzunluğu, listelerin öğe sayısını bildiren **`len()`** fonksiyonu kullanılarak elde edilir.

```
ayet = """Bugün doğru olanlara, doğruluklarının
fayda vereceği gündür!"""

print(len(ayet))
```

```
>>>

60
```

Stringin bir karakterini veya bir kesitini elde etmek için kullanılacak indeks numaralarının string uzunluğunu aşmaması gerekir. Aksi takdirde

IndexError hatası ile karşılaşılır. Dolayısı ile **len()** fonksiyonu ile string uzunluğunun eldesi önem taşımaktadır. İndekslerin 0'dan başladığı unutulmamalıdır. Bu durumda bir stringin son karakterine erişmek için indeks değeri olarak stringin uzunluğunun bir eksiği olan rakam girilmelidir. Örnekte 60 karakterden oluşan bir stringin son karakterine (son karakter ünlem işaretidir) 59 indeks numarası ile erişildiği görülmektedir. 60 indeks numarası girildiğinde program kodunun 4.satırı için bir hata mesajı alınır.

```
1  ayet60Karakter = """Bugün doğru olanlara,
   doğruluklarının fayda vereceği gündür!"""
2
3  print(ayet60Karakter[59])
4  print(ayet60Karakter[60])
```

```
>>>

!
Traceback (most recent call last):
  File "C:\Egitim\ders6.py", line 4, in <module>
    print(ayet60Karakter[60])
IndexError: string index out of range
```

6.5 Kaçış Karakteri

String girişi yaparken bazı yazı karakterlerinin doğrudan kullanımı mümkün değildir. Örneğin tek tırnak ve çift tırnak karakterleri program ile iletişim amaçlı kullanılmaktadır. Bunlar yazılan kod içinde stringin başladığı ve sona erdiği noktaları bildirmektedir. Bu karakterler, dildeki gerçek işlevleri doğrultusunda kullanılmak istendiğinde, yani örneğin bir kelime özellikle tırnak içine alınmak istendiğinde, bu niyet programa

bildirilmelidir. İşte programa bu niyeti bildirmek için kaçış karakteri kullanılır.

Python'da kaçış karakteri olarak ters eğik çizgi (\) kullanılır. Kaçış karakterinin kullanıldığı başlıca karakterler aşağıda verilmiştir.

- Çift tırnak (\")
- Tek tırnak (\')
- Ters eğik çizgi (\\)
- Yeni satıra geç (\n)
- Tab (\t)

```
1 print("Dizeler için \"string\" ifadesini
  kullanacağız.")
2 print()
3 print("A\tB\tC\tD\tE")
4 print()
5 print("Mühendislik\nFakültesi")
6 print()
7 print("DosyaKonumu = C:\\Users\\Public\\Desktop")
```

```
>>>

Dizeler için "string" ifadesini kullanacağız.

A       B       C       D       E

Mühendislik
Fakültesi

DosyaKonumu = C:\Users\Public\Desktop
```

6.6 Karakterlerin Sıralı Çağırımı

Bir stringi oluşturan karakterlerin sırasıyla çağrılması **for** ve **in** kodları ile yapılır. Bu kodlar, stringler üzerinde başta arama yapmak olmak üzere çeşitli işlemlerin yapılmasını sağlar.

```
1  # Kameranın okuduğu yazıda 0 ve O denetimi
2  okuma1 = "KAMY0N"
3  for x in okuma1:
4      print(x)
5      if x == "0":
6          print("Okuma hatası bulundu!")
```

```
>>>

K
A
M
Y
0
Okuma hatası bulundu!
N
```

6.7 `in` İfadesi

Python stringleri için kullanılan **in** ifadesi, aranan bir karakterin veya bir karakter dizisinin arama yapılan stringde bulunup bulunmadığını denetler. Bulunuyorsa **True**, bulunmuyorsa **False** dönütü verir.

```
1  # Kameranın okuduğu yazının kontrolü
2  okuma1 = "KAMY0N"
3  print("0" in okuma1)
4  print("KAM" in okuma1)
5  print("KAMYON" in okuma1)
```

```
>>>

True
True
False
```

6.8 String Metotları

lower()

Bu metot verilen stringin içerdiği tüm büyük harfleri küçük harfe çevirir.

```
1  hitap1 = "Merhaba Arkadaşlar, "
2  hitap2 = "NASILSINIZ?"
3
4  print(hitap1.lower() + hitap2.lower())
```

```
>>>

merhaba arkadaşlar, nasilsiniz?
```

upper()

Bu metot verilen stringin içerdiği tüm küçük harfleri büyük harfe çevirir.

```
1 hitap1 = "Merhaba Arkadaşlar, "
2 hitap2 = "nasılsınız?"
3
4 print(hitap1.upper() + hitap2.upper())
```

```
>>>

MERHABA ARKADAŞLAR, NASILSINIZ?
```

title()

Bu metot verilen stringin içeriğindeki her kelimenin ilk harfini büyük harf yapar.

```
1 hitap1 = "Merhaba Arkadaşlar, "
2 hitap2 = "nasılsınız?"
3
4 print(hitap1.title() + hitap2.title())
```

```
>>>

Merhaba Arkadaşlar, Nasılsınız?
```

split()

Bu metot verilen stringi parçalara bölerek bir liste haline getirir. Eğer bu metot argümansız kullanılırsa parçalama işlemini, string içindeki boşluk karakterlerinin bulunduğu noktalardan yapar. Eğer argüman ile

kullanılırsa verilen argümanın string içinde bulunduğu noktalardan stringi parçalara ayırır.

```
hitap1 = "Merhaba Arkadaşlar"

parçalar1 = hitap1.split()
parçalar2 = hitap1.split("a")

print(parçalar1)
print(parçalar2)
print(hitap1)
```

```
>>>

['Merhaba', 'Arkadaşlar']
['Merh', 'b', ' Ark', 'd', 'şl', 'r']
Merhaba Arkadaşlar
```

join()

Bu metot bir listedeki stringleri sırasıyla uç uca ekleyerek tek bir string haline getirir. `join()` metodu öğelerini birleştireceği listeyi argüman olarak alır. Bu string metodu, birleştirilecek öğelerin arasına konulacak string üzerinden çalıştırılır.

```
1  parçaAdres = ["Serdivan", "Sakarya", "Türkiye"]
2
3  bütünAdres1 = " ".join(parçaAdres)
4  bütünAdres2 = "/".join(parçaAdres)
5
6  print(bütünAdres1)
7  print(bütünAdres2)
```

```
>>>

Serdivan Sakarya Türkiye
Serdivan/Sakarya/Türkiye
```

strip()

Bu metot, argüman olarak verilmiş karakterlerin bir stringin başından ve sonundan silinmesini sağlar. Eğer argüman verilmemiş ise silme işlemi boşluk karakteri için gerçekleştirilir. Silme işlemi verilen karakterin ilk defa bulunamadığı yere kadar yapılır. Örnekte görüldüğü gibi 4. satırdaki silme işlemi, baştan ve sondan boşlukların bittiği ilk noktaya kadar devam etmiştir. Dolayısıyla iki kelime arasındaki boşluk silinmemiştir. 13. satırdaki silme işlemi bir fark oluşturmamıştır. Çünkü baştan ve sondan silinmesi istenen karakter (`'*'`), ilk karakterlerde bulunamadığı için silme işlemi son bulmuştur.

```
isim1 = "  Adem Zengin        "

# Burada boşluk karakteri siliniyor
silinmiş = isim1.strip()
print(silinmiş)

isim2 = "..*..Adem Zengin.*...."

# Burada "." karakteri siliniyor
print(isim2.strip('.'))

# Burada "*" karakteri siliniyor
print(isim2.strip('*'))

# Burada "*" ve "." karakteri siliniyor
print(isim2.strip('*.'))
```

```
>>>

Adem Zengin
*..Adem Zengin.*
..*..Adem Zengin.*....
Adem Zengin
```

replace()

Bu metot virgülle ayrılmış iki string argüman alır. Üzerinde çalıştırılan string içinde bulduğu ilk argümanı ikinci argüman ile değiştirir.

```
konum = "kirikkale"
düzeltilmiş = konum.replace('i','ı')
print(düzeltilmiş)
```

```
>>>

kırıkkale
```

find()

Bu metot bir string argüman alır. Üzerinde çalıştırıldığı string içinde verilen argümanı arar. İlk kez geçtiği yeri (indeksi) dönüt olarak verir. Eğer aranan argüman bulunamaz ise dönüt olarak -1 rakamını verir.

```
konum = "kirikkale"
iNerede= konum.find('i')
jNerede= konum.find('j')
print(iNerede)
print(jNerede)
```

```
>>>

1
-1
```

format()

Python'da stringler ve rakamlar artı (+) işareti ile doğrudan birleştirilemez. Çünkü bu iki veri türü farklı formatlardadır. **`format()`** metodu birleştirme işlemini gerçekleştirmenin bir yolunu sunmaktadır. Bu metoda istenilen sayıda argüman girilebilir. Metodun işlevi, argümanlara

gerekli format değişikliğini uyguladıktan sonra üzerinde çalıştığı string içerisine yerleştirmektir. Yerleştirilecek pozisyonlar önceden string içerisinde küme parantezi **{}** kullanılarak belirlenir.

```
1 puan = 85
2 metin = "Öğrenci {} aldı."
3 sonHal = metin.format(puan)
4 print(sonHal)
```

```
>>>

Öğrenci 85 aldı.
```

Bir string içinde birden fazla pozisyon belirlenebilir. Böyle durumlarda **format()** metodu argümanları sırasıyla bu pozisyonlara yerleştirir.

```
1 puan = 85
2 numara = 200510032
3 metin = "{} numaralı öğrenci {} aldı."
4 sonHal = metin.format(numara,puan)
5 print(sonHal)
```

```
>>>

200510032 numaralı öğrenci 85 aldı.
```

Çok sayıda argüman ve pozisyon bulunduğunda sıralama işlemi zor bir hal alabilir. Daha kolay bir argüman-pozisyon eşleşmesi için pozisyonlara indeks numaraları girilebilmektedir. Örneğin string içine **{2}** girildiğinde o pozisyona **format()** metodundaki 3. argüman yerleşir.

```
1 puan = 85
2 numara = 200510032
3 sınıf = 4
4 metin = "{1} numaralı {0}. sınıf öğrencisi {2}
  aldı."
5 sonHal = metin.format(sinif,numara,puan)
6 print(sonHal)
```

```
>>>

200510032 numaralı 4. sınıf öğrencisi 85 aldı.
```

Pozisyonlara indeks numarası girildiği gibi isim de verilebilmektedir. Bu da metodun desteklediği bir diğer kolaylıktır.

```
1 puan = 85
2 numara = 200510032
3 sınıf = 4
4 metin = "{no} numaralı {snf}. sınıf öğrencisi {p}
  aldı."
5 sonHal = metin.format(p=puan,no=numara,snf=sınıf)
6 print(sonHal)
```

```
>>>

200510032 numaralı 4. sınıf öğrencisi 85 aldı.
```

6.9 Diğer String Metotları

Metot	İşlev
`capitalize()`	İlk karakteri büyük harf yapar.
`center()`	Ortalanmış bir string döndürür.
`count()`	Verilen argümanın stringde kaç kez geçtiğini döndürür.
`expandtabs()`	Tab (\t) işaretinin bıraktığı boşluk sayısını ayarlar.
`isalnum()`	String alfanümerik (sadece rakamlar ve harfler) karakterlerden oluşuyorsa `True` döndürür.
`isalpha()`	String sadece a'dan z'ye harflerden oluşuyorsa `True` döndürür.
`isidentifier()`	String geçerli bir değişken ismi ise `True` değeri verir.
`islower()`	String tamamen küçük harflerden oluşuyorsa `True` döndürür.
`isnumeric()`	String tamamen rakamlardan oluşuyorsa `True` döndürür.
`isprintable()`	String yazdırılabilir karakterlerden oluşuyorsa `True` döndürür. Ör. yeni satır (\n) karakteri yazdırılamaz.

istitle()	String kelimelerinin ilk harfleri büyük diğer harfleri küçükse **True** döndürür.
isupper()	String tamamen büyük harflerden oluşuyorsa **True** döndürür.
ljust()	Stringi sola yaslı olarak geri döndürür.
rfind()	Aranan bir argümanın stringde bulunduğu son pozisyonu döndürür.
rjust()	Stringi sağa yaslı olarak geri döndürür.
splitlines()	Stringi satır başlarından bölerek liste olarak döndürür.
startswith()	String verilen argümanla başlıyorsa **True** döndürür.
zfill()	Stringin başına belirli sayıda sıfır ekleyerek döndürür.

SORULAR

1. Herhangi bir string oluşturarak bunun kaç karakterden meydana geldiğini ekrana yazdıran bir Python kodu yazınız.

2. Verilen stringin ilk karakterini ekrana yazdıracak kodu ekleyiniz.

```
1 metin = "abcçde"
```

3. Verilen stringin 3. , 4. ve 5. karakterlerini ekrana yazdıracak kodu ekleyiniz.

```
1 metin = "abcçde"
```

4. Verilen stringin başında ve sonunda bulunan boşlukları temizleyerek ekrana yazdıran kodu yazınız.

```
1 metin = "  abcçde "
```

5. Verilen stringin tüm karakterlerini büyük harf yaparak ekrana yazdıracak kodu ekleyiniz.

```
1 metin = "abcçde"
```

6. Verilen stringin tüm karakterlerini küçük harf yaparak ekrana yazdıracak kodu ekleyiniz.

```
1  metin = "AbCçDe"
```

7. Türkçe karakter sorunu sonucu ortaya çıkan aşağıdaki string örneğindeki "ý" karakterlerini "ı" karakteri ile değiştirecek ve düzeltilmiş string'i ekrana yazdıracak kodu veriniz.

```
1  metin = "kýlýbýk"
```

8. Konsol çıktısı göz önünde bulundurulduğunda aşağıdaki Python kodunda ? işaretli yere yazılması gereken kod ne olur?

```
1  puan = 85
2  metin = "Öğrenci ? aldı."
3  sonHal = metin.format(puan)
4  print(sonHal)
```

```
>>>

Öğrenci 85 aldı.
```

9.

I. Üç tırnak (""" """)
II. \n
III. \t
IV. \

Bir string birden fazla satır olarak konsola yazdırılmak istenirse hangi karakter(ler) kullanılabilir?

A. Yalnız I
B. Yalnız II
C. II ve III
D. II ve IV
E. I ve II

10. Aşağıda bir haber metni string olarak verilmiştir. Metin içinde geçen tarihleri algılayıp ekrana yazdıracak bir Python kodu yazınız. İsterseniz içinde çeşitli rakamlar ve tarihlerin geçtiği başka bir metin üzerinde de çalışabilirsiniz.

```
haber = """Ankara'da 2018,2019 ve 2020 yılının
         Aralık ayına kadar bulaşıcı hastalıktan
         vefat sayıları ortaya çıktı. Ankara'da
         2018 yılında bulaşıcı hastalıklardan
         67, 2019 yılında 21, 2020 yılının
         Aralık ayı başına kadar ise
         3 bin 561 kişi hayatını kaybetti."""
```

CEVAP ANAHTARI

1.

```
metin = "abcçdefgğhıijklmnoöprsştuüvyz"
print(len(metin))
```

2.

```
metin = "abc"
print(metin[0])
```

3.

```
metin = "abc"
print(metin[2:5])
```

4.

```
metin = "  abcçde "
print(metin.strip())
```

5.

```
metin = "abcçde"
print(metin.upper())
```

6.

```
metin = "AbCçDe"
print(metin.lower())
```

7.

```
metin = "kýlýbýk"
print(metin.replace('ý','ı'))
```

8.

```
puan = 85
metin = "Öğrenci {} aldı."
sonHal = metin.format(puan)
print(sonHal)
```

9. Doğru cevap “D” seçeneğidir.

10.

```
haber = """Ankara'da 2018,2019 ve 2020 yılının
        Aralık ayına kadar bulaşıcı hastalıktan
        vefat sayıları ortaya çıktı. Ankara'da
        2018 yılında bulaşıcı hastalıklardan
        67, 2019 yılında 21, 2020 yılının
        Aralık ayı başına kadar ise
        3 bin 561 kişi hayatını kaybetti."""

# Parçalama işlemine geçilmeden bazı tarihler
# arasına konan virgül işaretinin boşluk ile
# değiştirilmesi gerekir.
haber = haber.replace(","," ")
parçalar = haber.split()

# Her parça için birinci if sayı sorgulaması
# yapar. İkinci if sayılardan 4 basamaklı
# olanları tespit eder.
for x in parçalar:
    if x.isnumeric():
        if len(x)==4:
            print(x)
```

7. MODÜLLER

Modüller daha sonra kullanılmak maksadıyla önceden yazılmış çeşitli öznitelikler (değişkenler, fonksiyonlar vd.) içerebilen "*.py*" uzantılı kod dosyalarıdır.

7.1 Modül Oluşturma

Bir Python kodu, "*.py*" dosya uzantısı ile bilgisayara kaydedilerek modül oluşturulabilir. Bu "*.py*" uzantılı dosyaya verilen isim aynı zamanda modül ismi olarak kullanılır. Örneğin aşağıda iki satırlık bir fonksiyon "*türkçe.py*" isim ve uzantısıyla bilgisayara kaydedildiğinde "**türkçe**" isimli bir modül oluşturulmuş olur.

```
1  # türkçe.py modülü
2  def yaz(metin):
3      print(metin)
4
5  def topla(a,b):
6      print(a+b)
```

7.2 Modül Kullanma

Modüllerin kullanılabilmesi için öncelikle çalışma alanına dahil edilmeleri gerekir. Bu işlem **import** komutu ile yapılmaktadır. Bu komutu bir boşluk ve modül ismi takip etmelidir. Modül içindeki bir fonksiyon kullanılmak istendiğinde sırasıyla önce modül adı yazılır (**türkçe**), sonra bir nokta konur (**türkçe.**) ve fonksiyonun adı (**türkçe.yaz()**) yazılır. Tüm bu işlemlerin hata alınmadan gerçekleşmesi için önceden yazdığınız modül ile şimdi yazmakta olduğunuz kod dosyasının aynı klasörde olmaları gerekmektedir.

```
1  import türkçe
2
3  türkçe.yaz("Merhaba")
4  türkçe.topla(3,2)
```

```
>>>

Merhaba
5
```

Python'da modül ve fonksiyon isimlerine kısa isim atanabilmektedir. Bunun yapılmasındaki amaç, her defasında modülün tam adını yazmanın güçlük oluşturmasıdır. Kısa isim ataması "**as**" kodu ile yapılmaktadır.

```
1 import türkçe as tr
2 tr.yaz("Merhaba")
```

```
>>>

Merhaba
```

Bir modüldeki öznitelikler (değişken veya fonksiyonlar) salt olarak çalışma ortamına dahil edilebilirler. Yani bir modülde örneğin sadece bir fonksiyona ihtiyaç duyuluyorsa tüm modülü ortama dahil etmek yerine sadece o fonksiyon dahil edilebilir. Değişkenler için de geçerli olan bu yöneteme, modüle erişim kısıtlaması getirme amacıyla başvurulabilmektedir. Çalışma ortamına tek bir özniteliği dahil etmek için **from** ve **import** komutları aşağıda gösterildiği gibi birlikte kullanılmaktadır. Ortama dahil edilen öznitelik doğrudan ismi ile kullanılır.

```
1 from türkçe import yaz
2
3 yaz("Merhaba")
```

```
>>>

Merhaba
```

7.3 Hazır Modüller

Modüller, programcılar tarafından kendi amaçlarını gerçekleştirmek için yazılırlar. Diğer taraftan Python'da kullanıma hazır birçok ön yüklü modül

de vardır.[6] Bu modülleri kullanmak için de aynı şekilde çalışma alanına dahil etmek gerekir. Burada **platform**, **random** ve **getpass** isimli üç hazır modülden bahsedeceğiz. Bunlardan başka, kitabın DOSYALAR adlı bölümünde **os** modülü ve MATEMATİK adlı bölümünde **math** modülü ayrıca ele alınmıştır.

platform

Python'da programın hangi işletim sisteminde çalıştığını gösteren **platform** modülü bulunmaktadır. Yazacağınız program farklı işletim sistemlerinde farklı tepkiler verebilir. Eğer programınız böyle bir program ise kullanıcının programınızı hangi işletim sisteminde çalıştırdığını kodunuzun başında sorgulamanız ve ona göre bir senaryo kodlamanız gerekir. İşte bu sorgulamayı **platform** modülü ile yapmak mümkündür.

```
1  import platform
2
3  sistem = platform.system()
4  print(sistem)
```

```
>>>

Windows
```

[6] Bu modüllerin listesi ve dokümantasyonu için https://docs.python.org/3/py-modindex.html adresine bakabilirsiniz. Ayrıca Python Shell'e **help('modules')** komutunu girerek modüllerin listesini alabilirsiniz.

random

Hazır modüllere bir başka örnek **random** modülüdür. Bu modül, bir listeden rastgele bir öğe seçmek veya belli bir aralıkta rastgele rakamlar üretmek için metotlar barındırır. Kura çekimleri için kullanılabilen bir modüldür.

```
import random

# Kullanımı: random.randint(start, end)
# ÖR:2 ve 9 aralığında 2 ve 9 dahil rastgele rakam
verir.
rastgele = random.randint(2, 9)
print(rastgele)

# Bir dizideki öğelerden rastgele seçim
seçenekler = ["a", "b", "c", "d", "e"]
seçim = random.choice(seçenekler)
# random.choice(seçenekler): seçenekler
# listesinden rastgele seçilen öğenin taşıdığı
# değeri verir.
print(seçim)
```

```
>>>

9
c
```

İçinde belirsizlik bulunduran bir duruma dair yapay test verisi üretmek için bu modül oldukça kullanışlıdır. Örneğin ortam sıcaklığına göre bir vantilatörü üç farklı hız kademesinde çalıştıracak bir kod yazıyorsunuz. Ortam sıcaklığınızın değişken olduğunu ve 27 °C ile 40 °C arasında

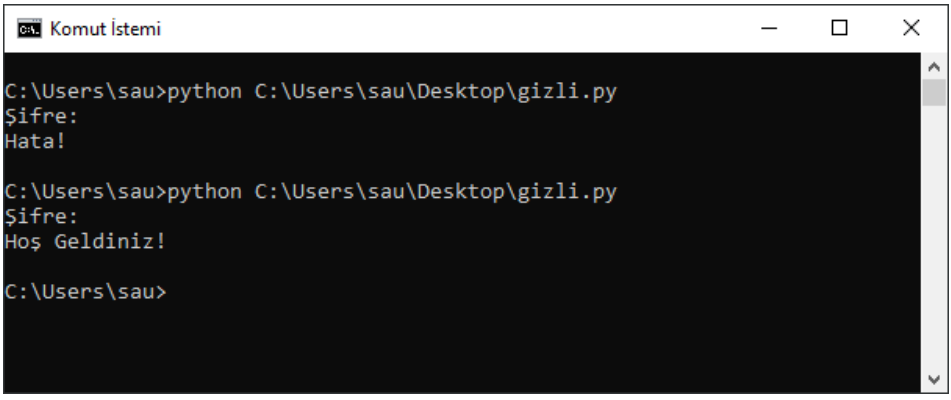
değiştiğini varsayalım. Ortam sıcaklığı 27-32 °C arasında ise vantilatörü 1.hız kademesinde, 33-37 °C arasında ise 2.hız kademesinde ve 38-40 °C arasında ise 3.hız kademesinde çalıştırmak istiyorsunuz. Buna dair bir kodlama yaptığınızda ve kodunuzu test etmek istediğinizde ortam sıcaklığı için 27 ile 40 arasında rastgele sayı verecek bir **`random`** kullanımı tanımlayabilirsiniz.

getpass

Daha önce kullanıcıdan girdi almak için **`input()`** fonksiyonunun kullanıldığından bahsetmiştik. Bu fonksiyon kullanıcının girdiği karakterleri ekranda aynen göstermektedir. Bu sebeple örneğin şifreler gibi gizli kalması gereken bilgilerin girişi için bu fonksiyon uygun değildir. Dışarıdan bakanların göremeyeceği şekilde bilgi girişi için Python'da **`getpass`** modülü bulunmaktadır. Bu modülün kullanımı aşağıdaki gibidir. Ancak **`getpass`** içeren bir kodlama, IDLE ile çalıştırıldığında uyarı vermekte ve işlevini yerine getirmemektedir. Dolayısı ile kodlamanın, Windows sisteminde "Komut İstemi" veya Mac sisteminde "Terminal" üzerinden çalıştırılması gerekir.[7]

[7] Komut İstemi veya Terminal'den bir Python belgesinin nasıl çalıştırıldığı, kitabın SENTAKS bölümünde verilmiştir.

```python
import getpass

gizliBilgi = getpass.getpass(prompt='Şifre: ')

if gizliBilgi == "1234":
    print("Hoş Geldiniz!")
else:
    print("Hata!")
```



Görüldüğü gibi şifre olarak girilen rakamlar ekrana yansımamaktadır.

7.4 Modül Değişkenleri

Modüller, değişkenleri de ihtiva edebilirler. Bunlar *list*, *dictionary*, *object* gibi türler olabilir. Aşağıda "listem.py" isimli modülün ihtiva ettiği *dictionary* türünde bir değişken verilmiştir.

```
müşteri1 = {
  "isim": "Murat",
  "yaş": 30,
  "il": "Ankara"
}
```

Değişkenlerin çalışma alanında kullanılması için yine öncelikle **import** komutu ile modül çağrılır. Fonksiyonlardaki gibi modül ismini (**listem**) bir nokta (**listem.**) ve değişken ismi (**listem.müşteri1**) takip eder. Daha sonra ulaşılmak istenen değerin anahtarı köşeli parantez içine (örnek **["yaş"]**) yazılır.

```
import listem

a = listem.müşteri1["yaş"]
print(a)
```

```
>>>

30
```

7.5 Özniteliklerin Listelenmesi

Çalışma alanına dahil edildikten sonra modüllerin sahip oldukları fonksiyonlar, **modül.fonksiyon()** formülü ile çağrılmakta idi. Ancak bunun için çağrılmak istenen fonksiyonun adının doğru yazılması gerekir. Bir modülün içerdiği fonksiyonların listesine erişmek bu anlamda kolaylık sağlar. İşte Python'da **dir()** fonksiyonu bu kolaylığı sağlamaktadır. Bu fonksiyon, bir Python nesnesinin (örneğin bir modülün) içerdiği öznitelikler (değişkenler, fonksiyonlar vd.) listesini

vermektedir. Python tarafından o nesneye otomatik tanımlanmış öznitelikler ve programcının tanımladığı öznitelikler alfabetik sıraya göre listede sıralanır. Aşağıda, “testModul.py” adında bir modül tanımlanmış ve sahip olduğu tüm özniteliklerin listesi istenmiştir.

```python
# testModul.py modülü

def yaz(metin):
    print(metin)

müşteri1 = {
  "isim": "Murat",
  "yaş": 30,
  "il": "Ankara"
}

a = 34
```

```python
import testModul

a = dir(testModul)
print(a)
```

```
>>>

['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__',
'a', 'müşteri1', 'yaz']
```

SORULAR

1. Bir modülü çalışma alanına dahil etmek için kullanılan kod nedir?

2. Çalışma alanına dahil edilecek modüle kısa isim ataması yapmak için kullanılan kod nedir?

3. Bir modülün sahip olduğu öznitelikleri listelemek için kullanılan kod nedir?

4. Bir modülün tamamını değil tek bir özniteliğini çalışma alanına dahil etmek için kullanılan kod nedir?

5. Bu soruda DÖNGÜLER bölümünde ikinci sürümünü yazdığınız oyunun üçüncü sürümünü yazmanız isteniyor. Oyun, bir sayı tahmin etme oyunudur. Programın kaynak kodunda tahmin edilecek sayı için bir değişken tanımlayın. Bu değişkene **`gizliSayı`** ismini verelim. Oyun çalıştırıldığında A şahsından bir sayı girmesi istenir. Bu sayı **`gizliSayı`** değişkenine atanır. Bu sayı, B şahsının tahmin edeceği gizli sayıdır. Bu sebeple girilen sayının ekranda görülmüyor olması gerekir. A şahsı giriş yaptıktan sonra oyun, B şahsının tahminini sorar. B şahsının ekrana yazacağı sayının görülüyor olması problem değildir. Program B şahsının girdiği sayıyı **`gizliSayı`** ile kıyaslar. B şahsı burada 3 farklı durum ile karşılaşabilir. Girdiği sayı, **`gizliSayı`**'ya eşit olabilir. Bu durumda ekranda "Tebrikler!" yazdığını görür. Girdiği sayı **`gizliSayı`**'dan küçük olabilir. Bu

durumda ekranda “Yukarı” yazdığını görür. Tersi durumda ise ekranda “Aşağı” yazdığını görür. B şahsı doğru tahmin yapana kadar program sonlanmaz. B şahsı yanlış cevap verdiğinde “Aşağı” ya da “Yukarı” çıktısı verilir ve tekrar yeni bir tahminde bulunması istenir. Her denemede ekrana kaçıncı denemeyi yaptığının bilgisi de yazdırılır. Örneğin “DENEME : 3” gibi. Yarışmacı doğru cevabı verdiğinde ekrana “Tebrikler!” ve “X denemede buldunuz” çıktısı yazdırılır. Doğru sonuca kaç denemede ulaştığı bir kağıda not edilir. Sonra diğer yarışmacılar oyunu oynar. Onların da kaç denemede doğru sonuca ulaştıkları not edilir. En az denemede **`gizliSayı`**’yı bulan yarışmacı oyunu kazanır.

CEVAP ANAHTARI

1. import

2. as

3. dir

4. from …… import ……

5.

```
# BU BİR SAYI TAHMİN ETME OYUNUDUR.
# Sürüm:3

import getpass

# Gizli sayı girişi istenir ve integer türüne
# çevrilir.
gizliSayı=getpass.getpass(prompt="Gizli Sayı: ")
gizliSayı = int(gizliSayı)

# Diğer gerekli değişkenler tanımlanır.
sonuç = False
deneme = 0

# Döngünün kaç çevrim olacağı bilinmediği için
# while döngüsü kullanılır.
while sonuç==False:
    deneme += 1
    print ("DENEME : " + str(deneme))
```

```
20
21      # Yarışmacının tahmini istenir.
22      tahmin = input("Bir tam sayı giriniz:")
23
24      # Alınan sayı integer türüne
25      # dönüştürülür.
26      tahmin = int(tahmin)
27
28      # Kıyaslama işlemi yapılır ve sonuçlar
29      # yazdırılır.
30      if tahmin > gizliSayı:
31          sonuç = False
32          print("Aşağı")
33      elif tahmin < gizliSayı:
34          sonuç = False
35          print("Yukarı")
36      else:
37          sonuç = True
38          print("Tebrikler!")
39          print(str(deneme)+ " denemede buldunuz")
```

8. DOSYALAR

Dosya işlemleri bilgisayar uygulamalarının önemli bir parçasıdır. Python, dosya oluşturma, dosya okuma, dosyaya yazma, dosyayı güncelleme ve dosya silme işlemleri için çeşitli araçlar sunar.

8.1 Dosya Açma

Dosyalar ile çalışmak için dosyaların öncelikle çalışma ortamına açılması gerekir. Bunun için Python'da **`open()`** fonksiyonu kullanılır. Bu fonksiyon, ilki dosya ismi ve ikincisi açma modu olmak üzere iki argüman alabilmektedir. Dosya açma modları şu şekildedir:

- "**`r`**" : Bu mod okuma modudur ve varsayılan moddur. Argüman girilmediği zaman dosya bu modda açılır. Açılmak istenen dosya mevcut değilse hata verir.
- "**`a`**" : Bu mod güncelleme modudur. Açılmak istenen dosya mevcut değilse, verilen isimde yeni bir dosya oluşturur.
- "**`w`**" : Bu mod yazma modudur. Açılmak istenen dosya mevcut değilse, verilen isimde yeni bir dosya oluşturur.
- "**`x`**" : Bu mod dosya oluşturma modudur. Oluşturulmak istenen dosya zaten var ise hata verir.

Bunlardan başka iki mod daha vardır. Bunlar:

- “**t**” : Açılacak dosyayı bir metin dosyası olarak açar. Bu mod varsayılan moddur. Bu grup argümanlarından biri girilmez ise dosya metin dosyası olarak açılır.
- “**b**” : Açılacak dosyayı bir binary dosyası (örneğin resim gibi) olarak açar.

Dosya açma modu için hiçbir argüman girilmez ise dosya, okuma modunda ve metin modunda açılır. Bunun argüman karşılığı “**rt**” dir. Görüldüğü gibi birinci grup mod karakterleri ile ikinci grup mod karakterleri yan yana yazılarak kullanılmaktadır. Argüman girişi yapmak için her zaman iki karakter girme zorunluluğu yoktur. İlk gruptan mod girişi yapıp ikinci gruptan mod girişi yapmamak mümkündür. Örneğin argüman olarak sadece “**w**” girişi yapmaya izin verilmiştir. Ancak bu durum ikinci grup için geçerli değildir. İkinci grup modları tek başlarına kullanılırsa hata alınır.

```
1 dosya1 = open("metin.txt")
2 dosya2 = open("metin.txt","rt")
```

Dosya açmak için kullanılan **open()** fonksiyonu bir dosya nesnesi döndürür. Yukarıdaki örnekte bu nesne, **dosya1** ve **dosya2** değişkenlerine atanmıştır. Dolayısı ile **read()** ve **write()** gibi dosya metotları bu değişkenler üzerinden örneğin **dosya1.read()** gibi çalıştırılır. Buna dair detayları ilgili başlıklara bırakıyoruz.

8.2 Dosya Okuma

Dosya okuma işlemi bir dosya nesnesi üzerinde **read()** metodunun çalıştırılmasıyla gerçekleştirilir. Örnek bir metin dosyası olmak üzere "metin.txt" dosyasının içeriği aşağıdaki gibi olsun.

```
Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Aşağıdaki kod, bu metin dosyasının nasıl okunduğunu göstermektedir.

```
dosya1 = open("metin.txt","r")
print(dosya1.read())
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Eğer aşağıdaki gibi Türkçe karakterlerin gösterilmesinde sorun yaşadıysanız **open()** fonksiyonuna bir argüman daha ilave etmeniz gerekir.

```
>>>

Bu bir Ã¶rnek metindir.
Python eÄŸitiminde kullanmak Ã¼zere yazÄ±lmÄ±ÅŸtÄ±r.
```

```
1 dosya1 = open("metin.txt","r",encoding="utf-8")
2 print(dosya1.read())
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Üçüncü argüman olan **encoding** argümanı, Türkçe karakter desteği sağlayan utf-8 karakter kodlama düzenine ayarlandığında karakter görüntüleme problemi ortadan kalkacaktır.[8]

Okunmak istenen dosya eğer Python kod dosyası ile aynı dosya konumunda ise (aynı klasörde ise) **open()** fonksiyonuna sadece metin dosyasının ismini yazmak yukarıdaki örneklerde görüldüğü gibi yeterlidir. Aynı konumda değillerse metin dosyasının dosya yolu ve ismi, aşağıda verilen örnekteki gibi fonksiyona girilmelidir. Bazı metin editörlerinden yapılan çalıştırmalarda, mutlaka tam dosya yolunun verilmesi gerekebilir. Aksi halde "dosya bulunamadı" hatası alınır.

```
1 dosya1 = open("C:\\Documents\metin.txt","r")
2 print(dosya1.read())
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

[8] Karakter kodlama düzenleri ile ilgili detaylı bilgi için https://tr.wikipedia.org/wiki/Karakter_kodlamas%C4%B1 adresine bakabilirsiniz.

Eğer metnin tamamı değil de bir kısmı okunmak isteniyorsa, bu durumda **`read()`** metoduna okunması istenen karakter adedinin argüman olarak girilmesi gerekir.

```
1 dosya1 = open("metin.txt","r")
2 print(dosya1.read(9))
```

```
>>>

Bu bir ör
```

Eğer metinden bir satır okunmak isteniyorsa, bu durumda **`readline()`** metodu kullanılır. Bu metot art arda çalıştırıldığında satırları sırasıyla okur.

```
1 dosya1 = open("metin.txt","r")
2 print(dosya1.readline())
```

```
>>>

Bu bir örnek metindir.
```

```
1 dosya1 = open("metin.txt","r")
2
3 # Birinci satırı okur
4 print(dosya1.readline())
5
6 # İkinci satırı okur
7 print(dosya1.readline())
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Eğer metnin tamamı satır satır okunmak isteniyorsa, bu durumda aşağıda gösterildiği gibi bir **for** döngüsü kullanılır.

```
1  dosya1 = open("metin.txt","r")
2
3  for satır in dosya1:
4      print(satır)
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Dosya ile ilgili işlemler bitirildiğinde **close()** metodu ile dosya kapatılır. Dosyada yapılan değişiklikler dosya kapatılıncaya kadar görülemeyebilir. Dolayısı ile üzerinde çalışılan dosya ile ilgili işlemler bittiğinde dosyanın kapatılması iyi bir kodlama pratiğidir.

```
1  dosya1 = open("metin.txt","r")
2
3  for satır in dosya1:
4      print(satır)
5
6  dosya1.close()
```

```
>>>

Bu bir örnek metindir.
Python eğitiminde kullanmak üzere yazılmıştır.
```

Metin içeren belgeler cümle ve paragraflardan oluşabildiği gibi tablo halinde verilerden de oluşabilmektedir. Buraya kadar cümle veya

paragraf halinde düzenlenmiş bir metin belgesinin okunmasını gördük. Şimdi tablo halinde düzenlenmiş metin belgelerinin nasıl okunacağına bakalım.

Tablo Okuma

Tablolar satır ve sütunlardan oluşan ve veri depolamada kullanılan araçlardır. Tablo içeren metin belgeleri ".txt" veya başka uzantılara sahip düz metin belgeleri olabilirler. Düz metin belgeleri metin editörleri (ör. Windows'ta Not Defteri uygulaması) ile okunabilen ve düzenlenebilen belgelerdir. Tablo yapısına sahip olduğu halde düz metin belgesi olmayan farklı uzantılara sahip belge türleri de vardır. Bunlar içerisinde en çok bilineni ".xlsx" uzantılı Microsoft Excel belgeleridir. Biz burada sadece tablo yapısına sahip düz metin belgelerinden nasıl veri okunabileceğini ele alacağız.

Tabloların satır ve sütunlardan oluştuğunu söylemiştik. Aşağıda tablo yapısında düz iki metin belgesi örneği verilmiştir. Bu tablolar 4 satır ve 3 sütundan oluşmaktadır. Verilerden anlaşıldığı üzere birinci sütun isim, ikinci sütun yaş ve üçüncü sütun ikamet yeri bilgisini barındırmaktadır.

tablo1.txt

```
1 Ahmet 18 İstanbul
2 Ali 21 Ankara
3 Meryem 23 İzmir
4 Zeynep 25 Adana
```

tablo2.txt

```
Ahmet;18;İstanbul
Ali;21;Ankara
Meryem;23;İzmir
Zeynep;25;Adana
```

Birinci örnekte sütunlar bir boşluk karakteri (*space*) ile birbirinden ayrılmıştır. İkinci örnekte sütunlar noktalı virgül (;) karakteri ile birbirinden ayrılmıştır. Sütunları birbirinden ayıran karakterin ne olacağına programcının kendisi karar verebilir. Burada dikkat edilmesi gereken husus, ayırıcı karakter olarak seçilen karakterin hiçbir şekilde metin içinde başka bir anlam ve amaçta kullanılmaması gerektiğidir. Şimdi bu metin belgesindeki verileri, bir liste değişkeni içerisine nasıl aktaracağımızı ele alalım.

Aşağıdaki örnekte sütunları boşluk ile ayrılmış tablo verisi (tablo1.txt), liste değişkenine aktarılmaktadır. Tablolar iki boyutlu veri yapıları olduğu için iki boyutlu liste içerisine alınabilirler. Yani bir iç içe liste kullanımı söz konusu olacaktır.

```
# Tablo verisi içeren metin belgesi açılır
belge = open("tablo1.txt", encoding="utf-8")

# Tek ve iki boyutlu olarak kullanacağımız
# boş listeler tanımlanır.
veri = []
veriListesi = []

# Belgenin satırları tek tek ele alınır.
# Her satır, boşluklardan bölünür.
# Böylece tekil veriler elde edilir.
# Bu veriler önce tek boyutlu listeye atanır.
# Sonra her çevrimde yenilenen bu liste iki
# boyutlu listeye ardarda eklenir.
for satır in belge:
    veri = satır.split()
    veriListesi.append(veri)

print(veriListesi)

print("----------")

# Büyük listeden bir yaş listesi elde etmek
# istersek önce bir boş liste tanımlarız.
yaşListesi = []

# Veri listesinin alt listeleri (x) tek tek ele
# alınır. Yaş bilgisi ikinci sırada olduğu için
# indeks olarak x[1] kullanılır.
for x in veriListesi:
    yaş = x[1]
    yaşListesi.append(yaş)

print(yaşListesi)
```

```
>>>

[['Ahmet', '18', 'İstanbul'], ['Ali', '21',
'Ankara'], ['Meryem', '23', 'İzmir'], ['Zeynep',
'25', 'Adana']]
----------
['18', '21', '23', '25']
```

Noktalı virgül ile ayrılmış sütunlar içeren “tablo2.txt” belgesi için ise aşağıdaki gibi bir kodlama yapılır.

```
# Tablo verisi içeren metin belgesi açılır
belge = open("tablo2.txt", encoding="utf-8")

veri = []
veriListesi = []

# Bu defa split fonksiyonuna ";" argümanı
# girilir
for satır in belge:
    # Satır sonlarında görünmez bir "\n"
    # karakteri olduğu için bu karakter strip
    # metodu ile temizlenir.
    satır = satır.strip()
    veri = satır.split(";")
    veriListesi.append(veri)

print(veriListesi)

print("----------")

yaşListesi = []

for x in veriListesi:
```

```
    yaş = x[1]
    yaşListesi.append(yaş)

print(yaşListesi)
```

```
>>>

[['Ahmet', '18', 'İstanbul'], ['Ali', '21',
'Ankara'], ['Meryem', '23', 'İzmir'], ['Zeynep',
'25', 'Adana']]
----------
['18', '21', '23', '25']
```

Görüldüğü üzere sütunlar hangi karakter ile ayrılmış ise o karakteri **`split`**[9] metodunda kullanıyoruz. Satır sonlarındaki görünmez yeni satır karakteri (**`\n`**) için de önlem olarak **`strip`**[10] metodunu kullanıyoruz. Tablodaki yaş verilerini örneğin bir liste içine aldıktan sonra kişilerin yaş ortalamasını hesaplayabilir veya yaşça en büyük kişiyi tespit edebiliriz. Bundan sonrası artık ne yapılmak istendiğine göre değişiklik gösterebilir.

8.3 Dosyaya Yazma

Yazma işlemi için **`open()`** fonksiyonuna "**`a`**" veya "**`w`**" argümanlarından birisi girilmek zorundadır. "**`a`**" argümanı güncelleme modu idi. Bu mod, önceki metnin korunmasını ve yeni metnin son satıra eklenmesini sağlar. Yani önceden dosyada mevcut olan yazılar silinmez ve son gönderilen metin, dosyanın sonuna eklenir. "**`w`**" argümanı yazma modu idi. Bu modda, önceki metin tümüyle silinir ve yeni gönderilen metin dosyaya

[9] Metodun kullanımı kitabın STRING bölümünde açıklanmıştır.
[10] Metodun kullanımı kitabın STRING bölümünde açıklanmıştır.

yazılır. Veri kaybı yaşamamak için bu iki argümandan hangisinin kullanılması gerektiğine dikkat edilmelidir.

Üzerine metin girişi yapılacak dosya, **`open("dosyaAdı","a")`** fonksiyon ve argümanlarıyla açılıp dosya nesnesi olarak bir değişkene (**`dosya1`**) atanınca artık **`write()`** metodu kullanılabilir duruma gelir. Eklenecek metin **`write()`** metoduna aşağıdaki gibi argüman olarak girilir.

```
1  # Dosyanın güncelleme modunda açılması ve
2  # değişkene atanması
3  dosya1 = open("metin.txt","a")
4
5  # Yeni metnin yazılması
6  dosya1.write("Bu, yeni bir metindir!")
7
8  # Dosyanın kapatılması
9  dosya1.close()
10
11 # Dosyanın tekrar okuma modunda açılması
12 dosya1 = open("metin.txt","r")
13
14 # Dosyanın okunarak konsola yazdırılması
15 print(dosya1.read())
16
17 #Dosyanın kapatılması
18 dosya1.close()
```

```
>>>

Bu bir örnek metindir.
Python eğitimde kullanmak üzere yazılmıştır.Bu, yeni
bir metindir!
```

Aynı yazma işlemi bu defa “**w**” modunda tekrarlanırsa önceki metnin silindiği görülecektir.

```
# Dosyanın yazma modunda açılması ve değişkene
# atanması
dosya1 = open("metin.txt","w")

# Yeni metnin yazılması
dosya1.write("Bu, yeni bir metindir!")

# Dosyanın kapatılması
dosya1.close()

# Dosyanın tekrar okuma modunda açılması
dosya1 = open("metin.txt","r")

# Dosyanın okunarak konsola yazdırılması
print(dosya1.read())

#Dosyanın kapatılması
dosya1.close()
```

```
>>>

Bu, yeni bir metindir!
```

Tabloya Yazma

Yazma ihtiyacı çoğu zaman verilerin yazılması olarak karşımıza çıkar. Verilerin ise bir düz yazı gibi değil, tablo halinde yazılması gereklilik arz eder. Tablo yapısı daha önce de söz ettiğimiz gibi satır ve sütunlar içerir. Satır oluşturmak için “\n” karakterinin kullanıldığından da söz etmiştik. Sütunları birbirinden ayırmak için ise nişan özelliği taşıyabilecek bir

karakter kullanılması gerekir. Bu karakterin tablo değerleri (tablo hücreleri) içinde başka bir amaçla kullanılmaması gerekir. Örneğin boşluk (*space*) karakterini sütun ayırıcı karakter olarak kullanmak istersek bunu sadece ve sadece sütun geçişlerinde (iki sütun arasında) kullandığımızdan emin olmak zorundayız.

Tablo formatında yazma yapılacağı zaman karar verilmesi gereken bir diğer konu yeni oluşturulacak belgenin hangi uzantıya sahip olacağı konusudur. İstediğiniz uzantıda bir belge oluşturabileceğinizi burada ifade edebiliriz. Yaygın olarak kullanılan düz metin belgesi uzantısı ".txt" uzantısını da kullanabilirsiniz. Eğer bu belgeyi sadece veri depolama amaçlı oluşturmuyorsanız kullanmanız gereken uzantı önem kazanır. Örneğin oluşturduğunuz tabloyu Hesap Tablosu programlarında (Microsoft Excel gibi) açabilmek ve üzerinde çalışabilmek isterseniz ".csv" uzantısını kullanmanız önerilir.[11] Bu uzantının avantajı hem düz metin editörlerinde açılabiliyor olması hem de Microsoft Excel programında tablo halinde açılabiliyor olmasıdır. ".csv" formatı düz metin formatı olduğu için Python'nun yerleşik dosya okuma ve yazma metotları ile okunabilmekte ve düzenlenebilmektedir. Harici bir modüle ihtiyaç kalmamaktadır.

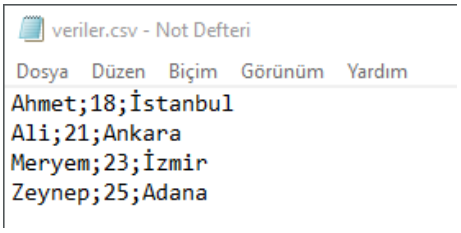
Tablo olarak düzenlenmiş düz bir metin belgesinin Microsoft Excel ile açılabilmesi için sadece uzantısını ".csv" olarak kaydetmek yeterli değildir. Sütun ayırıcı olarak kullanılacak karakterin seçimi de burada önemlidir. ".csv" uzantılı olarak düzenleyeceğimiz metin belgesinde sütun ayırıcı karakter olarak ya virgül (,) ya da noktalı virgül (;) kullanmamız gerekir. Dili İngilizce olan sistemlerde genellikle virgül (,)

[11] Csv uzantısının adı, "Comma-separated Values" ifadesinin baş harflerinden gelmektedir.

kullanılırken, dili Türkçe olan sistemlerde genellikle noktalı virgül (;) kullanılmaktadır. Çünkü Türkçe sistemlerde sayıların ondalık kısmı virgül ile ayrıldığı için bu karakter sütun ayırıcı olarak kullanılamıyor. Bu açıklamalardan sonra tablo formatına sahip, sütunları noktalı virgülle ayrılmış, ".csv" uzantılı, düz bir metin belgesi oluşturmayı görelim.

```
isim = ['Ahmet','Ali','Meryem','Zeynep']
yaş = ['18','21','23','25']
il = ['İstanbul','Ankara','İzmir','Adana']

belge = open("veriler.csv","w")
for x in range(4):
  belge.write(isim[x]+";"+yaş[x]+";"+il[x]+"\n")
belge.close()
```

Listeler halindeki üç sütunluk (isim, yaş ve il) ve dört satırlık veriler yukarıda gösterildiği gibi "veriler.csv" isimli metin belgesine yazdırılmıştır. Bu belgenin metin editöründe ve Microsoft Excel'de nasıl gözüktüğü aşağıda yan yana verilmiştir.

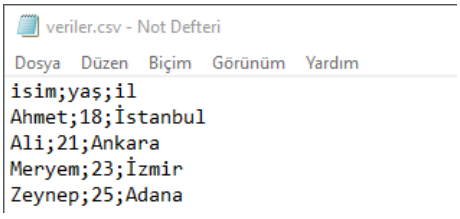


	A	B	C	D
1	Ahmet	18	İstanbul	
2	Ali	21	Ankara	
3	Meryem	23	İzmir	
4	Zeynep	25	Adana	
5				
6				

Sütunlara isim verilmek istenirse o taktirde yukarıdaki kodlamaya aşağıdaki gibi bir ilave yapılabilir.

```
isim = ['Ahmet','Ali','Meryem','Zeynep']
yaş = ['18','21','23','25']
il = ['İstanbul','Ankara','İzmir','Adana']

belge = open("veriler.csv","w")

# Sütunlara isim verme
belge.write("isim;yaş;il\n")

for x in range(4):
  belge.write(isim[x]+";"+yaş[x]+";"+il[x]+"\n")

belge.close()
```

Bu durumda belgenin metin editöründeki ve Microsoft Excel'deki görünümü aşağıdaki gibi olur.



	A	B	C	D
1	isim	yaş	il	
2	Ahmet	18	İstanbul	
3	Ali	21	Ankara	
4	Meryem	23	İzmir	
5	Zeynep	25	Adana	
6				

8.4 Dosya Oluşturma

Python dosya işlemlerinde **`open()`** fonksiyonu, "**`a`**", "**`w`**" veya "**`x`**" modlarından biriyle kullanılırsa yeni bir dosya oluşturulma ihtimali oluşur. "**`a`**" modu, eğer verilen dosya isminde bir dosya yoksa o isimde yeni bir dosya oluşturur. "**`w`**" modu da dosya oluşturma konusunda aynı "**`a`**" modu gibi davranır. "**`x`**" modu ise dosya oluşturmaya özgü bir moddur. Verilen dosya yolunda ve verilen isimde bir dosya mevcut değilse o dosya

yolunda ve o isimde bir dosya oluşturur. Eğer o dosya yolunda, o isimde bir dosya varsa hata verir.

```
1  # Fonksiyonun dosya oluşum modunda çağrılması
2  dosya1 = open("metin.txt","x")
```

```
>>>

FileExistsError: [Errno 17] File exists: 'metin.txt'
```

8.5 Dosya Silme

Dosya silme işlemleri **os** modülünde bulunan **remove()** fonksiyonu ile gerçekleştirilebilir. Dolayısıyla **import** kodu ile os modülünün ortama dahil edilmesi gereklidir. Silinecek dosya bulunamaz ise **FileNotFoundError** hatası alınır.

```
1  import os
2  os.remove("metin.txt")
```

8.6 Dosya Sorgulama

Bir dosya işlemi yapılacağı zaman dosyanın ilgili klasörde olup olmadığının sorgulanması iyi bir kodlama pratiğidir. Çünkü bu sayede programın hata vermesinin önüne geçilmiş olur. Bir dosyanın sorgulanması **os** modülüne ait fonksiyonla yapılmaktadır. Aşağıda silme işleminden önce bir sorgulama gerçekleştirilmektedir.

```
import os
if os.path.exists("metin.txt"):
  os.remove("metin.txt")
else:
  print("Dosya mevcut değil!")
```

8.7 Klasör Silme

Bir klasör silinmek isteniyorsa bu durumda **os** modülünün **rmdir()** fonksiyonu kullanılır. Bu işlem, sadece içi boş bir klasörü silmektedir.

```
import os
os.rmdir("dosyam")
```

SORULAR

1. Verilen Python kodunda çalışma ortamına bir dosya açılmak istenmektedir. Bu dosya, yazma ve metin modunda açılmak istendiğine göre ? işareti olan yere hangi kodlar yazılmalıdır?

```
dosya = open("metin.txt","?")
```

2. Verilen Python kodunda bir metin belgesi okunmak ve tüm içerik ekrana yazdırılmak istenmektedir. Bunu sağlayacak kodu ilave ederek işlemi tamamlayınız.

```
1 dosya = open("metin.txt")
```

3. Verilen Python kodunda bir metin belgesi okunmak ve içeriğin sadece ilk satırı ekrana yazdırılmak istenmektedir. Bunu sağlayacak kodu ilave ederek işlemi tamamlayınız.

```
1 dosya = open("metin.txt")
```

4. Verilen Python kodundaki metin belgesi güncelleme modunda açılmak ve metnin sonuna "bu yeni bir metin girişidir." ifadesi eklenmek istenmektedir. Bunu sağlayacak kodu ilave ederek işlemi tamamlayınız.

```
1 dosya = open("metin.txt","?")
2 ?
```

5. Bir metin belgesini çalışma ortamına açmak için kullanılan **`open()`** fonksiyonu hangi argümanlarla açılırsa ilgili dosya eğer yerinde yoksa sıfırdan oluşturulur?

6. Verilen Python kodunda bir metin belgesi silinmek istenmektedir. Bunu sağlamak için ? işaretli yerlere gelmesi gereken kodlar nelerdir?

```
? os
os.?("metin.txt")
```

7. Bir işletme, sattığı ürünlerin fiyatlarını bir dosyaya kaydetmek ve barkod sorgulaması ile fiyat bilgisine ulaşmak istemektedir. Dolayısı ile ihtiyaç duyulan program hem yeni kayıt yapmaya hem de barkod sorgulamaya imkân tanımalıdır. Program açıldığında kullanıcıya üç seçenek sunulmalı ve bir seçim yapması istenmelidir. Bu seçenekler "Yeni Kayıt" ve "Barkod Sorgulama" ve "Çıkış" olmalıdır. Örneğin "Yeni Kayıt için (1), Barkod Sorgulama için (2) ve Çıkış için (3) rakamını giriniz" gibi bir yönlendirme yapılabilir. Yeni kayıt seçildiğinde kullanıcıdan önce bir şifre girmesini isteyiniz. Şifre doğrulaması yapıldıktan sonra "Yeni barkodu giriniz" uyarısı ile barkod numarası girmesini isteyiniz. Daha sonra "Ürün fiyatını giriniz" uyarısı ile fiyat bilgisini girmesini isteyiniz. Daha sonra bu bilgileri bir dosyaya kaydediniz. Her yeni kayıt aynı dosyaya yapılmalı ve eski bilgiler silinmemelidir. Barkod sorgulama seçildiğinde ise kullanıcıdan şifre sormaksızın bir barkod numarası girmesini isteyiniz. Girilen barkod numarasının dosyada olup olmadığını kontrol ediniz. Dosyada ise o barkoda ait fiyat bilgisini ekrana yazdırınız. Dosyada öyle bir numara yok ise "Barkod bulunamadı!" bilgisini veriniz. Her durumda yapılmak istenen işlem bittiğinde programı en başa yani seçim ekranına döndürünüz. Çıkış seçeneği seçildiğinde ise program döngüsünü sonlandırınız.

CEVAP ANAHTARI

1.

```
dosya = open("metin.txt","wt")
```

2.

```
dosya = open("metin.txt")
print(dosya.read())
dosya1.close()
```

3.

```
dosya = open("metin.txt")
print(dosya.readline())
dosya.close()
```

4.

```
dosya = open("metin.txt","a")
dosya.write("bu yeni bir metin girişidir.")
```

5. a, w, x

6.

```
import os
os.remove("metin.txt")
```

7.

```
# BARKOD-FİYAT KAYDETME VE SORGULAMA PROGRAMI
import getpass

# Programdan çıkış için kullanılacak boolean
devam = True

# Ana menüye dönüş için while döngüsü
while devam:
    # Seçeneklerin kullanıcıya sunumu ve giriş
    # istemi
    seçenek = input("""
Yeni Kayıt\t\t(1)
Barkod Sorgulama\t(2)
Çıkış\t\t\t(3)

Bir seçim yapınız :""")
    # Seçenek denetimi
    # Yeni kayıt şeçeneği
    if seçenek == "1":
        # Şifre istemi
        şifre =getpass.getpass(prompt='Şifre: ')
        # Şifre doğrulaması
        if şifre == "1234":
            # Excel ile açılabilmesi için csv
            # uzantısı kullanılır.
            dosya1 = open("defter.csv","a")
            barkod = input("Barkod :")
            fiyat = input("Fiyat :")
            # Türkçe sistemlerde csv noktalı
```

```
30              # virgül ile çalışır.
31              dosya1.write(barkod+";"+fiyat+"\n")
32              dosya1.close()
33          else:
34              print("Hatalı şifre!")
35
36      # Barkod sorgulama seçeneği
37      elif seçenek == "2":
38          # Dosya bulunamadı hatasına karşı önlem.
39          try:
40              dosya2 = open("defter.csv","r")
41              sözlük = {}
42              liste = []
43              # Dosyadaki veriler satır satır önce
44              # listeye sonra listeden de sözlüğe
45              # alınır. Sözlük öğeleri için
46              # liste[0] anahtar olurken
47              # liste[1] de değer olur.
48              for satır in dosya2:
49                  liste = satır.split(";")
50                  sözlük[liste[0]] = liste[1]
51
52              x = input("Aranacak Barkod: ")
53              if x in sözlük:
54                  print(sözlük[x])
55              else:
56                  print("Aranan barkod
    bulunamadı!")
57
58              dosya2.close()
59          except:
60              print("Kayıt yapılmamış")
61      # Çıkış seçeneği
62      else:
63          devam = False
```

9. MATEMATİK

Python'da bazı matematiksel fonksiyonlar yerleşik olarak bulunur. Bunlar doğrudan kod içerisinde kullanılabilen fonksiyonlardır. Bunlardan başka, matematiksel fonksiyonları içeren **`math`** isimli bir hazır modül vardır. Modül içindeki fonksiyonların kullanılması için öncelikle çalışma ortamına **`import`** kodu ile dahil edilmeleri gerekir.

9.1 Yerleşik Fonksiyonlar

Yerleşik fonksiyonlardan bazıları aşağıda listelenmiş ve kullanımları örneklenmiştir.

- **`min(a,b,c,...)`** : En küçük değeri döndürür.
- **`max(a,b,c,...)`** : En büyük değeri döndürür.
- **`abs(a)`** : Sayının mutlak değerini döndürür.
- **`pow(a,b)`** : a üzeri b'nin değerini döndürür.

```
1  f1 = min(5,-23,34)
2  f2 = max(5,-23,34)
3  f3 = abs(-4.78)
4  f4 = pow(2,3)
5
6  print(f1)
7  print(f2)
8  print(f3)
9  print(f4)
```

```
>>>

-23
34
4.78
8
```

9.2 Math Modülü

Math modülü, Pythonda hazır olarak bulunan bir modüldür. Kullanılması için **`import`** komutu ile çalışma ortamına çağrılması gerekir. Bu yapıldıktan sonra içerdiği fonksiyonlar, **`math.fonksiyonAdı()`** şeklinde kullanılabilir. Modül ayrıca bazı matematiksel sabitleri de içermektedir. Bunların kullanımı ise **`math.sabitAdı`** şeklindedir. Bu fonksiyon ve sabitlerden bazıları aşağıda listelenmiştir.

9.2.1 Sabitler

- **`math.pi`** : pi sayısını verir.
- **`math.tau`** : tau sayısını verir.
- **`math.inf`** : pozitif sonsuz için kullanılır.
- **`math.e`** : e sayısı (Euler sayısı)

9.2.2 Fonksiyonlar

Math modülünde bulunan bazı önemli fonksiyonlar tabloda verilmiştir.

Fonksiyon	İşlevi
`math.pow(a,b)`	a üzeri b'nin değerini döndürür.
`math.sqrt(a)`	sayının karekökünü verir.
`math.factorial(a)`	sayının faktöriyel değerinin verir.
`math.remainder(a,b)`	bir bölme işleminde kalan miktarı verir.
`math.fsum(a)`	indeksli değişkendeki sayıların toplamını verir.
`math.prod(a)`	indeksli değişkendeki sayıların çarpımını verir.
`math.exp(a)`	e üzeri a'nın değerini verir.
`math.log(a)`	sayının doğal logaritmasını verir.
`math.log10(a)`	sayının 10 tabanında logaritmasını verir.
`math.ceil(a)`	sayıyı yukarı yuvarlar.
`math.floor(a)`	sayıyı aşağı yuvarlar.
`math.gcd(a,b)`	iki sayının en büyük ortak bölenini verir.

math.radians(a)	derece cinsinden açıyı radyan cinsine dönüştürür.
math.sin(a)	radyan açının sinüs değerinin verir.
math.sinh(a)	radyan açının hiperbolik sinüs değerinin verir.
math.cos(a)	radyan açının kosinüs değerinin verir.
math.cosh(a)	radyan açının hiperbolik kosinüs değerinin verir.
math.tan(a)	radyan açının tanjant değerinin verir.
math.tanh(a)	radyan açının hiperbolik tanjant değerinin verir.
math.comb(a,b)	a'nın b'li kombinasyonlarının sayısı.
math.perm(a,b)	a'nın b'li permütasyonlarının sayısı.

KAYNAKLAR

1. Computer program. https://www.britannica.com/technology/computer-program, 2021.

2. Hello World. https://www.codecademy.com/learn/learn-python-3/modules/learn-python3-hello-world/cheatsheet, 2021.

3. How to Think Like a Computer Scientist. http://openbookproject.net/thinkcs/python/english3e/index.html, 2021.

4. Karakter kodlaması. https://tr.wikipedia.org/wiki/Karakter_kodlamas%C4%B1, 2021.

5. Python 3.9.2 documentation. https://docs.Python.org/3/, 2021.

6. Python Tutorial. https://www.w3schools.com/Python/, 2021.

7. Wentworth, P., Elkner, J., Downey A. and Meyers, B. Programming with Python. https://swcarpentry.github.io/python-novice-inflammation/, 2021.