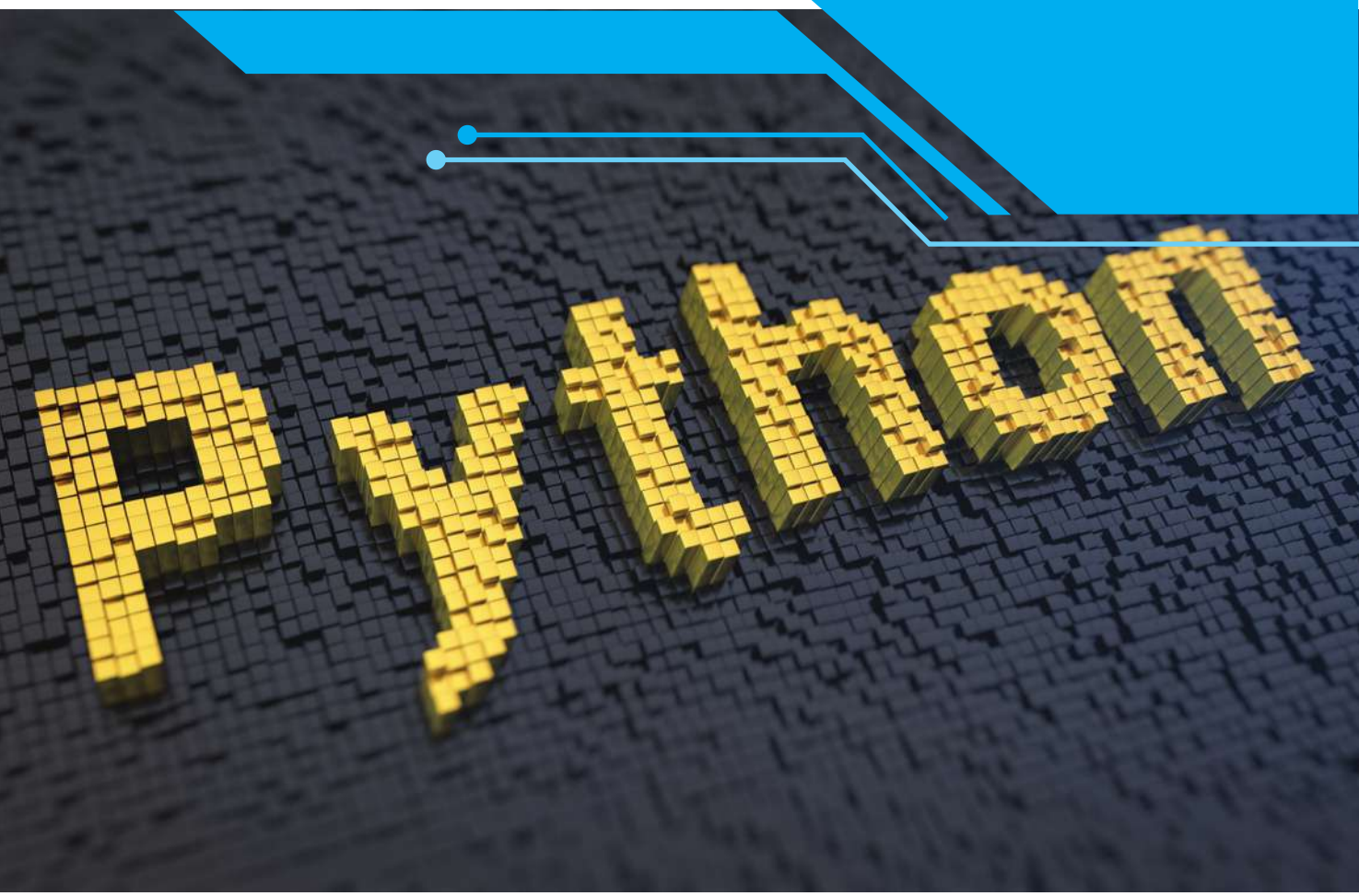
1. PYTHON İLE PROGRAMLAMANIN TEMELLERİ



Bu bölümde;

- ✓ Yazılım geliştirme süreci konusunda bilgi sahibi olacak,
- ✓ Yazılım geliştirme sürecinde gerekli olan araçları tanıyacak,
- ✓ Python dilinde program geliştirme ortamlarını inceleyebileceksiniz.

1.1. Yazılım Geliştirme Süreci

Bir bilgisayar programı, bilgisayar sistemindeki elektrik sinyallerinin akışını yöneten bir dizi yönergedir. Bu sinyaller, bilgisayarın hafızasını etkileyerek ekran, klavye, fare ve hatta diğer bilgisayarlar ile etkileşim sağlar. Bu şekilde insanlar da karmaşık problemleri çözmek, oyun oynamak gibi değişik işlemleri gerçekleştirebilir. Bir program hesap makinesi rolü üstlenirken bir diğeri makineyi satranç tahtasına dönüştürebilir. Buradaki iki örnek;

- Alt düzeyde, elektrik sinyalleri somut biçimde bilgisayarın mevcut durumunu değiştirirken,
- Üst düzeyde, soyut bir biçimde kullanıcıların eğlenme ya da işleri gereği farklı işlemleri gerçekleştirmelerini sağlar.

Günümüzde, kullanıcı arayüzüne aktarılan üst düzey işlemler ile alt düzey işlemleri kontrol edebilmek, son derece kolaylaşmıştır ve bu nedenle pek çok programcı bu dilleri kullanarak daha soyut bir biçimde kod yazmaktadır. Programlama seçenekleri programlamayı sürpriz bir biçimde basit hâle getirmiştir. Programlama kavramları, temelinde mantıksal ve matematiksel olarak sınıflandırılabilir. Bilgisayar programları bir bilgisayar kullanmadan da yazılabilir. Programcılar, bir programın çalışabilirliğini ve doğruluğunu, gerçek yaşamda yeri olmayan soyut semboller kullanarak tartışabilirler. Bu yaklaşımlar çok değerli olmakla beraber, programcılar çoğu zaman makinelerden uzak kalamazlar. Yazılımlar, gerçek bilgisayar sistemlerinde kullanılmak üzere yazılır. Yazılım mühendisleri belirli sistemler üzerinde çalışacak şekilde programları üretirler. Bu sistemler, donanım alt yapıları ve kullandıkları işletim sistemi ile tanımlanır. Yazılım geliştirenler; derleyici (compiler), hata ayıklayıcı (debugger) ve yanaylaç (profiler) gibi somut araçları kullanırlar.

Yazılım geliştirme süreci şu şekilde işler;

- Programcı programlama dili kullanarak kodları oluşturur,

- Yazılan kod bütünü, hata ayıklayıcı (debugger) kullanılarak hatalara karşı denetlenir,

- Hataları giderilmiş kodlar, derleyici (compiler) kullanılarak bilgisayarın yorumlayabileceği elektriksel sinyallere dönüştürülür.

Bu süreç sonunda bilgisayar, elektriksel sinyalleri yorumlayarak komutların gereğini yapar. Ayrıca yazılım geliştirme sürecinde yanaylaç (profiler) kullanımı, yazılımcının, programın daha hızlı çalışmasını sağlayacak şekilde kodları revize etmesine olanak sağlar.

1.1.1 Yazılım

Yazılım, bilgisayarın donanımını anlamlı hale getiren, bilgisayarları kullanıcıların amaçları doğrultusunda kullanmasını sağlayan kod, komut ve programlardır.

Bilgisayar yazılımlarına örnek olarak bilgisayar programları verilebilir. Bir program, bir yazılım parçası olarak nitelendirilebilir. Yazılımlar daha soyut, programlar ise biraz daha somuttur. Yazılımlar CD, DVD, sabit disk, taşınabilir bellek gibi farklı araçlar üzerinde saklanabilir. Yazılımların kullanabilmesi için bilgisayarın hafızasında kayıtlı olması gerekir. Bilgisayar programları hafızaya belirtilen araçlardan yüklenir. Bilgisayarın sabit diskine yüklenmiş olan program elektromanyetik bir örüntü oluşturur. Bu elektronik sembollerden oluşan örüntünün, program çalışmadan önce hafızaya transfer edilmesi gerekir. Program bir CD ya da İnternet üzerinden yüklenebilir. Bu elektronik semboller ikilik sayı sistemini kullanan, sıfır ve bir değerlerinden oluşan bir dizidir.

Örnek: Bir ikilik program şu şekilde görünür:

10001011011000010001000001001110

İşlemci olarak adlandırılan bilgisayar donanımı, bu sinyalleri çözümleyerek yapılmak istenen işlemi (grafik ara birimine veri göndererek ekranın bir bölümünün mavi görünmesini sağlamak gibi) gerçekleştirir.

1.1.2. Yazılım Geliştirme Ortamları

Yazılımlar, ikilik dizileri daha anlaşılabilir kılan kelime ve sembolleri kullanır. Böylece bilgisayarların dilini öğrenmek ve karmaşık problemleri çözen programlar yazmak kolaylaşır. Bu amaçla, komutları üst düzey yazılımlardan alt düzey makine diline çevirebilen araçlar da kullanılabilir. Python gibi üst düzey diller, programcıların İngilizce konuşma diline çok yakın bir şekilde program kodlarını yazabilmelerine olanak sağlar. Geçtiğimiz yaklaşık 60 yıllık süreçten günümüze kadar FORTRAN, COBOL, Lisp, Haskell, C, Perl, C++, Java ve C# gibi pek çok üst düzey dil geliştirilmiştir. Bu tür programlama dilleri ile uygulama geliştiren programcılar, donanım ya da makine dili gibi konulardaki detaylar ile ilgilenmeden çok etkili yazılımlar geliştirebilir. Böyle bir dönüştürme aracının, kendi dilimizi anlayıp işleme dökmesini bekleyebiliriz ancak günlük konuşma dilleri programlama dillerine göre son derece karmaşık olduğundan bu işlem olası değildir. Derleyici olarak kullandığımız, bir programlama dilini diğerine çeviren programlar yaklaşık 60 yıldır hayatımızdadır ancak konuşma dilinin işlenerek programa dönüştürülmesi hâlâ yapay zekâ araştırma konuları arasındadır. Günlük konuşma dilini belirli standartlar çerçevesinde anlaşılır kılmak, bugünkü yazılımların kapasitesinin çok üstünde bir beklentidir. Programlama dilleri oldukça basit bir yapı ve kesin kurallar ışığında bilgisayar tarafından çözülebilecek problemler için çözüm üretmektedir.

Geleneksel olarak yeni bir dilde yazılan ilk program "Merhaba, Dünya!" adı verilen programdır. Python'da aşağıdaki şekilde yazılmaktadır:

Python dili ile yazılmış ilk örneğe bakalım:

```
print "Merhaba, Dünya!"
```

Bu print komutunun bir örneği olup ekranda Merhaba, Dünya! yazar. Tırnak işaretleri programda bir değerin başlangıcını ve sonucunu gösterir ve ekranda gözükmez.

Python dili ile yazılmış aşağıdaki örneğe bakalım:

```
Toplam = 0
DersSaati = 3
Hafta = 14
Toplam = DersSaati * Hafta
```

Bu satırlar bir Python programındaki bazı satırlar olabilir. Bu satırlar bazı hesaplama işlemleri (= ve *) ile benzerlik göstermektedir. DersSaati, hafta ve toplam olarak ifade edilen kelimeler, değişken olarak adlandırılmaktadır. Bu değişkenler verileri bilgisayarın hafızasında korumak için kullanılır. Bu satırlar Python dili ile yazıldığından herhangi bir makine dili tarafından anlaşılır değildir. Kullanıcı programı çalıştırdığında, yorumlayıcı programlar, Python kodunu makine koduna çevirir. Üst düzey program kodu kaynak kod (source code) olarak adlandırılır. Bu koda karşılık gelen makine diline ise hedef kod (target code) adı verilir. Yorumlayıcı, kaynak kodu hedef koda dönüştürür. Üst düzey programların güzelliği, kodlamanın donanımdan bağımsız olarak yapılabilmesidir. Üstünde çalışılan platform ne olursa olsun, Python yorumlayıcısı kurulu ise tüm programlar tüm platformlarda çalıştırılabilir. Programcıların yazılım geliştirme sürecini destekleyen pek çok araç vardır. Bunlardan bazıları aşağıda listelenmiştir.

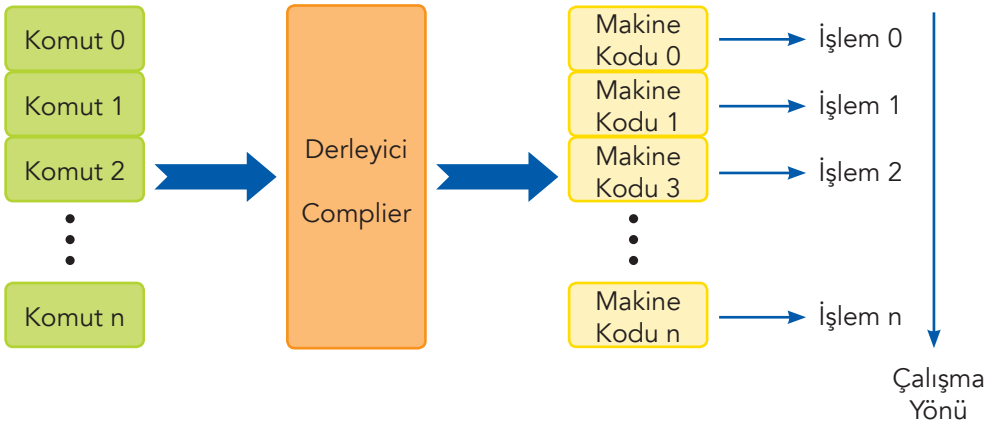
1.1.3. Editörler

Bir editör, programcının kaynak kodu yazmasını ve dosyaya kaydetmesini sağlar. Çoğu editör, renklendirme desteği sunarak dilin özelliklerini ortaya çıkarır ve programcının üretkenliğinin artmasını destekler. Dili oluşturan parçaların kurallara uygun bir şekilde düzenlenmesi söz dizimi (syntax) olarak ifade edilir. Geliştirme araçlarının yazılanları tam olarak doğru anlaması için, kullanılan kelime ve sembollerin kurallara uygun biçimde dizilmesi önemlidir. Yalnızca doğru biçimde ifade edilen programlar

makine koduna dönüştürülmek üzere kabul edilir. Bu nedenle bazı editörler yazım yanlışları konusunda renkleri ya da farklı vurgulamaları kullanarak yazım hataları konusunda programcıyı uyarır.

1.1.4. Derleyiciler

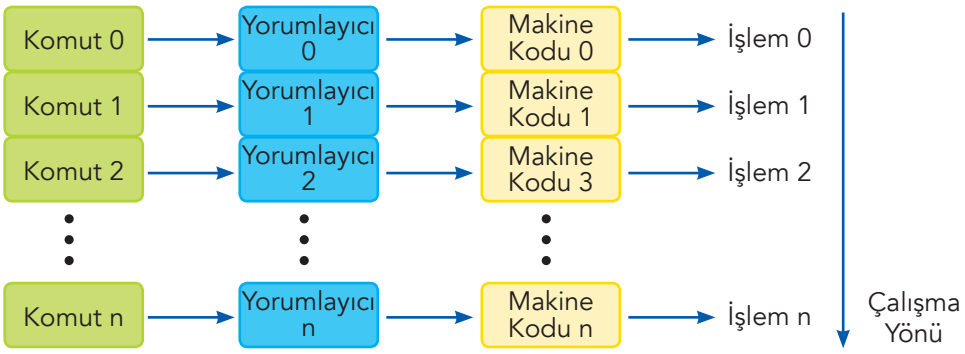
Derleyiciler, kaynak kodları hedef koda dönüştürür. Hedef kod, belirli bir platform ya da gömülü bir araç için makine dili olabilir (Şekil 2.1). Hedef kod diğer bir üst düzey kaynak dil de olabilir. Derleyiciler, kaynak kod içeriğini dönüştürerek hedef kod içeren bir dosya oluşturur. Derlenerek oluşturulan popüler dillere örnek olarak C, C++, Java ve C# verilebilir.



Şekil 2.1: Derleyici yapısının grafiksel gösterimi

1.1.5. Yorumlayıcılar

Yorumlayıcılar da derleyiciler gibi üst düzey kaynak kodu hedef koda (genellikle makine kodu) çevirir ancak derleyicilerden farklı çalışır (Şekil 2.2). Derleyiciler herhangi farklı bir dönüşüm gerekmeden defalarca çalışılabilir bir program kodu üretirken yorumlayıcılar kullanıcı kaynak kodu her çalıştırdığında satır satır makine diline çevirir. Derlenmiş bir program, değişiklik yapılmadığı sürece tekrar derlenmez ancak yorumlayıcı ile çalışan program için yorumlama işlemi değişiklik yapılmamış olsa bile tekrarlanmalıdır. Bu nedenle yorumlanan diller daha çok senaryo dili (scripting language) olarak ifade edilir. Yorumlayıcı, programın kaynak kodu olan senaryoyu okur. Genellikle derlenen programlar yorumlanan programlara göre daha hızlı çalışır çünkü derleme işlemi yalnızca bir kez yapılır. Diğer yandan yorumlanan programlar, herhangi bir platformda tekrar derlenmelerine gerek kalmadan uygun bir yorumlayıcı ile hemen çalıştırılabilir. Python, yorumlanan bir dil olmakla birlikte, bunun derleyicileri de vardır. Popüler senaryo dillerine örnek olarak Python, Ruby, Perl ve web ortamı için Javascript verilebilir.



Şekil 2.2: Yorumlayıcı yapısının grafiksel gösterimi

1.1.6. Hata Ayıklayıcılar

Hata ayıklayıcılar, programcının bir programdaki olası hataları bulmasına ve düzeltmesine olanak sağlayarak programın doğru çalışması için yardımcı olur. Hata ayıklayıcı programlar ile programın hangi satırlarında hata olduğu belirlenir. Programcı, değişkenlerin değerlerine bakarak neyin yanlış gittiğini anlayabilir.

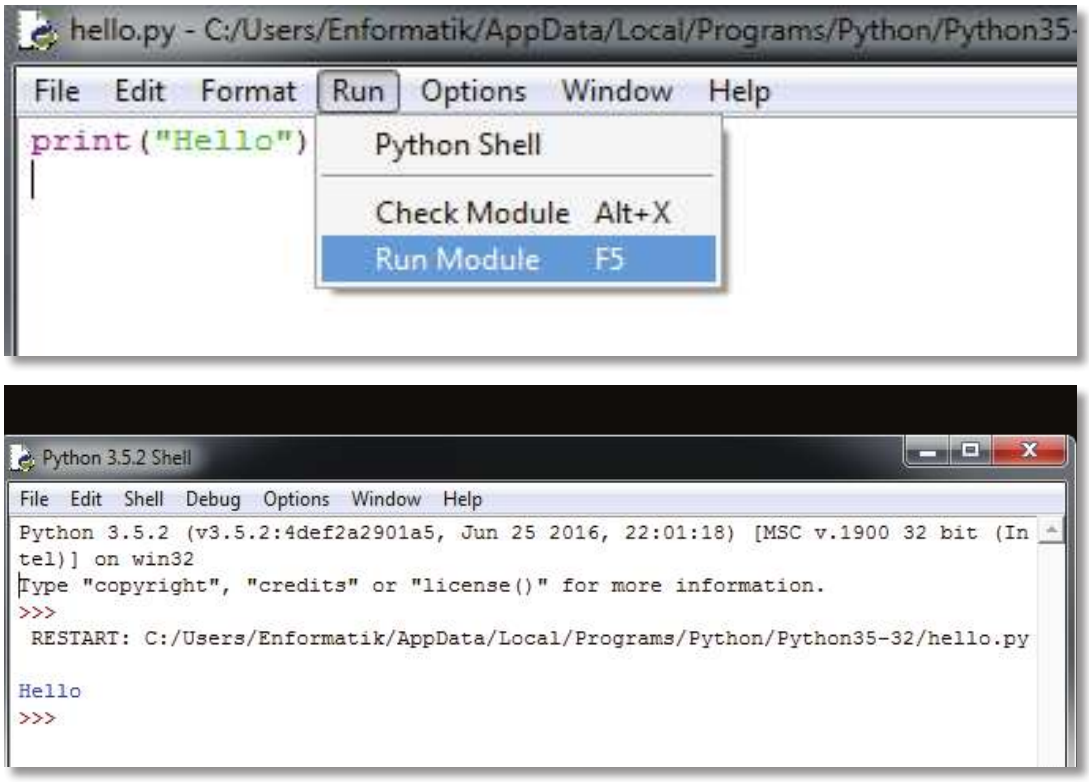
1.1.7. Yanaylaçlar

Yanaylaçlar, bir programın çalışmasına ilişkin istatistiki veri toplar. Böylece programcılar, genel olarak performansını arttırmaya yönelik önlemler alabilir ve programın belirli bölümlerini yeniden yapılandırabilir. Yanaylaç, program her çalıştırıldığında program parçalarının kaç kere çalıştırıldığını ve bu işlemin ne kadar sürdüğünü ortaya çıkarır. Bu işlem, programın gerçekten tüm parçalarının kullanılıp kullanılmadığını belirlemek için de kullanılabilir. Buna kaplam (coverage) denilir. Genel olarak programın belirlenen parçaları iyileştirilerek programın daha hızlı çalışması sağlanır.

1.1.8. Bütünleştirilmiş Geliştirme Ortamları

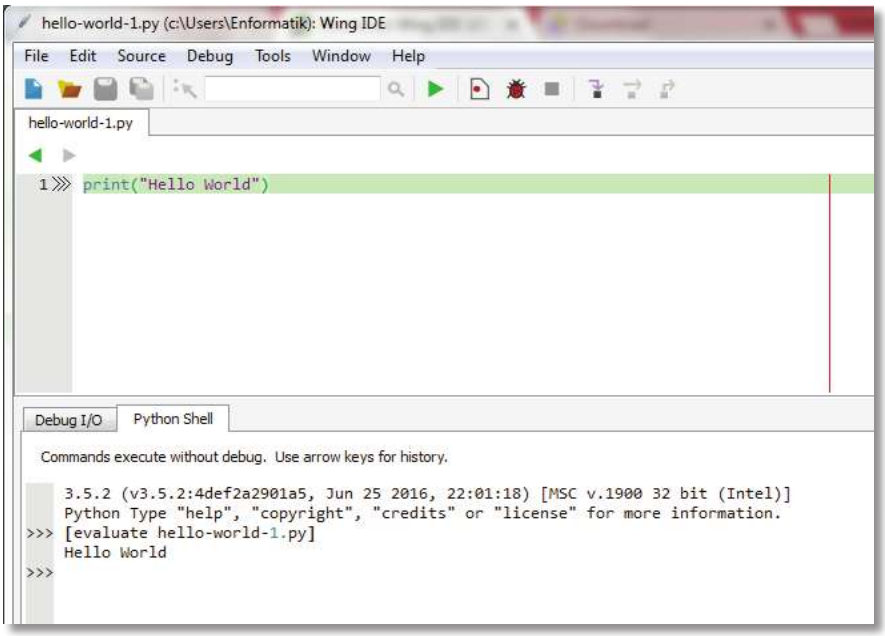
Çoğu yazılım, geliştirici bütünleştirilmiş geliştirme ortamlarını (integrated development environment-IDE) kullanır. Bu ortamlar, editörleri, hata ayıklayıcıları ve diğer programlama yardımcılarını kapsar. Aşağıda en sık kullanılan Python IDE ortamları listelenmiştir.

- IDLE: Python.org web sitesinde yer alan ücretsiz program geliştirme ortamıdır (Şekil 2.3). Bu site içerisinde downloads menü başlığı altında Python için son sürümleri bulmak mümkündür. Ayrıca, birçok işletim sistemi için gerekli dosyalara erişim bağlantıları da bu sayfada yer almaktadır.



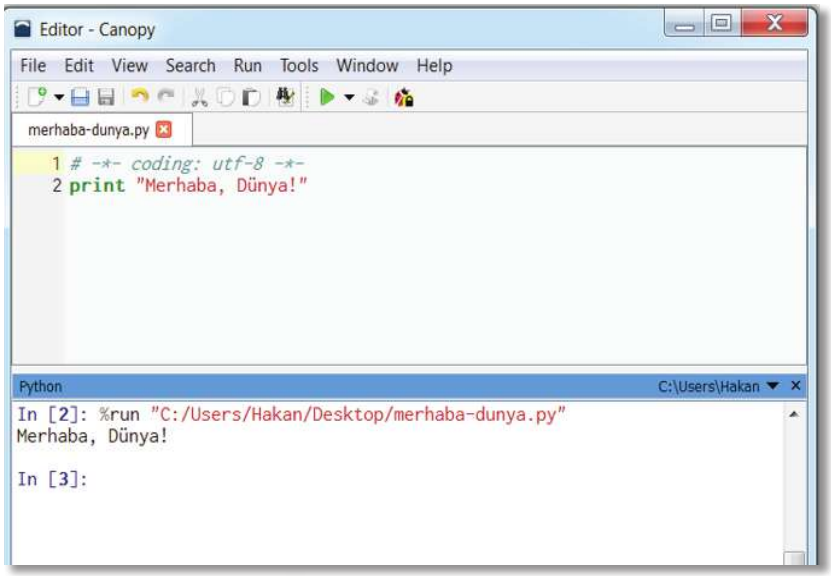
Şekil 2.3: IDLE geliştirme ortamı

- Wing IDE: Wingware Python'da program geliştiriciler için üç farklı türde IDE ortamı sunmaktadır: Wing IDE Professional, Wing IDE Personal ve Wing IDE 101. Bunlardan Wing IDE 101 ücretsiz olup başlangıç seviyesindeki Python kullanıcıları için basit bir IDE ortamı sunmaktadır (Şekil 2.4). https://wingware.com/downloads/wingide-101 adresinden indirilebilir. Bu ortamda kod yazmak, hata bulmak ve çalışma alanını kişiselleştirmek mümkündür.



Şekil 2.4: Wing IDE 101 geliştirme ortamı

- Canopy: Enthought çatısı altında yer alan Canopy, içerisinde birçok farklı programlama dilinde kod yazmayı ve derlemeyi kolaylaştırmaktadır. https://store.enthought.com/downloads/#default adresinden Canopy program geliştirme ortamı ücretsiz indirilebilir (Şekil 2.5).



Şekil 2.5: Canopy program geliştirme ortamı

Düşünelim/Deneyelim

Farklı özellikleri olan ve farklı işletim sistemlerinde çalışan IDE'ler için https://wiki.python.org/moin/PythonEditors adresini ziyaret ediniz.

1.2. Neden Python?

Python, öğrenmesi kolay, tamamen özgür ve ücretsiz bir programlama dilidir. Nesnelere dayalı bir dil olup okunabilirliği yüksektir. Python'un dili başka programlama dilleri ile kıyaslandığında, bunun daha az kod ile işlemleri yapmasının mümkün olduğu görülecektir. Python, bütün işletim sistemleri ile uyum içerisinde çalışmaktadır.

Programlama yapısı içerisinde birçok kütüphaneyi barındırmaktadır. Bu kaynaklarla daha az kod yazmak mümkündür. Pythton ile masaüstünde çalışan uygulamalar geliştirilebileceği gibi, web üzerinde çalışan uygulamalar geliştirmek hatta Rasperry-Pi gibi donanımları da programlamak mümkündür.

1.3. Python Sürümleri

Python programlama dilinin 2016 yılı için en güncel sürümü Python 3.5.2'dir. Bu kitapta yer alan örnekler 3.X sürümlerinde çalışan uygulamalardan oluşmaktadır. Ancak başka kaynaklarda yer alan birçok örneğin daha önceki sürümlerde yazıldığını görmeniz mümkün olabilir. Sürümler arasındaki farklılıklar, özellikle ileride görülecek olan fonksiyonların, print komutu gibi bazı komutların farklı olarak ifade edilmesini kapsamaktadır. Farklı kaynaklardan bulduğunuz örnekler yeni sürümlerde çalışmayabilir. Bu nedenle bu kitapta yer alan söz dizimi kurallarına göre komutların değiştirilmesi gerekmektedir.

Düşünelim/Deneyelim

1. Derleyici ve yorumlayıcı nedir? Benzerlik ve farklılıkları nelerdir?
2. Derlenen ve yorumlanan kodun, kaynak koddan farkı nedir?
3. Python programlama dili nasıl bir dildir ve programı çalıştırmak için ne gerekir?
4. Üst düzey programlama dili kullanmanın yararları nelerdir?
5. Bütünleştirilmiş geliştirme ortamlarının programcıya sağladığı yararlar nelerdir?
6. Programlama dillerinde seviye kavramı neyi ifade etmektedir? Bu konuda bir araştırma yapınız ve bulduklarınızı raporlaştırınız.

2. DEĞERLER VE DEĞİŞKENLER

Bu bölümde;

- ✓ Python programla dilinde değerler ve değişkenler konusunda bilgi sahibi olacak,
- ✓ Veri türlerini tanıyacak,
- ✓ Farklı değişkenleri kullanan program yazabileceksiniz.

2.1. Tam Sayı ve Diziler

Herhangi bir sayı, sayısal değerdir. Örneğin matematikte 4 sayısı tam sayı olarak ifade edilir. Tam sayılar pozitif, negatif ya da sıfır değeri alabilir. Kesirli değerleri içermez. Örneğin 99, 105, 0, -56 ve 7896 birer tam sayıdır. Ancak 4.5 bir tam sayı değildir. Python programlama dili, sayısal ve sözel ifade kullanımını destekler. Python programları, tam sayıları kullanarak işlem yapabilir. Örneğin;

```
print(4)
```

ifadesi 4 değerini ekrana yazdırır. İfadede tırnak (") işareti kullanılmadığına dikkat ediniz. Python tam sayılar dışındaki diğer veri türlerini de desteklemektedir. Bu ifade Python satırlarının temel yapı taşıdır. Tek başına 4 sayısı bir anlam ifade etmez. Ancak yorumlayıcılar, bir anlatım olursa Python ifadelerini değerlendirir. Python etkileşimli yorumlayıcısı, hem ifadeleri hem de anlatımları değerlendirir. Örneğin;

```
>>> 4
```

Komutu yazıldığında, yorumlayıcı satırı okur, işler ve sonucu 4 olarak ekrana yazdırır. x = 10 girilirse bu ifadenin sonuca yönelik bir anlamı olmadığı için etkileşimli yorumlayıcı ekrana hiçbir şey yazdırmaz. Eğer kullanıcı bu ifadeden sonra yalnızca "x" girerse yorumlayıcı bunu değerlendirir ve ekrana x değeri olan 10 sayısını yazar. Şayet kullanıcı bundan sonra ekrana "y" yazarsa, bu değer daha önce tanımlanmadığı için yorumlayıcı hata verir.

Python, toplama işlemi için toplama (+) sembolünü kullanır. Böylece yorumlayıcı bir hesap makinesi gibi işlem yapar.

```
>>> 5 + 4
9
>>> 1 + 6 + 4 + 10
21
>>> print(1 + 6 + 4 + 10)
21
```

Son satırda toplama işleminin, doğrudan print ifadesi içerisinde yer alabildiği de görülmektedir. Şimdi de aşağıdaki işleme bakalım:

```
>>> 16
16
>>> "16"
"16"
>>> '16'
'16'
```

İlk satırda tam sayı olan 16 değeri tek ya da çift tırnak içine alındığında sayı olarak değil metin (alfasayısal kelime yani dizi) olarak işlem görmektedir. Bu ifadeler karakter dizisi ya da dizi olarak anılır. Python, dizileri ayırt etmek ve sınırları belirlemek için hem tek (') hem de çift tırnak (") kullanımına izin vermektedir. Sınırları belirlemek demek, başlangıç ve bitiş noktalarını işaret etmek demektir. Kullanılan ilk ve soldaki sembol (') dizinin başlangıcını, sonra kullanılan ve sağda bulunan sembol (') ise dizinin bitişini ifade eder. Benzer biçimde çift tırnak sembolleri de (") başlangıç ve bitiş için iki kez kullanılmalıdır.

İki sembol tek tırnak (') ve çift tırnak ("), birbiri yerine kullanılamaz. Bu nedenle;

```
>>> "ABC"
"ABC"
>>> 'ABC'
'ABC'
```

ifadeleri doğru çalışırken;

```
>>> 'ABC"
File "<stdin>", line 1
'ABC"
^
SyntaxError: EOL while scanning string literal
>>> "ABC'
File "<stdin>", line 1
"ABC'
```

ifadeleri hataya neden olur.

İfade olarak 4 ve '4' yazımının farklı anlamlar taşıdığına dikkat ediniz. İlk değer bir tam sayı iken, ikinci değer karakter olarak algılandığı için bir dizidir. Python içindeki tüm ifadelerin bir türü vardır. İfadenin türü anlatımın da türünü ifade eder. Bazen ifadelerin türü onların sınıfı olarak belirtilir.

Şu ana kadar yalnızca tam sayı ve dizileri inceledik. Gömülü bir fonksiyon, Python ifadelerinin türünü belirtmek için kullanılabilir.

```
>>> type(4)
<class "int">
>>> type("4")
<class "str">
```

Gömülü str fonksiyonu tam sayı olarak görülen bir ifadeden dizi oluşturur.

```
>>> str(4)
"4"
>>> "5"
"5"
>>> int("5")
5
```

str(4) ifadesi 4 değerini karakter olarak değerlendirir, int('5') ifadesi ise bu karakter değeri tam sayıya dönüştürür.

```
>>> int(4)
4
```

```
>>> str("Python")
"Python"
```

Tahmin edilebileceği gibi, bir programcı için bu dönüşümü yapmaya çoğunlukla gerek yoktur. Bu nedenle str ve int fonksiyonlarının kullanımı, değişkenler kullanılmadıkça çok anlamlı olmayacaktır. Herhangi bir tam sayı, dizi olarak ifade edilebilir, ancak her dizi bir tam sayı olarak ifade edilemez.

```
>>> str(1024)
"1024"
>>> int("sus")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: "sus"
>>> int("3.4")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: "3.4
```

Görüldüğü gibi ne sus ne de 3.4, Python için geçerli bir tam sayıdır. Özetlemek gerekirse eğer dizi içerisinde sayısal ifade varsa int fonksiyonu kullanılarak bu, kolayca tam sayıya çevrilir. Bu arada toplama amacıyla kullanılan + sembolü diziler için farklı biçimde çalışır.

```
>>> 5 + 10
15
>>> "5" + "10"
"510"
>>> "abc" + "xyz"
"abcxyz"
```

Görüldüğü gibi sayı değerleri için toplama işlemi yapan + sembolü, diziler için birleştirme işlemi gerçekleştirir. Bu iki farklı tanımı aynı satırda bulundurmak hataya neden olur.

```
>>> "5" + 10
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can"t convert "int" object to str implicitly
>>> 5 + "10"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: "int" and "str"
```

Ancak int ve str fonksiyonları ile desteklenirse doğru sonuca ulaşılır.

```
>>> 5 + int("10")
```

```
15
>>> "5" + str(10)
"510"
```

Python için type fonksiyonu karmaşık ifadeler için kullanılabilir.

```
>>> type(4)
<class "int">
>>> type("4")
<class "str">
>>> type(4 + 7)
<class "int">
>>> type("4" + "7")
<class "str">
>>> type(int("3") + int(4))
<class "int">
```

Python kapsamında sayıları ifade ederken "," ya da "." gibi ayırma sembolleri kullanılmaz. Dört bin altı yüz elli sekiz, 4658 olarak yazılmalıdır.

2.2. Değişkenler ve Atama

Değişkenler, değerleri korumak için kullanılır. Bu değerler sayı, dizi gibi farklı biçimlerde olabilir. Örneğin;

```
x = 10
```

ifadesi bir atama satırıdır. Atama işlemi bir değeri bir değişken ile eşleştirir. Bu ifadedeki en önemli ayrıntı, atama (=) sembolüdür. Bu ifade ile 10 değeri, x değişkenine atanmaktadır. Bu noktada, x değişkeninin türü tam sayı olur çünkü atanan değer bir tam sayı değerdir. Bir değişkene birden fazla kez atama yapılabilir. Eğer bu sırada öncekinden farklı türdeki bir değer ataması yapılırsa değişkenin türü de değişir. Burada atama (=) sembolünün anlamı matematikte kullanıldığı şeklinden daha farklıdır. Matematikte bu sembol, eşitlik sağlar, bu yüzden bu sembolün sağ ve sol tarafında yer alan ifadelerin birbirine eşit olduğu anlamına gelir. Python dilinde ise atama (=) sembolünün sol tarafında yer alan ifade, sağ taraftaki ifadeyi üstlenir. Bu yüzden bu ifadeyi "5 değeri x değişkenine atandı." ya da x'e 5 atandı." şeklinde yorumlamak doğru olacaktır. Buradaki kullanım, matematikteki kullanımdan farklı olduğu için önemlidir. Matematikte eşitlik ve eşitliğin her iki yanında simetri söz konusudur yani matematiksel açıdan x = 5 ve 5 = x olacak biçimde her iki ifadede doğru iken Python kapsamında bu, simetri olmadığı için 5 = x ifadesi hatalı olacaktır. Çünkü tam sayı bir değişken olmadığı için atama yapma davranışı da yanlıştır.

Bir değişkene defalarca farklı değerler atayabiliriz.

```
x = 10
print(x)
x = 20
print(x)
x = 30
print(x)
```

Görüldüğü gibi print fonksiyonu her satırda aynı olmasına rağmen her seferinde farklı bir değer yazdırılmaktadır.

```
10
20
30
```

Bu program bize, değişkenlere bağlı durumlardaki davranışları her zaman öngörmediğimizi göstermektedir. Bazı fonksiyonlar bir ya da daha fazla değişkene bağımlı hareket edebilir.

```
x = 10
print("x = " + str(x))
x = 20
print("x = " + str(x))
x = 30
print("x = " + str(x))
```

Yukarıdaki program aşağıdaki çıktıyı oluşturur.

```
x = 10
x = 20
x = 30
```

Burada print fonksiyonu içerisinde kullanılan toplama + işlemi dizileri birleştirmek amacıyla kullanılmaktadır. Bu nedenle "x =" + x ifadesi çıktıda görülen sonucu üretmektedir. Bu örnekte print fonksiyonu, iki parametre kabul etmektedir.

```
print("x =", x)
```

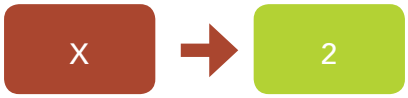
İlk parametre "x =" dizisi ve ikinci parametre ise x değişkeni ile eşleşmiş değerdir. Print fonksiyonu her biri virgül ile ayrılmış çoklu parametre kullanımına izin verir, hepsini sıra ile yazdırır ve yazdırırken her biri arasına bir boşluk değeri bırakır. Bir programcı tek bir satırda çok ögeli yaklaşımı kullanarak birden fazla değer ataması yapabilir. Bu işleme **çoklu atama** denilir.

```
x, y, z = 100, -45, 0
print("x =", x, " y =", y, " z =", z)
```

Çoklu atamada değerler virgüller ile ayrılmış olarak listelenir. x, y, z = 100, -45, 0 ifadesinde x, y, z bir grup çoklu ögeyi; 100, -45, 0 ise diğer grup çoklu ögeyi ifade eder. Çoklu atama şu şekilde çalışır: Sol taraftaki çoklu öge grubundaki ilk değişken, sağ taraftaki çoklu öge grubunun ilk elemanı ile eşleşir (x = 100). Benzer şekilde ikinci ve sonraki ögeler de birbiri ile eşleşir. İkinci ögelerin eşleşmesi, y = -45 ve son ögelerin eşleşmesi, z = 0 ile sonuçlanır. Bu işlemden sonraki durum aşağıdaki gibidir:

```
x = 100 y = -45 z = 0
```

Çoklu atama, sol taraftaki çoklu öge grubu ile sağ taraftaki çoklu öge grubunun sayıları eşit ise gerçekleşir. Atanan değer bir değişken ismini bir nesneye bağlar.

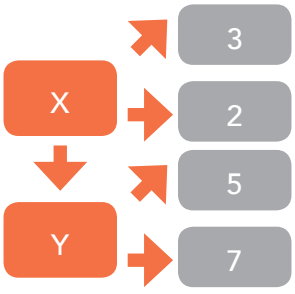


Şekil 2.6: Değişkene değer atama

Örnek olarak x = 2 ifadesini inceleyelim (Şekil 2.6). Bir kutu değişkeni ifade eder ve bu, değişken ismi ile adlandırılır. Diğeri kutudan nesneye doğru yönelen ok, değişkenin bağlandığı nesneyi gösterir. Bu durumda ok 2 değerini içeren başka bir kutuyu işaret eder. İkinci kutu 2 değerinin ikilik düzendeki karşılığını içeren hafıza yerini temsil eder.

Bilgisayar her bir program satırını işledikçe değişkenlerin değerlerinin nasıl değiştiğini gözleyelim.

```
x = 2
y = 5
x = 3
x = y
y = 7
```



Burada özellikle x = y ifadesine dikkat ediniz. Bu atama hem x hem de y değerinin aynı değer ile eşleştiği anlamına gelmektedir. Sonra y değerinin değişmesi x değerini etkilemez.

Programın çalışması sırasında bir değişkenin yalnızca değeri değil, türü de değişebilir.

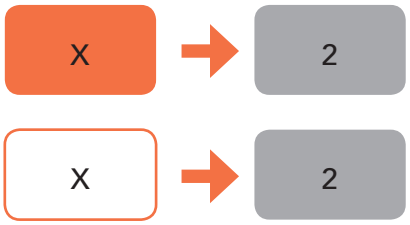
```
a = 10
print("a değişkeninin ilk değeri", a, "ve tipi", type(a))
a = "ABC"
print("a değişkeninin yeni değeri", a, "ve tipi", type(a))
```

Bu ifade aşağıdaki çıktıyı oluşturur.

```
a değişkeninin ilk değeri 10 ve tipi <class "int">
a değişkeninin yeni değeri ABC ve tipi <class "str">
```

Programcılar program akışı içinde bir değişkenin türünü nadiren değiştirmeye gerek duyarlar. Bir değişkenin program çalıştığı sürece belli bir anlamı ve rolü olmalıdır ki bu, genellikle değişmez. Herhangi bir değer atanmamış bir değişken, tanımsız değişken olarak ifade edilir. Böyle bir değişken program içerisinde kullanıldığında hata ile karşılaşılır. Nadiren daha önce tanımlanmış bir değişkeni tanımsız bir değişkene dönüştürmek isteriz. Bu işlemi del satırını kullanarak gerçekleştiririz.

```
>>> x = 2
>>> x
2
>>> del x
>>> x
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name "x" is not defined
```



Kullanılan del komutu sil anlamına gelir ve yorumlayıcı, içerisinden ya da program içerisinden değişken tanımını siler. Örneğin a, b ve c önceden tanımlanmış değişkenler ise;

```
del a, b, c
```

ifadesi bütün değişkenlerin silinmesini sağlar.

2.3. Reel Sayılar

Pek çok hesaplamalı işlem, kesir parçası olan sayıları kullanır. Örneğin bir dairenin çevresini ya da alanını hesaplamak için π değerine ihtiyacımız vardır ve bu değer yaklaşık 3,14159 olarak ifade edilir. Python, bu şekilde noktalı sayılarla işlem yapar ve bu sayılara reel ya da gerçek sayı denir. İsminden de anlaşılacağı gibi matematiksel hesaplamalar sırasında ondalık nokta, önemli sayıları ifade etmek için farklı basamaklara kayabilir. Python ile programlama yaparken bu tanımı yapmak için ***float*** kelimesi kullanılır.

```
>>> x = 5.62
>>> x
5.62
>>> type(x)
<class "float">
```

Python için reel sayıların alabileceği en küçük ve en büyük değer ile duyarlılık düzeyi, kullanılan makine ve uygulamaya göre değişiklik gösterebilir. Reel sayılar da tam sayılar gibi pozitif ve negatif olabilir. Yaygın olarak geçerli olan durum aşağıda görülmektedir.

Tanım	Bellek	Minimum Değer	Maksimum Değer	Duyarlılık
float	64 bit	2,22507 x 10-308	1,79769 x 10-308	15 basamak

Tablo 1: Float değeri

Çoğu programlama editörü, alt ya da üst simge ve özel sembollerin kullanımını desteklemediği için sayıların gösterimi farklılaşmaktadır. Python ile kod yazarken 6.022×10^{23} yerine $6.022e^{23}$ olarak yazmamız gerekir. Burada "e" karakterinin solunda kalan kısım normal sayı; sağında kalan kısım ise 10 üzerindeki sayıdır. Simge olarak "e" yerine "E" de kullanılabilir.

Reel sayılardan farklı olarak tam sayılar kesirli ifadeleri içeremez. Reel bir sayıyı bir tam sayıya dönüştürmenin iki temel yolu vardır:

- Yuvarlama: Reel sayıya en yakın tam sayıya ulaşmak için kesrin belirli bir miktarı eklenerek ya da çıkarılarak yuvarlama yapılır.
- Kesme: Sayının kesirli kısmı tamamen göz ardı edilir.

Yuvarlama ve kesme işlemlerinin sonuçlarının ne şekilde farklılaştığını gözlemleyebiliriz:

```
>>> 28.71
28.71
>>> int(28.71)
28
>>> round(28.71)
29
>>> round(19.47)
19
>>> int(19.47)
19
```

Görüldüğü gibi kesme, her zaman aşağı doğru yuvarlamaktadır.

Yuvarlama yapmak için round fonksiyonunu noktadan sonra belirli bir sayıda basamağı koruyarak sonuç elde etmek için de kullanabiliriz.

```
>>> x = 93.34836
>>> x
93.34836
>>> round(x)
93
>>> round(x, 2)
93.35
>>> round(x, 3)
93.348
>>> round(x, 0)
93.0
>>> round(x, 1)
93.3
>>> type(round(x))
<class "int">
>>> type(round(x, 1))
<class "float">
>>> type(round(x, 0))
<class "float">
```

Görüldüğü gibi tek argüman tam sayı sonucu verirken iki argüman reel sayı sonucu vermektedir.

Kullanılan ikinci argüman negatif bir değerde olabilir: round(n, r) ifadesi n sayısını 10-r ile çarpma işlemini yapar. Örneğin round(n, -2) n sayısını 10-2 ile çarpmaktadır.

```
>>> x = 28793.54836
>>> round(x)
28794
```

```
>>> round(x, 1)
28793.5
>>> round(x, 2)
28793.55
>>> round(x, 0)
28794.0
>>> round(x, 1)
28793.5
>>> round(x, -1)
28790.0
>>> round(x, -2)
28800.0
>>> round(x, -3)
29000.0
```

Python için round fonksiyonu tam sayılar için de kullanılabilir. Burada ilk argüman tam sayı, ikinci argüman ise yuvarlama için sola doğru kayması beklenen basamak sayısını ifade eder. İkinci değer pozitif bir değerse sayının orijinal değeri elde edilir. Her durumda sonuç her zaman tam sayıdır.

```
>>> round(65535)
65535
>>> round(65535, 0)
65535
>>> round(65535, 1)
65535
>>> round(65535, 2)
65535
>>> round(65535, -1)
65540
>>> round(65535, -2)
65500
>>> round(65535, -3)
66000
>>> round(65535, -4)
70000
>>> round(65535, -5)
100000
>>> round(65535, -6)
0
```

2.4. Belirteçler

Matematik işlemlerinde genellikle x ve y tek karakterden oluşan değişkenler için kullanılır. Programcıların bundan kaçınarak çok daha uzun, anlamlı ve açıklayıcı değişken isimleri seçmesi gerekir. Bu nedenle t, at, y ve s gibi isimler yerine, toplam, araToplam, yükseklik ve süre gibi içeriğindeki değeri ifade eden değişken isimleri kullanmak çok daha etkilidir. Değişken ismi program içerisindeki kullanım amacına uygun olmalıdır. Değişken isimleri ne kadar doğru seçilirse program okuyan kişiler için o

kadar çabuk anlaşılır ve anlamlı olur.

Python, büyük küçük harf duyarlıdır ve değişken isimleri için kesin kurallar kullanır. Bir değişken ismi belirteç için bir örnektir. Belirteç, ögeleri isimlendirmek için kullanılan kelimedir. Belirteçler fonksiyon, sınıf ve metot gibi parçaları isimlendirmek için de kullanılır.

Belirteçlerin özellikleri şöyle sıralanabilir:

1. Bir belirteç en az bir karakter içermelidir.
2. Belirtecin ilk karakteri harf (küçük ya da büyük) ya da alt çizgi olmalıdır (ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_).
3. Devam eden karakterler, harf (küçük ya da büyük), alt çizgi ya da sayı olabilir (ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789).
4. Belirteçlerde boşluk dâhil diğer özel karakterler kullanılamaz.
5. Programlama diline ait ayrılmış kelimeler belirteç olarak kullanılamaz (Reserved Words).
6. Son olarak print, int, str ya da type gibi özel kelimeler kullanılabilmesine rağmen iyi bir program için bunlar asla önerilmez.
7. Türkçe karakterler kullanılamaz.

Python için Ayrılmış Kelimeler			Python için Doğru Belirteçler	Python için Yanlış Belirteçler
and	del	from		☒ ara-toplam (- sembolü kullanılamaz)
as	elif	global	☑ x	☒ ilk değer (boşluk kullanılamaz)
assert	else	if	☑ a2	☒ 4ogrenci (sayı ile başlayamaz)
break	except	import	☑ Toplam	☒ *2 (* sembolü kullanılamaz)
class	False	in	☑ Toplam_Brut	☒ öğrenci (Türkçe karakter içeremez)
continue	finally	is	☑ Anahtar_10	☒ class (class ayrılmış kelime olduğu için kullanılamaz)
def	for	lambda		

Şekil 2.7: Python için özel durumlar

3. İFADELER VE ARİTMETİK İŞLEMLER

Bu bölümde;

- ✓ Python programlama dilinde kullanılan ifadeler ve aritmetik işlemler konusunda bilgi sahibi olacak,
- ✓ Operatör, işlem önceliği ve birleşim durumlarını açıklayabilecek,
- ✓ Python dilinde ortaya çıkan hata türlerini kavrayacaksınız.

3.1. Sabit Değerler

Sabit değerler (ör. 34) ve değişkenler (ör. x) basit ifadelerdir. Değerleri ve değişkenleri birleştirmek ve daha karmaşık ifadeler oluşturmak için operatörler kullanabiliriz. Aşağıdaki örnekte kullanıcı tarafından girilen 2 sayıyı toplamak için toplama operatörü kullanabiliriz.

```
deger1 = int(input("Bir sayı giriniz: "))
deger2 = int(input("Lütfen diğer sayıyı giriniz: "))
toplam = deger1 + deger2
print(deger1, "+", deger2, "=", toplam)
```

3.2. Python'da Sık Kullanılan Aritmetik İkili Operatörler

Python programlama dilinde kullanılan aritmetik operatörler aşağıdaki tabloda yer almaktadır:

İfade	Anlamı
x+y	Eğer x ve y rakamsa, x, y'ye eklenir. Eğer x ve y harf dizisi ise x ile y birleştirilir.
x-y	Eğer x ve y rakamsa, x'ten y çıkarılır.
x*y	Eğer x ve y rakamsa, x ile y çarpılır. Eğer x harf dizisi ise ve y rakamsa, x, y kez sıralanır. Eğer x rakamsa ve y harf dizisi ise y, x kez sıralanır.
x/y	Eğer x ve y rakamsa, x, y'ye bölünür.
x//y	Eğer x ve y rakamsa, x'in içinde y değişkenin katsayısını arar. Örn. 10 // 4 = 2
x%y	Eğer x ve y rakamsa, bölme işleminde x / y kalan kısmını verir.
x**y	Eğer x ve y rakamsa, x, y'nin kuvvetiyle çarpılır.

Tablo'da belirtilen operatörlerin ikili operatör olarak belirtilmesinin nedeni 2 işlenen (operand) üzerinde çalışmasıdır. x = y + z deyiminde, atama operatörünün sağ tarafında y + z bir toplamsal ifadedir. + operatörünün iki işleneni y ve z'dir.

- +, -, *, //, %, or ** operatörlerini 2 sayıya uyguladığımızda sonuç tam sayı olacaktır.
- print(25//4, 4//25) yazdığımızda 6 ve 0 sonuçlarını verecektir.
- Mod operatörü (%) sayı bölümünden kalanı hesaplar. print(25%4, 4%25) yazdığımızda 1 ve 4 sonuçlarını verecektir.
- / operatörü 2 sayıya uygulandığında ondalıklı sayı döndürür. print(25/4, 4/25) yazdığımızda 6.25 0.16 sonuçlarını verecektir.

3.3. Karışık Türlü İfadeler

İfadeler tam sayı ve ondalıklı sayı değerlerini içerebilir. Örneğin,

```
x = 4
y = 10.2
toplam = x + y
```

X, bir tam sayıdır ve Y, bir ondalık sayıdır. x + y ifadesin türü , / operatörü hariç, sadece tam sayılar içeren aritmetik ifadeler bir tam sayı sonuç üretir. Ondalıklı sayılara uygulanan tüm aritmetik operatörler bir ondalıklı sonuç üretir.

3.4. Operatör Önceliği ve Birleşim

Farklı operatörler aynı ifadede yer aldığında, aritmetiğin normal kuralları uygulanır. Tüm Python operatörlerinde öncelik (precedence) ve birleşim (associativity) vardır.

- Öncelik, bir ifade iki farklı türde operatörler içerdiği zaman, hangisi ilk olarak uygulanacak?
- Birleşim, bir ifade aynı önceliğe sahip iki operatörleri içerdiği zaman, hangisi ilk olarak uygulanacak?
- Çarpımsal operatörler (*, /, // ve %) birbirleri ile eşit önceliğe sahiptir ve toplamsal operatörler (ikili + ve -) birbirleri ile eşit önceliğe sahiptir.
- Çarpımsal operatörleri, toplamsal operatörleri üzerinde önceliğe sahiptir.
- Standart aritmetikte olduğu gibi bir Python programcısı öncelik kurallarını geçersiz kılmak için ayraçları kullanabilir ve çarpmadan önce toplama işleminin yapılmasını sağlayabilir.

Her satırdaki operatörler, altındaki operatörlerden daha yüksek bir önceliğe sahiptir. Bir satır içerisinde yer alan operatörler aynı önceliğe sahiptir.

Arity	Operatörler	Birleşim
İkili	**	Sağ
Tekli	+, -	
İkili	*, / , // , %	Sol
İkili	+, -	Sol
İkili	=	Sağ

3.5. İfadeleri Biçimlendirme

Python, aritmetik ifadeleri biçimlendirmek için önemli bir esneklik sunar;

```
3x + 2y-5
```

Cebirin aksine, Python dilinde örtülü hiçbir çarpma yoktur. Bu, 3x'i , 3 * x olarak yazmak gerektiği anlamına gelir. * operatörünü atamayabiliriz. Boşluk, operatör önceliğini etkilemez.

```
print(3*x + 2*y -5)
print(3*x+2*y-5)
print(3 * x + 2 * y - 5)
print(3 * x+2 * y-5)
print(3 * (x+2) * (y-5))
print(3*(x + 2)*(y - 5))
```

3.6. Yorumlar

Python programlama dilinde yazılan programlar uzadıkça karmaşık bir hâl alabilir. Bu da zamanla okumayı ve hatta hataları bulmayı zorlaştırabilir. Bu durumu ortadan kaldırmak ve programcıya yardımcı olması amacı ile programa küçük notlar hâlinde açıklama eklenmesi gerekebilir. # işareti ile program içerisine yorum yazmak, açıklama eklemek mümkündür. Olası kullanım durumları aşağıda belirtilmiştir:

```
# Girilen dakika değerinin kaç saat olduğunu hesaplar.
yuzde=(dakika*100)/60
```

Bir satırın sonuna da yorum eklenebilir.

```
yuzde=(dakika*100)/60 # dikkat: bir saatten küçük değerler ondalıklı sonuç verecektir.
```

Yorum satırlarını derleyici, göz ardı eder ve ilgili satırın sonuna kadar programın çalışmasına herhangi bir etki etmez. Yorum satırları ile küçük hatırlatma yapmak veya daha başka programcılar için uyarılar eklemek program geliştirme sürecine yardımcı olabilir.

3.7. Hatalar

Python'da, üç genel hata türü vardır: söz dizimi hataları, çalışma zamanı istisnaları ve hataları. Yorumlayıcı tüm geçerli Python programlarını çalıştırmak için tasarlanmıştır. Yorumlayıcı Python kaynak dosyasını okur ve yürütülebilir bir forma çevirir. Bu, çeviri aşamasıdır (Translation). Yorumlayıcı bir çeviri aşamasında geçersiz program deyimi algılarsa bu, programın yürütülmesini sonlandıracak ve bir hata raporu verecektir. Bu tür hatalar programcının dili kötü kullanmasından kaynaklanmaktadır. Bir söz dizimi hatası, bir Python deyimi makine diline çevirmeye çalışırken yorumlayıcı tarafından algılanabilen yaygın bir hatadır. Aşağıdaki programda, yorumlayıcı bir hata mesajı verecektir. Hatalı bir atama işlemi yapmaya çalışmaktadır.

```
>>> y = 5
>>> x = y + 2
>>> y + 2 = x
```

```
File "error.py", line 3
y + 2 = x
^
SyntaxError: can't assign to operator
```

Diğer yaygın söz dizimi hataları şunlardır:

- Eşleşmeyen ayraç gibi basit yazım hataları ()3 + 4))
- Eşleşmeyen harf dizini tırnak işaretleri ((‘hello”))
- Hatalı girinti (faulty indentation)

Yorumlayıcı, söz dizimi hatalarını programı çalıştırmaya başlamadan önce algılar ve bu nedenle söz dizimi hatalarını içeren bir programın herhangi bir parçasını çalıştırmaz.

3.7.1. Çalışma Zamanı Hataları

Doğru yazılmış bir Python programının hâlâ sorunları olabilir. Bazı dil hataları programın yürütülmesi durumuna bağlıdır. Yorumlayıcı bir istisna yaratır. Çalışma zamanı istisnaları yorumcunun çeviri aşamasından sonra ve programın yürütme aşamasında ortaya çıkmaktadır.

Yorumlayıcı aşağıdaki gibi söz dizimsel olarak doğru bir ifade için bir istisna verebilir.

```
x = y + 2 d
```

Eğer y değişkeni henüz atanmamışsa;

`(NameError: name "y" is not defined)` hatası mesajını verir.

Kullanıcıyı 32 (pay) ve 0 (payda) olarak yazdığınızda

```
Zero Division Error: division by zero
```

veya harf dizinini bölmeye kalktığınız takdirde

`unsupported operand type(s) for /: "str" and "int"` uyarısı alabilirsiniz.

3.7.2. Mantık Hataları

Bölünen/bölen ifadesi yerine bölen/bölünen olarak değiştirdiğinizde etkilerini düşünün. Program çalışır ve kullanıcı bölünene sıfır değeri girmedikçe, yorumlayıcı hata raporu vermeyecektir. Ancak hesaplanan cevap genel olarak doğru değildir. Program sadece bölünen ile bölen eşit olduğu zaman doğru cevap yazdıracaktır. Program bir hata içermektedir. Fakat yorumlayıcı sorunu algılayamaz. Bu tür bir hata, bir mantık hatası olarak bilinir. Yorumlayıcı, mantık hatalarının konumu için herhangi bir fikir sağlamakta güçsüzdür. Mantık hataları, bu nedenle, bulma ve onarmakta en zor olma eğilimindedir. Yorumlayıcı mantık hatalarına yönelik hiçbir yardımda bulunmaz.

3.8. Aritmetik Örnekler

Örnek: Sıcaklığı Fahrenheit derecesinden Celcius derecesine dönüştürmek istediğinizi varsayalım.

```
>>> # Sıcaklık değerini okumak için
>>> dereceF = float (input ("Sıcaklığını F derece olarak girin:"))
>>> # Dönüşümü gerçekleştirin
>>> dereceC = 9/5 * (dereceF - 32)
>>> # Sonucu bildir
>>> print (dereceF, "derece F =", dereceC, "C derece")
Sıcaklığını F derece olarak girin: 212
212 ° F = 100.0 ° C
```

Örnek: Kullanıcının girdiği saniyeleri, saat, dakika ve saniye olarak parçalara ayıran programda tam sayı bölme ve modül kullanır.

```
>>> saniye = int (input ("saniye sayısını girin:"))
>>> saat = saniye // 3600 # 3600 saniye = 1 saat
```

```
>>> saniye = saniye% 3600
>>> dakika = saniye // 60 # 60 saniye = 1 dakika
>>> saniye = saniye% 60
print (saat, "sa", dakika, "dk", saniye, "sn")
```

Kullanıcı 10000 girerse program 2 saat, 46 dakika, 40 saniye yazdırır.

Örnek: Dijital saat ekranlarında yapıldığı gibi dakika ve saniyelerde tek haneli değerlerin önüne sıfır koymak için bazı ekstra aritmetik gerekir.

```
>>> saniye = int (input ("saniye sayısını girin:"))
>>> saat = saniye // 3600 # 3600 saniye = 1 saat
>>> saniye = saniye% 3600
>>> dakika = saniye // 60 # 60 saniye = 1 dakika
>>> saniye = saniye% 60
>>> print (saat, ":", sep = "", end = "")
>>> onlar = dakika // 10
>>> birler = dakika % 10
>>> print (onlar, birler, ":", sep = "", end = "")
>>> onlar = saniye // 10
>>> birler = saniye% 10
>>> print (onlar, birler, sep = "")
```

3.9. Aritmetik İfadeler

Python basit aritmetik aracılığıyla bir değişkeni değiştiren bir deyimin basitleştirilmesinde daha genel bir yol sağlar. Örneğin x= x + 5 deyimi x + = 5 olarak kısaltılabilir. Bu ifade " x'i 5 arttır." anlamına gelir. x * = y + z deyimi ile x = x * (y + z) deyimi aynıdır.

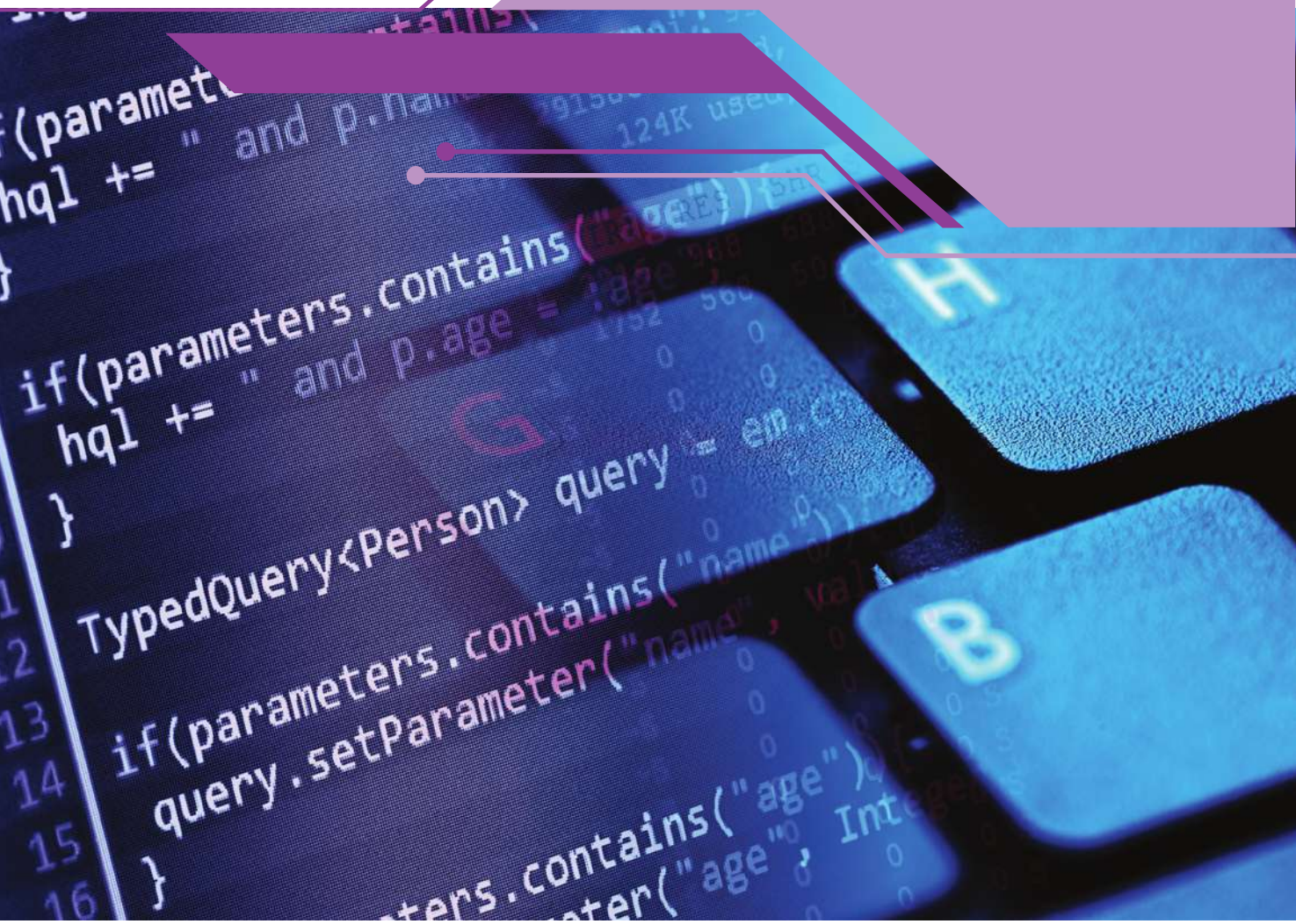
Örnek:

```
>>> y=5
>>> y=y+15
>>> x=5
>>> x+=15
>>> print("x-->",x," y-->",y)
x--> 20 y--> 20
```

Bu örnekte ilk olarak x ve y değişkenlerine 5 değeri aktarılmış sonra aynı değişkenlerin üzerine 15 değeri farklı yöntemlerle eklenmiştir. Print komutuyla ekrana yazdırıldığında çıkan değerlerin aynı olduğu görülecektir.

İki veya daha fazla ifadeyi karşılaştırmak gereken koşullu durumlarda koşul ifadeleri kullanılır. Koşul ifadeleri, program yazarken çok kullanılan ifadelerden biridir. Farklı türden durum ve/veya koşulları ifade etmek için kullanılan koşul ifadeleri sırayla açıklanmıştır.

4. KOŞULLU DURUMLAR



Bu bölümde;

- ✓ Program yazma sürecinde kullanılan koşullu durumlar konusunda bilgi sahibi olacak,
- ✓ İç içe koşul durumu ifade etmeyi kavrayacak,
- ✓ Koşul durumlarında yapılan hatalar konusunda fikir sahibi olacaksınız.

4.1. Boolean İfadesi

Bilgisayar bilimi temelde 0 ve 1 değerleri üzerine kurulmuştur; 0 değeri False(Yanlış), 1 değeri True(Doğru) demektir. Bu ifadelere Boolean (bool) İfadeleri denir. Doğru ve Yanlış değerleri korumak için kullanılan tipe bool adı verilmektedir. Sadece iki Boolean ifade değeri vardır:

- True (Doğru) (1)
- False (Yanlış) (0)

Python açısından büyük harfle başlamaları önemlidir. Boolean deyimi, İngiliz Matematikçi George Boole'in soyadından gelmektedir. Soyut matematiğin bir dalı olan Boole Cebiri (mantık cebiri), George Boole'nin mantıksal ifadelerle ilgili çalışmalarına ithaf edilmiştir.

4.2. Python'da İlişkisel Operatörler

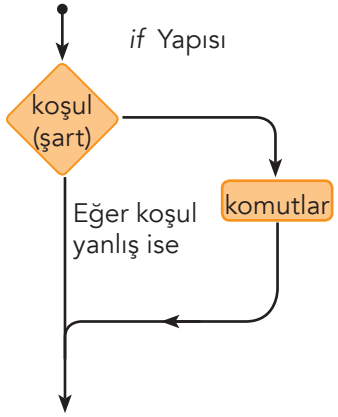
İfade	Anlamı
x==y	Eğer x ve y birbirine eşitse (matematiksel olarak) doğrudur, değilse yanlıştır.
x<y	Eğer x, y'den küçükse doğrudur; değilse yanlıştır.
x<=y	Eğer x, y'den küçük ya da eşitse doğrudur; değilse yanlıştır.
x>y	Eğer x, y'den büyükse doğrudur; değilse yanlıştır.
x>=y	Eğer x, y'den büyük ya da eşitse doğrudur; değilse yanlıştır.
x!=y	Eğer x, y'den farklı ise (büyük ya da küçük) doğrudur; değilse yanlıştır.

Python'da ilişkisel operatörlere örnekler

İfade	Değer
10 < 20	True
10 >= 20	False

4.3. "if" İfadesi

Türkçede EĞER anlamına gelen if ifadesi, adından da anlaşılacağı üzere, bir koşula bağlı durumları kontrol etmek amacıyla kullanılır.



if ifadesinin yapısı:

```
if koşul:
        ifadeler
```

şeklindedir.

Burada koşul "true" değer alıyorsa yani koşul sağlanıyorsa blok kısmındaki ifadeler gerçekleşecektir. Eğer koşul "false" değer alıyorsa yani koşul sağlanmıyorsa blok kısmındaki ifadeler gerçekleşmeden program devam edecektir.

```
print("Lütfen bölme için iki sayı giriniz.")
bolum=int(input("Lütfen bölme için ilk sayınızı giriniz:")
bolen=int(input("Lütfen bölme için ikinci sayınızı giriniz:")
if bolen!=0:
     print(bolum,"/",bolen,"=", bolum/bolen)
```

Ekran çıktısı aşağıdaki gibi olur:

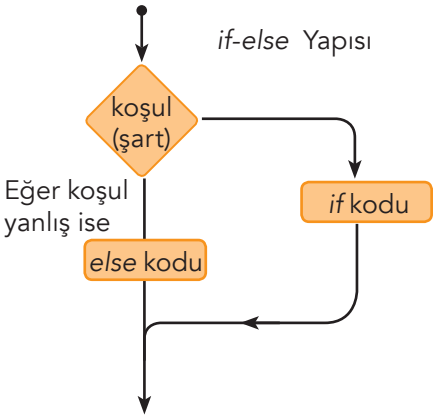
```
Lütfen bölme için iki sayı giriniz.
Lütfen bölme için ilk sayınızı giriniz:32
Lütfen bölme için ikinci sayınızı giriniz:8
32/8=4.0
```

Başka değerler için test edecek olursak:

```
Lütfen bölme için iki sayı giriniz.
Lütfen bölme için ilk sayınızı giriniz:32
Lütfen bölme için ikinci sayınızı giriniz:0
```

4.4. "if/else" İfadesi

"if/else" ifadesi, "if" ifadesi ile birlikte çalışır ve if koşullarının sağlanmadığı tüm durumları kapsar.



if/else ifadesinin yapısı:

```
if koşul:
        ifadeler
else:
        ifadeler
```

else, isteğe bağlı bir ifadedir ve if bloku ile birlikte sadece bir kez kullanılır.

```
print("Lütfen bölme için iki sayı giriniz.")
bolum=int(input("Lütfen bölme için ilk sayınızı giriniz:")
bolen=int(input("Lütfen bölme için ikinci sayınızı giriniz:")
if bolen!=0:
     print(bolum,"/",bolen,"=", bolum/bolen)
else
     print("Sıfıra bölme işlemi yapılamaz.")
```

Ekran Çıktısı

```
Lütfen bölme için iki sayı giriniz.
Lütfen bölme için ilk sayınızı giriniz:32
Lütfen bölme için ikinci sayınızı giriniz:0
Sıfıra bölme işlemi yapılamaz.
```

Örnek

```
# Ağacın boyunu sorgulayan program.
boy = input("Ağacın boyu kaç cm? ")
if int(boy) > 150:
  print ("Ağacın boyu uzun demek ki!")
else:
  print ("Ağacın boyu kısa")
```

4.5. Birleşik Boolean İfadesi

Boolean ifadesi ile bir ilişkisel operatörle birleştirilerek daha karmaşık Boolean ifadeleri oluşturabilir. Bu tür durumlarda 3 farklı mantıksal operatörden yararlanılabilir: **and, or ve not.** Mantıksal operatörler aracılığı ile iki veya daha fazla Boolean ifadesinin kullanıldığı deyimlere **Birleşik Boolean İfadeleri denir.**

Birleşik Boolean İfadeleri oluşturan mantıksal operatörler –e1 ve e2 örneği

e1	e2	e1 and e2	e1 or e2	not e1
False (yanlış)	False (yanlış)	False (yanlış)	False (yanlış)	True (doğru)
False (yanlış)	True (doğru)	False (yanlış)	True (doğru)	True (doğru)
True (doğru)	False (yanlış)	False (yanlış)	True (doğru)	False (yanlış)
True (doğru)	True (doğru)	True (doğru)	True (doğru)	False (yanlış)

Mantıksal operatörlerden and ve or sola birleşmeli, not sağa birleşmelidir. Örneğin,

x <= y and x <= z ifadesi
(x <= y) and (x <= z) olarak işlem görür.
x = 10, y = 20 olarak veriliyor. Buna göre aşağıda verilen kod örneklerini inceleyiniz.

```
b = (x == 10)                  # b'ye True değerini atar.
b = (x != 10)                  # b'ye False değerini atar.
b = (x == 10 and y == 20)      # b'ye True değerini atar.
b = (x != 10 and y == 20)      # b'ye False değerini atar.
b = (x == 10 and y != 20)      # b'ye False değerini atar.
b = (x != 10 and y != 20)      # b'ye False değerini atar.
b = (x == 10 or y == 20)       # b'ye True değerini atar.
b = (x != 10 or y == 20)       # b'ye True değerini atar.
b = (x == 10 or y != 20)       # b'ye True değerini atar.
b = (x != 10 or y != 20)       # b'ye False değerini atar.
```

4.6. Pass İfadesi

Pass ifadesi Python'da herhangi bir işlem yapmadan geçeceğimiz durumlarda kullanılır. Kısaca "Hiçbir şey yapmadan yola devam et!" anlamı katar.

```
if x == 2:
     print(x)
else:
     pass # x 2'ye eşit değilse hiçbir şey yapma
if x == 2:
     print(x) # yalnızca x 2'ye eşitse yazdır
```

4.7. Kayan Noktalı Eşitlik

Eşitlik operatörü (==) gerçek eşitlik olup olmadığını kontrol eder. Ancak kayan noktalı sayılarla işlem yaparken bu durum sorun oluşturabilir. Örneğin,

```
d1 = 1.11 - 1.10
d2 = 2.11 - 2.10
     print("d1 =", d1, " d2 =", d2)
     if d1 == d2:
          print("Aynı")
     else:
          print("Farklı")
```

Normalde, matematiksel işlem yapıldığında aşağıdaki gibi bir eşitliğin olduğu görülür.

1.11 - 1.10 = 0.01 = 2.11 - 2.10

Ancak; bilgisayar sistemlerinde tüm işlemler bitler (0 ve 1) ile yapıldığından sayıların hafızadaki değerleri de bu şekilde tutulur. Kayan noktalı sayılar bilgisayar sisteminde ikilik tabanda karşılığı ve bunun köküst şeklinde temsil edilir. Bu sebepledir ki;

d1 = 0.010000000000000009

d2 = 0.009999999999999787

olduğu için, yazılan kod çalıştırıldığı zaman d1 ve d2'nin farklı olduğunu yazacaktır.

4.8. İç İçe Koşul İfadeleri

Karşılaştırma yapıları kullanırken bazı durumlarda istenilen koşulların birden fazla şarta aynı anda uyması istenebilir. Bu durumda koşul yapılarının birbirinin içinde kullanılması gerekir. Bu şekilde bir yapı kullanıldığında istenilen komut veya komut kümelerinin yapılması için iki koşul ifadesinin de True olması gerekir. Örneğin;

```
deger = int(input("Lütfen 0….10 aralığında bir tam sayı giriniz: ")
     if deger >= 0 and deger <= 10: # İkili koşul kontrolü
          print("aralıkta")
     print("tamamlandı")
```

yerine, her koşul ayrı olacak şekilde daha basit bir iç içe koşul yazılabilir.

```
deger = int(input("Lütfen 0….10 aralığında bir tam sayı giriniz: ")
     if deger >= 0: # İlk kontrol
          if deger <= 10: # İkinci kontrol
               print("aralıkta")
     print("tamamlandı")
```

4.9. Çok Yönlü Karar İfadeleri

Basit if/else ifadesinde iki farklı koşul varken çok yönlü karar ifadelerinde daha fazla koşulun gerçekleşme durumuna göre işlem yapılır. Bunun için kod yazılırken iç içe if/else ifadeleri gerekir. Örneğin;

```
deger = int(input("Lütfen 0 yada 5 tam sayı değerlerinden birini
girin: "))
if deger < 0:
     print("çok küçük")
else:
     if deger == 0:
        print("sıfır")
     else:
          if deger == 1:
             print("bir")
          else:
               if deger == 5:
                  print("beş")
               else:
                  print("çok büyük")
print("Tamamlandı")
```

Python'da çok yönlü koşullu durumlar iç içe if ifadeleri yanında if/elif/else ifadesi ile birlikte de kullanılabilir. Elif ifadesi, else if ifadesinin kısaltmasıdır. Örneğin,

```
# Klavyeden girilen 0-5 arasındaki sayıların yazı karşılığını veren
program
deger = int(input("Lütfen 0 - 5 arasında bir değer girin: "))
if deger < 0:
  print("çok küçük")
elif deger == 0:
  print("sıfır")
elif deger == 1:
  print("bir")
elif deger == 2:
  print("iki")
elif deger == 3:
  print("üç")
```

```
elif deger == 4:
  print("dört")
elif deger == 5:
  print("beş")
else:
  print("çok büyük")
print("Tamamlandı")
```

Örnek

```
if 9 > 5:
  print("Evet, 9 5'ten büyük")
if 9 != 5:
  print("Evet, 9 5'e eşit değildir")
# Başka bir örnek
if not (10 == 4) and 9 > 5:
  print("Tabii ki, çok basit bir karşılaştırma bu")
else:
  print(":(")
```

4.10. Çok Yönlü ve Zincirleme Durum İfadeleri

İkiden fazla olasılığın olduğu durumlarda ikiden fazla dallanmaya (yol) gereksinim duyarız. Bu tür durumlarda zincirleme koşul ifadeleri kullanılır. Her koşul sırasıyla sınanır. İlki yanlış ise sonraki kontrol edilir ve yazılan kodun tamamı bu şekilde çalıştırılır. Koşullardan biri doğru ise ilgili dal yürütülür ve cümlenin işlevi biter. Eğer birden fazla koşul doğru ise sadece ilk karşılaşılan doğru dal çalışır.

Çok Yönlü Koşullu Durum İfadesi

```
value = int (input("Lütfen 0...3 arasında bir tam sayı giriniz:"))
if value==0:          Kontrol 1
      print("sıfır")
elif value==1:        Kontrol 2
      print("bir")    Sonuç
elif value==2:
      print("iki")
elif value==3:
      print("üç")
else
      print("çok büyük")
print("Tamamlandı")
```

Diğer kontroller atlanır.

Zincirleme Durum İfadesi

```
value = int (input("Lütfen 0...3 arasında bir tam sayı giriniz:"))
if value==0:          Kontrol 1
      print("sıfır")
if value==1:          Kontrol 2
      print("bir")    Sonuç
if value==2:          Kontrol 3
      print("iki")
if value==3:          Kontrol 4
      print("üç")
if value>3:           Kontrol 5
      print("çok büyük")
print("Tamamlandı")
```

Yukarıdaki problemde kullanıcının klavyeden "1" girmesi durumunda işletilecek şartlar belirtilmiştir. Çok Yönlü Koşullu Durum yapısında Kontrol 1 kısmına girilecek fakat şart sağlanmadığı için sonuç kısmına girilmeyecek ve Kontrol 2 kısmına girilecektir. Geçilen bu aşamada şart sağlandığı için print() komutu işletilecek ve diğer tüm kontrol satırları atlanacaktır. Böylece fazla koşul kontrollerinden kaçınılmış olacaktır. Ancak Zincirleme Durum ifadesinde tüm şartlar sırasıyla kontrol edilecek, şartı sağlayanlar için print() satırları yapılacaktır.

4.11. Koşullu İfadeler

Koşullu ifadelerin genel yapısı aşağıdaki gibidir.

Koşul doğru ise koşullu ifadenin sonucu birinci durumdur. Koşul, "if" ifadesinde de görülebilen Boolean ifadesidir. Eğer koşul yanlış ise koşullu ifadenin sonucu ikinci durumdur.

```
n = int(input("Bir sayı giriniz: "))
print("|", n, "| = ", (-n if n < 0 else n), sep="")
```

Ekran Çıktısı

```
Bir sayı giriniz: -34
|-34| = 34
Bir sayı giriniz: 100
|100| = 100
```

Örnekte n değişkenine klavyeden giriş alınmış ve kullanıcı -34 değerini girmiştir. print() komutu ile ifade yazdırılırken koşullu ifade kullanılmış şart olarak n değişkeninin 0 'dan küçük durumu (if n < 0) kontrol edilmiştir. Şart doğru ise –n işlemi, yanlış ise n değeri yazdırılacaktır.

sep Parametresi : print() komutunda birden fazla değer yazdırılırken, yazdırılan ifadeler arasında istenilen bir karalter eklemek için sep parametresi kullanılır.

```
>>> print("T","C",sep=".")
T.C
```

4.11.1. Koşullu İfadelerde Hatalar

```
deger = int(input("Lütfen 0 - 5 arasında bir değer girin: "))
cevap="aralıkta değil" #Varsayılan Cevap
if deger == 0:
  cevap="Sıfır"
elif deger == 1:
  cevap="Bir"
elif deger == 2:
```

```
  cevap="iki"
elif deger == 3:
  cevap="üç"
elif deger == 4:
  cevap="dört"
elif deger == 5:
  cevap="beş"
print("Girdiğiniz sayı",cevap)
```

```
Lütfen 0 – 5 arasında bir değer girin: 2
Girdiğiniz sayı iki
>>>
Lütfen 0 – 5 arasında bir değer girin: 8
Girdiğiniz sayı aralıkta değil
```

Düşünelim/Deneyelim

Mantıksal operatörlerden and ve or operatörlerinin karışması durumu, en yaygın programlama hatasıdır. Programcılar, Python kaynak kodunu analiz etmek için Pylint'i (http://www.pylint.org/) kullanabilirler.

4.12. Mantık Karmaşası

Python, çok karmaşık durum/koşul ifadelerini oluşturmak için gerekli araçları sağlar. Ancak önemli olan, mantık karmaşasına yol açmadan kullanabilmektir. Boolean ifadeleri and ve not ile birlikte kullanılmak istendiğinde, karmaşık mantığa dayalı koşullar oluşturmamıza olanak sağlar. Örneğin aşağıda verilen 4 farklı Bollean ifade kodu çalıştırıldığı zaman aynı sonucu verecektir.

1. not (a == b and c != d)
2. not (a == b and not (c == d))
3. not (a == b) or not (c != d)
4. a != b or c == d

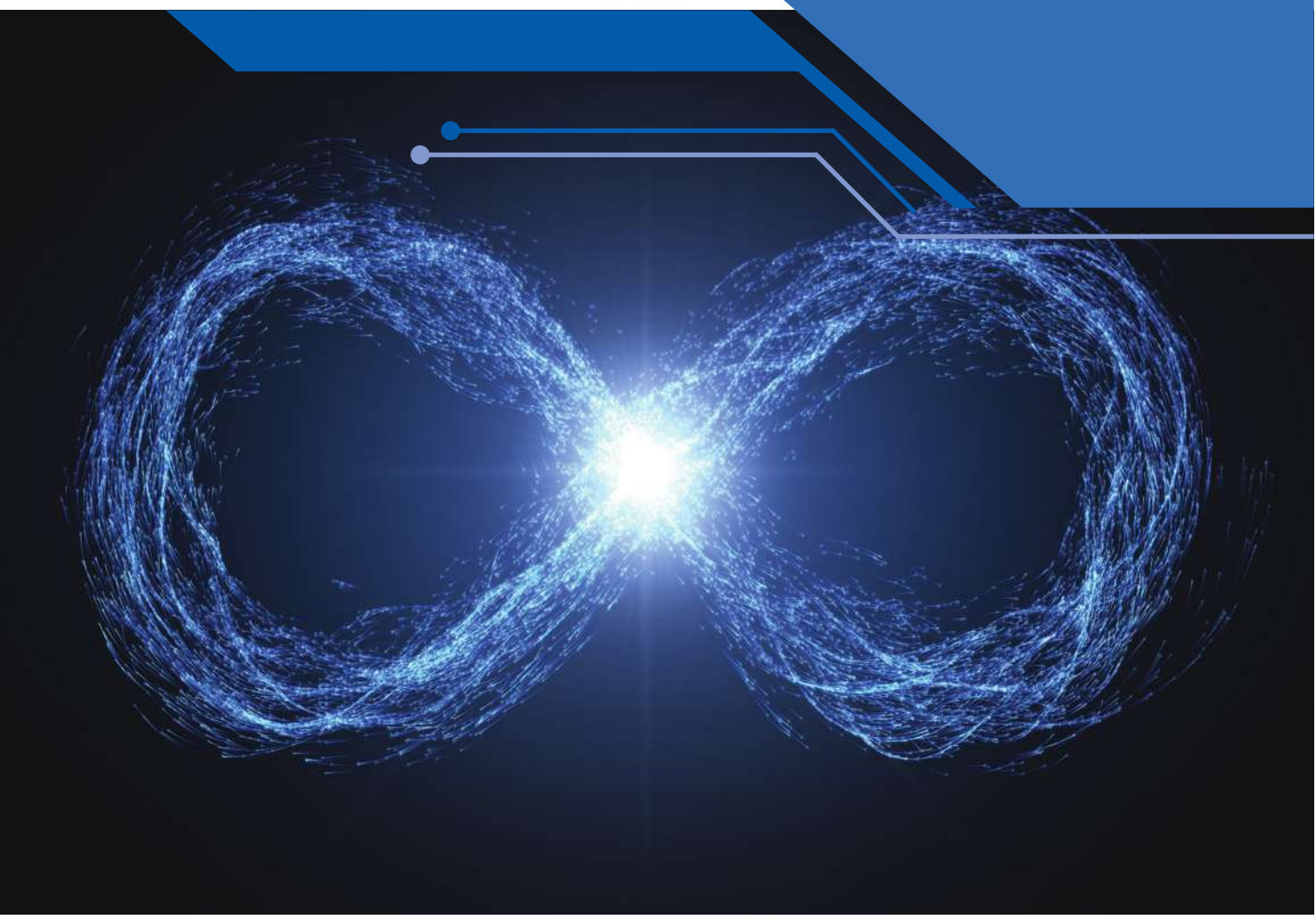
Ancak unutulmamalıdır ki;

- Çalıştırılırken en verimli yöntem basit düzeydeki mantıksal ifadelerdir.
- Basit düzeydeki mantıksal ifadeleri yazmak ve çalıştırmak daha kolaydır.
- Basit düzeydeki mantıksal ifadeler, çalıştırılırken de en verimli yöntemdir.
- Basit düzeydeki mantıksal ifadelerin değiştirilmesi, düzenlenmesi ve genişletilmesi de daha kolaydır.

Tartışalım

1. Bir Boolean ifadesi hangi değerleri alabilir?
2. Boolean deyimi nereden gelmektedir?
3. Python'da True deyimi hangi tam sayıya denk gelmektedir?
4. Python'da False deyimi hangi tam sayıya denk gelmektedir?
5. x, y, z = 3, 5, 7 olarak verilmektedir. Buna göre, aşağıda verilen işlemlerin sonucunu bulunuz.
   (a) x == 3
   (b) x < y
   (c) x >= y
   (d) x <= y
   (e) x != y - 2
   (f) x < 10
   (g) x >= 0 and x < 10
   (h) x < 0 and x < 10
   (i) x >= 0 and x < 2
   (j) x < 0 or x < 10
   (k) x > 0 or x < 10
   (l) x < 0 or x > 10

5. DÖNGÜLER



Bu bölümde;

- ✓ Python dilinde kullanılan döngü yapılarını kavrayacak,
- ✓ Döngü yapılarını ve kontrol yapılarını bir arada kullanabilecek,
- ✓ While ve for döngülerini içeren programlar geliştirebilecek,
- ✓ İç içe döngü yapılarını kullanan program yazabilecek,
- ✓ Döngü yapılarından çıkmak için kullanılan komutları kavrayacaksınız.

5.1. Döngü Yapıları

Döngüler, sıralı bir kod blokunun istenilen sayıda tekrarlanmasıdır. Döngü ve karar yapıları, algoritma oluşturma ve programlamada birçok problemin çözümünde kullanılır.

Aşağıdaki kod bloku 1'den 5'e kadar sayıları ekrana yazar.

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

Ekran Çıktısı

```
1
2
3
4
5
```

Ancak, 1'den 10.000'e kadar yazmak gerekirse böyle bir çözüm yolu doğru olmayacaktır!

Tekrar sayısı fazla olduğunda döngü yapıları tercih edilmelidir. Programlamada tekrar sayısının sayılması bir değişken yardımı ile elde edilebilir. Python dilinde döngü için while ve for döngü yapıları kullanılır.

5.2. For Döngüsü

For döngüleri belirli sayıda işlemlerin tekrarlanması için kullanılan döngülerdir. For döngüleri başlangıç ve bitiş değerleri arasında artım miktarına göre istenilen sayıda tekrar yapar.

Yukarıdaki örnek çıktı for döngüsüyle yapılmak istenseydi yazılması gereken kodlar şu şekilde olurdu:

```
for n in range(1,6)
     print(n)
```

Örnek Kullanım

```
for n in range(1, 11):
     print(n)
```

range (1,11) ifadesi, n değerinin hangi aralıkta çalışacağını gösterir. n'nin alacağı değerler: 1, 2 , 3 , 4 , 5, 6, 7, 8, 9, 10 yani $1 \leq n < 11$'dir. n'in ilk değeri 1 olup her çalışma anında sırayla artmaktadır.

5.2.1. For Döngüsü İçin Söz Dizimi

range (başlangıç değeri, son değer, arttırma/azaltma değeri) :

Başlangıç değeri: Döngü değişkeninin alacağı ilk değerdir. Eğer boş bırakılırsa 0 olarak belirlenir.

Son değer: Döngü değişkeninin bitiş değeridir. Boş bırakılmamalıdır.

Artırma/azaltma değeri: Döngü değişkeninin artırma veya azaltma miktarını belirler. Eğer boş bırakılırsa, 1 olarak belirlenir.

Başlangıç, bitiş, artırma ve azaltma değerlerinin hepsi tam sayı olmalıdır. Ondalıklı değerler veya diğer veri türleri kullanılmaz. Bunun dışında range ifadesi esnek kullanıma sahiptir:

```
for n in range(21, 0, -3):
      print(n, end=" ")
```

Ekran Çıktısı

```
21 18 15 12 9 6 3
```

end parametresi: print() içerisinde kullanılan end, bir parametre olarak görev yapar. İşlevi ise yazdırılmak istenen ifadelerin sonuna hangi karakterin geleceğini belirler. Varsayılan olarak "\n" karakteri ile birlikte gelir. Yani yazılan ifade bitince bir alt satıra geçer.

Başka bir örnek

```
top= 0
for i in range(1, 100): # burada döngü değişkeni olarak i kullanılmıştır.
      top+= i
print(top)
```

Ekran Çıktısı

```
4950
```

range(1000) denildiğinde 999'a kadar olan sayıların toplamı işlemini yapar ve ekrana 4950 yazar. Çünkü range() komutunda bitiş değeri döngüye dahil değildir.

5.2.2. For Döngüsü İçin Farklı Örnekler

Örnek

```
range(10) → 0,1,2,3,4,5,6,7,8,9
range(1, 10) → 1,2,3,4,5,6,7,8,9
range(1, 10, 2) → 1,3,5,7,9
range(10, 0, -1) → 10,9,8,7,6,5,4,3,2,1
range(10, 0, -2) → 10,8,6,4,2
```

```
range(2, 11, 2) → 2,4,6,8,10
range(-5, 5) → -5,-4,-3,-2,-1,0,1,2,3,4
range(1, 2) → 1
range(1, 1) → ()
range(1, -1) → ()
range(1, -1, -1) → 1,0
range(0) → ()
```

- range içerisinde 1 değer varsa bitiş değerini gösterir. 0'dan başlayıp birer artarak çalışır.
- range içerisinde 2 değer varsa başlangıç ve bitiş değerini simgeler ve birer artarak ilerler.
- 3 değer varsa başlangıç, bitiş ve artma miktarını ifade eder.

Örnek

10'un katlarını yazmak için aşağıdaki kod satırları kullanılabilir.

```
for i in range(16):
print("{0:3} {1:16}".format(i, 10**i))
```

Ekran Çıktısı

```
0                1
1               10
2              100
3             1000
4            10000
5           100000
6          1000000
7         10000000
8        100000000
9       1000000000
10     10000000000
11    100000000000
12   1000000000000
13  10000000000000
14 100000000000000
15 1000000000000000
```

format() metodu: print() komutunda çıktı verilirken yazdırılmak istenilen değerlerde hizalama, ekranda istenildiği yere yazdırma gibi biçimlendirme işlemlerinde kullanılan bir metotdur.

Örnek

```
 >>> print("{} {}"yi seviyor!".format("Ali", "Ayşe"))
"Ali Ayşe"yi seviyor!"
>>> print("{} {} yaşında bir {}dur".format("Ahmet", "18", "futbolcu"))
"Ahmet 18 yaşında bir futbolcudur"
```

Örnek

```
for i in range(10):
     print(i, end=" ")
     if i == 5:
          i = 20
     print("({})".format(i), end=" ")
print()
```

Ekran Çıktısı

```
0 (0) 1 (1) 2 (2) 3 (3) 4 (4) 5 (20) 6 (6) 7 (7) 8 (8) 9 (9)
```

Örnek

Aşağıdaki örnek girilen ifadenin harfleri üzerinde işlem yapar.

```
kelime= input("Bir kelime yaz: ")
for harf in kelime:
print (harf)
```

Ekran Çıktısı

```
Bir kelime yaz: Python
P
y
t
h
o
n
```

Örnek

Girilen kelimenin sesli harflerinin bulunmasını sağlayan program

```
kelime = input("Cümle Giriniz: ")
sesliHarfSayisi = 0
for c in kelime:
    if c == "A" or c == "a" or c == "E" or c == "e" \
    or c == "I" or c == "ı" or c == "İ" or c == "i" \
    or c == "O" or c == "o" or c == "Ö" or c == "ö" \
    or c == "U" or c == "u" or c == "Ü" or c == "ü":
        print(c, ",", sep=" ", end=" ")
        sesliHarfSayisi += 1
print(" (", sesliHarfSayisi, "sesli)", sep=" ")
```

Ekran Çıktısı

```
Cümle Giriniz: Bugün hava çok güzel
u , ü , a , a , o , ü , e ,  ( 7 sesli)
```

5.3. İç İçe Döngüler

İf ifadelerinde olduğu gibi while ve for blokları, başka döngü yapılarını içerebilir. İç içe döngülerin çalışma mantığını anlayabilmek için programın bir çarpım tablosu ürettiğini varsaymak gerekir. Benzer şekilde satır ve sütun değerleri olan bir tablo gibi de düşünülebilir.

Örnek

```
# Satır oluşturmak için
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))
for satir in range(1, sayi + 1):
     print("Satır #", satir)
```

Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10
Satır # 1
Satır # 2
Satır # 3
Satır # 4
Satır # 5
Satır # 6
Satır # 7
Satır # 8
Satır # 9
Satır # 10
```

Satırların yanına sütun eklemek istenirse kodlar aşağıdaki şekilde düzenlenmesi gerekir.

```
# Hem satır hem de sütun oluşturmak için :
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))
for satir in range(1, sayi + 1):
  for sutun in range(1, sayi + 1):
    deger = satir*sutun
    print(deger, end="  ")
  print()
```

Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10
1    2    3    4    5    6    7    8    9   10
2    4    6    8   10   12   14   16   18   20
3    6    9   12   15   18   21   24   27   30
4    8   12   16   20   24   28   32   36   40
5   10   15   20   25   30   35   40   45   50
6   12   18   24   30   36   42   48   54   60
7   14   21   28   35   42   49   56   63   70
8   16   24   32   40   48   56   64   72   80
9   18   27   36   45   54   63   72   81   90
10  20   30   40   50   60   70   80   90  100
>>>
```

Tablonun formatını hizalı yapmak için

```
# Hem satır hem de sütun oluşturmak için :
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))
for satir in range(1, sayi + 1):
  for sutun in range(1, sayi + 1):
    deger = satir*sutun
    print("{0:4}".format(deger), end="")
  print()
```

Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10
1    2    3    4    5    6    7    8    9   10
2    4    6    8   10   12   14   16   18   20
3    6    9   12   15   18   21   24   27   30
4    8   12   16   20   24   28   32   36   40
5   10   15   20   25   30   35   40   45   50
6   12   18   24   30   36   42   48   54   60
7   14   21   28   35   42   49   56   63   70
8   16   24   32   40   48   56   64   72   80
9   18   27   36   45   54   63   72   81   90
10  20   30   40   50   60   70   80   90  100
>>>
```

Örnek

```
sayi = int(input("Lütfen tablo ölçüsünü giriniz:"))
print(" ",end=" ")
for sutun in range(1, sayi + 1):
  print("{0:4}".format(sutun), end" ")
print()
print("+", end=" ")
for sutun in range(1, sayi + 1):
  print("----", end=" ")
print()
for satir in range(1, sayi + 1):
  print("{0:3}|".format(satir), end=" ")
  for sutun in range(1, sayi + 1):
    deger = satir*sutun # Çarpım sonucunun hesaplanması
    print("{0:4}".format(deger), end=" ")
  print()
```

Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10
     1    2    3    4    5    6    7    8 9 10
   +---------------------------------------
1  | 1    2    3    4    5    6    7    8    9  10
2  | 2    4    6    8   10   12   14   16   18  20
3  | 3    6    9   12   15   18   21   24   27  30
4  | 4    8   12   16   20   24   28   32   36  40
5  | 5   10   15   20   25   30   35   40   45  50
6  | 6   12   18   24   30   36   42   48   54  60
7  | 7   14   21   28   35   42   49   56   63  70
8  | 8   16   24   32   40   48   56   64   72  80
9  | 9   18   27   36   45   54   63   72   81  90
10 | 10  20   30   40   50   60   70   80   90 100
>>>
```

İç içe döngülerde önemli olan, döngü dışında kalacak kod satırlarının belirlenmesidir. Hangi satırların bir kere, hangi satırların döngü içerisinde olacağını belirlemek gerekir. İç içe döngüde satır (satir) dıştaki kontrol döngüsü olup sütun (sutun) döngüsü ise içteki döngüdür. İçteki döngü, dıştaki döngünün her tekrarında en baştan son bitiş değerine ulaşıncaya kadar kendini yineler.

5.4. İç İçe 3'lü Döngü

```
# ABC harflerinin farklı permütasyonu:
for ilk in "ABC":
for ikinci in "ABC":
    if ikinci != ilk:
        for ucuncu in "ABC":
          if ucuncu != ilk and ucuncu != ikinci:
            print(ilk + ikinci + ucuncu)
```

Ekran Çıktısı

```
ABC
ACB
BAC
BCA
CAB
CBA
```

5.5. While Döngüsü

```
sayac = 1 # Başlangıç değeri kontrol değişkenine atanır.
while sayac <= 5: #İstenilen değere ulaşıp ulaşmadığını kontrol
eder.
      print(sayac)      # Sayaç değerini ekrana yazar.
      sayac+= 1         # Sayaç değerini 1 arttırır.
```

While ifadesi, aşağıdaki ifadeleri 5 kere tekrar eder.

```
>>>print(sayac)
>>>sayac += 1
```

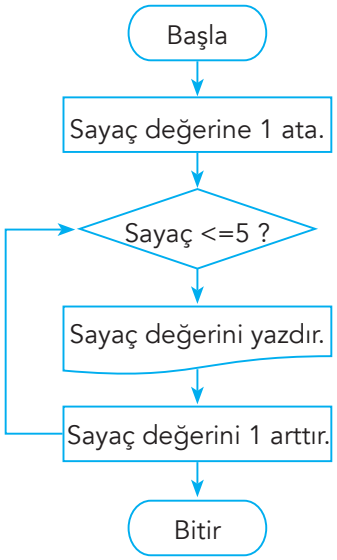
Sayac değişkeninin değerini sürekli olarak ekrana yazar. Yazma işlemi sonrasında değişkenin değerini 1 artırır. Bu işlemden sonra sayac değerinin 5'ten küçük veya eşit olması durumuna göre yazma işlemine devam eder. Şart sağlanmadığında ilgili kod blokunun tekrarlanması duracaktır.

5.5.1. While Döngüsü İçin Söz Dizimi

While ifadesi, ilgili kod satırlarının çalıştırılıp çalıştırılmayacağını belirler. Şart doğru olduğu sürece kod blokunu tekrar tekrar çalıştırır. Şart yanlış olduğunda ise döngü sonlanır. While için söz diziminde ilk önce while ifadesi yazılır. Şart ifadesi sonrasında : işareti ile yazılmalı ve bu şarta bağlı olarak çalışacak kod satırları alt alta yazılmalıdır.

```
while koşul :
    komutlar
```

5.5.2. While Döngüsü İçin Akış Şeması



While Döngüsü İçin Örnek Soru

Dışarıdan N/n karakteri girilinceye kadar döngünün kaç kere döndüğünü ekrana yazan while döngüsü

```
sayac = 0
giris= "Y"
while giris!= "N" and giris!= "n":
    print(sayac)
    giris= input("Devam etmek için " Y" - çıkmak için "N" giriniz: ")
    if giris == "Y" or giris == "y":
        sayac += 1
    elif giris != "N" and giris != "n":
        print("" + giris + "" geçerli bir giriş kodu değil")
```

Ekran çıktısı aşağıdaki gibi olur.

```
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
1
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
2
Devam etmek için " Y" - çıkmak için "N" giriniz: y
3
Devam etmek için " Y" - çıkmak için "N" giriniz: q
" q" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: r
"r" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: W
"W" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
4
Devam etmek için " Y" - çıkmak için "N" giriniz: y
5
Devam etmek için " Y" - çıkmak için "N" giriniz: n
```

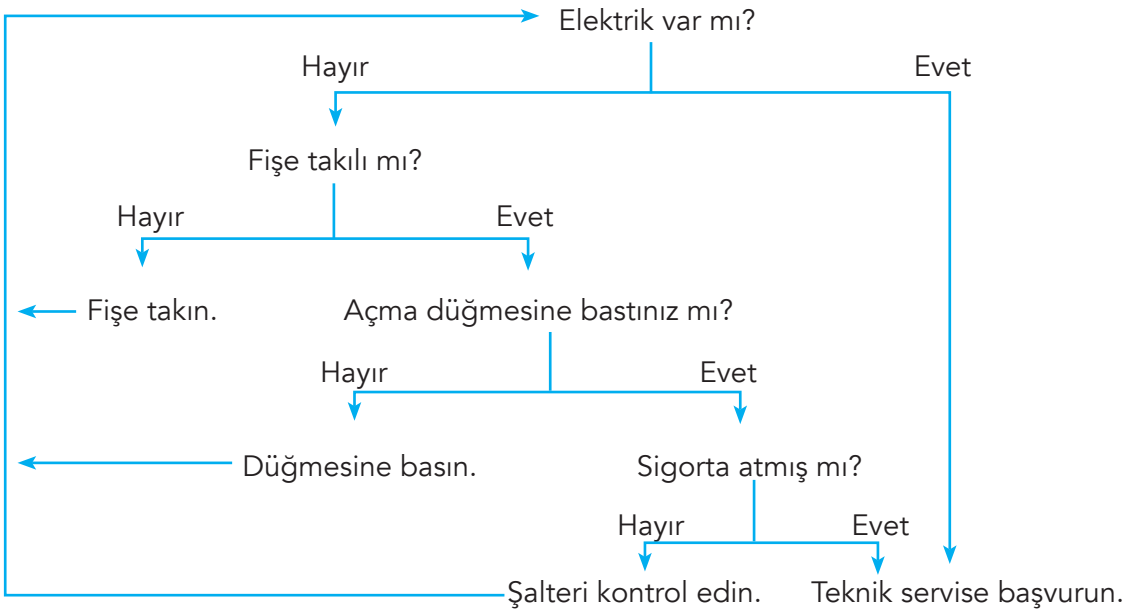
While Döngüsü İçin Örnek Soru 2

Dışarıdan negatif sayı girilinceye kadar sayıları toplayan while döngüsü

```
sayi = 0
toplam = 0
print("Bir sayı giriniz, negatif sayı döngüyü sonlandırır:")
while sayi >= 0:
      sayi= int(input())
      toplam += sayi
print("Toplam=", toplam)
```

While Döngüsü İçin Örnek Soru 3

Çalışmayan bir bilgisayar için sorun çözme adımlarını gösteren while döngüsü




```
# Bilgisayar Arıza Çözüm Programı
print("Yardım Edin! Bilgisayarım Çalışmıyor")
cozum = False
while not cozum:
  print("Bilgisayardan herhangi bir ses geliyor mu (fans vb) ")
  secim = input("Veya herhangi bir ışık yanıyor mu? (y/n):")
  if secim == "n":
    secim = input("Fişe takılı mı (y/n):")
    if secim == "n":
      print("Fişe takın")
    else:
      secim = input("Açma düğmesine bastınız mı (y/n):")
      if secim == "n":
        print("Açma düğmesine basın.")
      else:
        secim = input("Sigorta atmış mı? (y/n):")
        if secim == "n":
          secim = input("Şalter inmiş mi (y/n):")
          if secim == "n":
          print("Şalteri kontrol edin veya yenisi ile değiştirin. ")
          else:
```

```
            print("Teknik servise başvurun.")
            cozum = True
        else:
          print("Sigortayı kontrol edin. ")
  else:
    print("Teknik servise başvurun")
    cozum = True
```

Çözümün Açıklaması

While döngüsünü, mantıksal değeri olan değişken kontrol ediyor. Çözüm değişkeni yanlış olduğu sürece döngü, çalışmaya devam ediyor. Bu soruda kullanılan çözüm değişkeni, bayrak (flag) olarak kullanılmıştır. Bayrak aşağı indiğinde değer yanlış; yukarı kalktığında ise değer doğru olarak değişir. Bu örnek çözümde ise bayrak yukarı doğru kaldırılmış ve döngü sonlandırılmıştır. Ayrıca, bu soruda kullanılan "not cozum" ifadesi de önemlidir. Döngünün çalışması sorunun çözümüne bağlı olduğu için kontrol şartı, mantıksal değil olarak belirlenmiştir. Bu durum değişkenin değerini değiştirmeden kontrol edilmesini sağlayacaktır.

Python, tam sayı değeri olan 0 ve ondalıklı sayı olan 0,0 değerlerini yanlış (false); diğer tüm değerleri ise (pozitif ve negatif olanlar da dâhil) doğru (true) olarak kabul eder.

5.6. Belirli ve Belirsiz Döngüler

Döngünün tekrar sayısının bilindiği veya bilinmediği durumlar olabilir. Belirli döngülerde döngünün kaç defa döneceği, kaç kere çalışacağı kestirilebilir. Çünkü şart ifadesi bu konuda bilgi verir. Ancak bu durumun aksine kullanıcı girişine göre değişiklik gösteren ve farklı sayıda çalışan döngüler olabilir. Böyle durumlarda kullanılan döngülere de **belirsiz döngü** denir.

Belirli döngü örneği 1

```
n = 1
while n <= 10:
     print(n)
     n += 1
```

Belirli döngü örneği 2

```
n = 1
karar= int(input())
while n <= karar:
     print(n)
     n += 1
```

Belirsiz döngü örneği 3

```
karar = False
while not karar:
     giris = int(input())
     if giris == 999:
          karar = True
     else:
          print(giris)
```

5.7. Döngü'den Çıkma Komutları

While döngüsü, şart sağlandığı sürece ilgili kod satırlarını çalıştırır. Benzer şekilde for döngüsü de aralık içerisinde ilgili işlemleri gerçekleştirir. Ancak bazı değerler için döngü yapısından çıkmak, başka bir deyişle döngü işleyişinde değişiklik yapmak mümkündür. Bunun için **break** ve **continue** komutları kullanılabilir.

```
giris = 0
toplam = 0
print("Lütfen bir sayı giriniz, negatif sayılar döngüyü sonlandırır:")
while True:
giris = int(input())
     if giris < 0:
          break # Döngüden çıkılıyor
     toplam += giris
print("Toplam =", toplam)
```

Örnek

```
# Metin içerisindeki sesli harfleri bulma
kelime = input("Lütfen bir metin giriniz (Çıkış için X / x): ")
sesliHarfSayisi = 0

for c in kelime:
 if c == "A" or c == "a" or c == "E" or c == "e" \
or c == "I" or c == "ı" or c == "O" or c == "o"\
or c == "U" or c == "u" or c == "Ö" or c == "ö"\
or c == "Ü" or c == "ü" or c == "İ" or c=="i":
print(c, ",", end=" ", sep=" ")
sesliHarfSayisi += 1
elif c == "X" or c =="x":
break

print(" (", sesliHarfSayisi, " adet sesli harf)", sep="")
```

Ekran Çıktısı

```
Lütfen bir metin giriniz (Çıkış için X / x): AaEeIıİiOoÖöUuÜü
A, a, E, e, I, ı, İ, i, O, o, Ö, ö, U, u, Ü, ü, (16 adet sesli harf)
```

5.7.2. Continue İfadesi

Break ifadesi kullanıldığında ilgili kod satırları çalıştırılmadan atlanırken continue ifadesi kullanıldığında döngü başı yapılarak bir sonraki yeni değer için işlem yapılır.

Örnek

```
# 999 girilene kadar girilen pozitif sayıların toplamını alan program
# < girilen negatif sayılar işleme alınmayacaktır. >
toplam = 0
durum = False
while not durum:
  deger = int(input("Lütfen pozitif tam sayı giriniz (Çıkış için 999):"))
   if deger < 0:
    print("Negatif değer girildi, ", deger, "degeri işleme alınmadı")
     continue
   if deger != 999:
     print("Eklenen değer", deger)
     toplam += deger
   else:
     durum = (deger == 999)
print("Toplam =", toplam)
```

Ekran Çıktısı

```
Lütfen pozitif tam sayı giriniz (Çıkış için 999):1
Eklenen değer 1
Lütfen pozitif tam sayı giriniz (Çıkış için 999):2
Eklenen değer 2
Lütfen pozitif tam sayı giriniz (Çıkış için 999):3
Eklenen değer 3
Lütfen pozitif tam sayı giriniz (Çıkış için 999):6
```

Örnekte negatif değerde döngü başı yapılır.

5.7.3. While/else ve for/else

Python döngüler için opsiyonel else bloku kullanımını destekler. Break ifadesine rağmen döngünün terkedilmediği durumlarda döngüye ait else ifadesi kullanılabilir.

While / else

```
# Girilen 5 sayının ortalamasını alan program
# Negatif sayı girildiğinde program sonlandırılır
sayac = toplam = 0
print("Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz")
while sayac < 5:
      sayi = float(input("Sayı giriniz: "))
      if sayi < 0:
            print("Negatif sayılar kabul edilmemektedir. Çıkılıyor")
            break
      sayac += 1
      toplam += sayi
else:
      print("Ortalama =", toplam/sayac)
```

Ekran Çıktısı

```
Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz
Sayı giriniz: 6
Sayı giriniz: 7
Sayı giriniz: 8
Sayı giriniz: 6
Sayı giriniz: 5
Ortalama = 6.4
>>>
=============== RESTART:
Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz
Sayı giriniz: -1
Negatif sayılar kabul edilmemektedir. Çıkılıyor
```

Örnekte beş sayı girilene kadar while döngüsü dönecek şart yanlış olduğunda else satırına geçilerek ortalama değeri ekrana yazdırılacaktır.

5.8. Döngü Örnekleri

Faktöriyel Hesaplama

```
# Program girilen sayının faktöriyelini hesaplar
faktoriyel=1
sayac=1
sayi=int(input("Lütfen bir sayı giriniz.."))
while sayac<=sayi:
      faktoriyel*=sayac
      sayac+=1
print(sayi," sayısının foktöriyeli:",faktoriyel)
```

Ekran Çıktısı

```
Lütfen bir sayı giriniz..5
5 sayısının foktöriyeli: 120
```

Faktöriyel hesaplamak için klavyeden girilen sayıdan 1'e kadar döngü kurulmuş ve her döngü değeri çarpılma işlemine alınarak (faktöriyel*=sayac) amaca ulaşılmıştır.

Ağaç Çizimi

```
# Girilen değere göre "*" karakterinden ağaç çizen program
yukseklik = int(input("Çizilecek ağaçın yüksekliğini giriniz: "))
satir = 0
while satir < yukseklik:
      sayac = 0
      while sayac < yukseklik - satir:
            print(end=" ")
            sayac += 1
      sayac = 0
      while sayac < 2*satir + 1:
            print(end="*")
            sayac += 1
      print()
      satir += 1
```

Ekran Çıktısı

```
    Çizilecek ağacın yüksekliğini giriniz: 12

                  *
                 ***
                *****
               *******
              *********
             ***********
            *************
           ***************
          *****************
         *******************
        *********************
       ***********************
```

Asal Sayıları Listeleme

```
sonDeger = int(input("Kaça kadar asal sayıları görmek istersiniz? "))
sayi = 2
while sayi <= sonDeger:
      kontrol = True
      gecici = 2
      while gecici < sayi:
            if sayi % gecici == 0:
            kontrol = False
            break
            gecici += 1
      if kontrol:
            print(sayi, end= " ")
      sayi += 1
print()
```

Ekran Çıktısı

```
Kaça kadar asal sayıları görmek istersiniz? 77
2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73
```

İstenilen Sayıdaki Fibonacci Sayılarını Listeleyen Program

```
# İstenilen sayıdaki Fibonacci Sayılarını Listeleyen Program
deger1=0
deger2=1
sayac=2
sonDeger=int(input("Listelemek istediğiniz Fibonacci Sayıları adetini giriniz..:"))
print(str(deger1) + "  " +str(deger2),end="  ")
while sayac<=sonDeger:
      deger3=deger1+deger2
      print(deger3,end=" ")
      deger1=deger2
      deger2=deger3
      sayac+=1
```

Ekran Çıktısı

```
Listelemek istediğiniz Fibonacci Sayıları adedini giriniz..:15
0  1  1  2  3  5  8  13  21  34  55  89  144  233  377  610
```

Fibonacci Sayıları:

Her sayının kendisinden önce gelen iki sayının toplamı şeklinde yazılıp devam ettiği sayı dizisine Fibonacci Sayı Dizisi denir.

1,1,2,3,5,8,13,21,34,55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, …

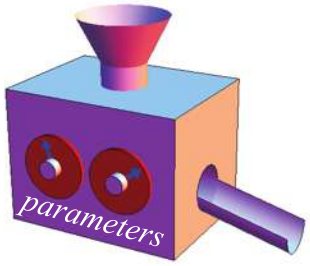
6. FONKSİYONLAR 1

Bu bölümde;

- ✓ Fonksiyon kullanmanın gerekçelerini öğrenebilecek,
- ✓ Python programlama dilinde kullanılan standart fonksiyonları kullanabilecek,
- ✓ Fonksiyon türleri konusunda bilgi sahibi olabileceksiniz.

6.1. Neden Fonksiyonlar?

Büyük ve kapsamlı bir program yazdığımızı ve program kapsamında pek çok kez aynı işlemi yapmamız gerektiğini düşünelim. Örneğin karekökü hesaplamak. Matematik bağlamında iki geometrik nokta (x1, y1) ve (x2, y2) arasındaki uzaklığı hesaplamamız gerekebilir. İkinci dereceden bir denklemin (ax2+bx+c = 0) çözüm kümesini bulmamız istenebilir. Elektrik mühendisliği ya da fizik bağlamında bir dizi değerin, sayıların karelerinin ortalamasının karekökünü bulmamız beklenebilir.



$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \sqrt{\frac{a_1^2 + a_2^2 + a_3^2 + \ldots + a_n^2}{n}}$$

Bu durumda her formül için karekök hesaplama yapan program satırlarını tekrar tekrar gereken yerlere kopyalamamız mı gerekir? Peki ya karekök bulma işlemi pek çok farklı program tarafından kullanılan bir hesaplama ise o zaman bütün programların içerisine yine bu satırları eklememiz mi gerekir? Acaba bu şekilde tekrarlayan işlemler için bu kodu paketleyip tekrar kullanmamızı sağlayan bir yöntem var mıdır?

6.2. Fonksiyon Nedir?

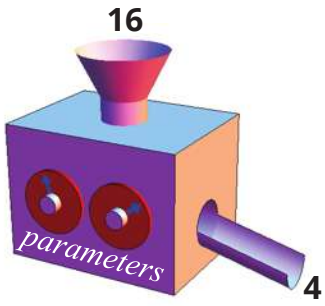
Bu kodu paketleyerek tekrar tekrar kullanmamızı sağlayan yaklaşımlardan biri "fonksiyonlar"dır. Bir fonksiyon, tekrar kullanılabilen kod parçacığıdır. Kendimiz fonksiyon yazabileceğimiz gibi önceden yazılmış ve kullanıma hazır fonksiyonları da kullanabiliriz. Diğer programlama dillerinde olduğu gibi Python kapsamında da standart fonksiyonların bulunduğu bir kütüphane vardır. Programcılar, "modül" olarak adlandırılan bu fonksiyonları kendi kodları içinden çağırarak kullanabilirler.

6.3. Fonksiyonlara Giriş

Aslında biz ilk bölümden itibaren fonksiyonları kullanmaya başlamıştık: print, input, int, float, str, ve type. Sıkça kullanılan işlemlerin çoğu için Python kütüphanesinde pek çok fonksiyon bulunmaktadır. Fonksiyon kavramını anlatmak için karekök bulma ve kare alma işlemlerini kullanalım. Fonksiyonlar, belirli bir işlemi gerçekleştiren kod blokudur. İlgili işlemi gerçekleştirmek için fonksiyon çağrılır. Python kütüphanesinde sqrt isimli bir fonksiyon karekök alma işlemini yapmaktadır. Bu fonksiyon tam sayı ya da reel sayı kabul etmektedir. Örneğin parametre olarak 16 gönderildiğinde 4 değeri geri dönmektedir.

$$\sqrt{16} = 4$$

Fonksiyonları kapalı bir kutu olarak düşünebiliriz. İçerisindeki kodlama detaylarını bilmemize gerek olmadan kolayca çağırıp işlem yaptırmak için kullanabiliriz. Dolayısıyla işlemi nasıl yaptığından daha çok ne yaptığını bilmemiz yeterli olur.



6.3.1. sqrt() Fonksiyonu

Burada sqrt (sayi) komutu ilgili "fonksiyonu çağırmak" için kullanılmaktadır. Önceden kullandığımız fonksiyonlar gibi sık kullanılan fonksiyonların küçük koleksiyonu kapsamı dışındadır. Bu fonksiyon standart kütüphane içerisinde ayrı bir modül olarak düşünülebilir. Bu nedenle "import" anahtar kelimesi kullanılarak ve "math" yani matematik kütüphanesinden çağırılarak kullanılır. Böylece sqrt() fonksiyonu programa tanıtılmış olur. "sayi" ise fonksiyona gönderilecek parametredir. Parametreler fonksiyona işlem yapması için ihtiyaç duyduğu değerleri göndermek ve bilgi alışverişini sağlamak için kullanılır.

```
from math import sqrt
# Kullanıcıdan değer alınıyor
sayi = float(input("Sayı Giriniz: "))
# Karakök hesaplanarak kok değişkenine aktarılıyor
kok = sqrt(sayi)
# Sonuçlar yazdırılıyor
print(sayi," sayısının karekökü" "=", kok)
```

sqrt() fonksiyonu sadece sayısal değerleri kabul eder. O yüzden farklı bir değer göndermek hata ile karşılaşılmasına neden olur. Örneğin sqrt("16") söz dizimi hatalı bir yapıdır.

6.3.1.1. sqrt() Fonksiyonunun Farklı Kullanımları

```
# Bu program sqrt() fonksiyonunun farklı kullanımlarını gösterir.
from math import sqrt
x = 16
# İstenilen sabit değerin karekökünün alınması
print(sqrt(16.0))
# Değişkenin karekökünün alınmasını sağlar
print(sqrt(x))
# sqrt() fonksiyonunun içerisinde işlem kullanımı
print(sqrt(2 * x - 5))
```

```
# İşlem sonucu geri dönen değerin değişkene aktarılması
y = sqrt(x)
print(y)
# İçerisinde işlem kullanılan sqrt() fonksiyonunun dönen değerinin
işleme tabi tutulması
y = 2 * sqrt(x + 16) - 4
print(y)
# İç içe sqrt() fonksiyonunun kullanılması
y = sqrt(sqrt(256.0))
print(y)
print(sqrt(int("45")))
```

Fonksiyonlar kendilerini çağırırken gönderilen parametreleri genellikle değiştirmez. Sonuçları fonksiyon adında ya da birden fazla parametre gönderildiği durumda sonucu içeren parametre ile iletir. Değer çağıran kişi tarafından bir değişkene atanmadığı sürece değişkenin değeri değişmez.

```
>>> from math import sqrt
>>> x = 2
>>> sqrt(x)
1.4142135623730951
>>> x
2
>>> x = sqrt(x)
>>> x
1.4142135623730951
```

6.3.2. Fonksiyonların Bölümleri

Çağıran kişi açısından fonksiyonun 3 önemli bölümü vardır:

İsmi: Her fonksiyonun, nasıl bir işlem yapılacağını ifade eden bir adı vardır. Değişkenleri isimlendirirken dikkat ettiğimiz kurallar, fonksiyon isimleri için de geçerlidir.

Parametreler: Bir fonksiyon belli sayıda parametre ile çağrılır ve her birinin doğru türde olması gerekir. Beklenenden daha az ya da çok sayıda parametre göndermek hataya neden olur.

Sonuç Türü: Fonksiyon kendini çağıran programa bir değer döndürür. Bu değer beklenen veri türü ile aynı olmalıdır.

```
>>> sqrt(10)
3.1622776601683795
>>> sqrt()
```

```
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
sqrt()
TypeError: sqrt() takes exactly one argument (0 given)
>>> sqrt(10, 20)
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
sqrt(10, 20)
TypeError: sqrt() takes exactly one argument (2 given)
>>> sqrt(16)
4.0
>>> sqrt("16")
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
sqrt("16")
TypeError: a float is required
>>> type(sqrt(16.0))
<class "float">
```

6.3.3. Parametresiz Fonksiyonlar

Bazı fonksiyonlar parametre kabul etmez. Örneğin rastgele bir sayı oluşturmamızı sağlayan random fonksiyonu bu duruma bir örnektir. Bu fonksiyonu çağırarak rastgele bir sayı değeri elde ederiz. Bu fonksiyonu parametre ile çağırmak hataya neden olacaktır.

```
>>> from random import random
>>> random()
0.9595266948278349
>>> random(20)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: random() takes no arguments (1 given)
```

6.3.4. Değer Döndürmeyen Fonksiyonlar

Bazı fonksiyonlarda parametre beklemelerine rağmen sonuç, değeri döndürmeyebilir. Örneğin print fonksiyonunun işlevi, hesaplama yapmak değil, ekranda görüntülemek olduğu için bu fonksiyonun sonucu diğer fonksiyonlara göre farklıdır.

```
>>> print(print(4))
4
None
```

Bu örnekte içerideki parametre ekrana 4 yazdırırken dıştaki print, içerideki fonksiyon değerini döndürür. Ayrıca bir değişkene "None" değeri atayabiliriz. Bu durumda hiçbir değer atanmamış demektir.

6.4. Fonksiyon ve Modüller

Bir Python modülü Python kodları içeren bir dosyadır. Dosyanın adı modülün adına işaret eder. Örneğin math.py isimli bir dosya standart matematik modülünde yer alan fonksiyonları içerir. Python standart kütüphanesinde 230 modül kapsamında yer alan binlerce fonksiyon vardır. Bu modüllerin geniş bir uygulama alanı bulunmaktadır. Örneğin built-ins (yerleşik işlevler) isimli modül (__builtins__), daha önce kullandığımız fonksiyonları kapsamaktadır: print, input vb. Bu yerleşik işlevler standart kütüphanenin çok küçük bir kısmını oluşturmaktadır. Geri kalan tüm diğer fonksiyonlara ulaşmak için programcıların program ya da yorumlayıcı içinden öncelikle "import" komutunu kullanarak ilgili kütüphaneye erişim sağlamaları gerekir.

from modül adı **import** fonksiyon adı

Bu komut ile ilgili kod parçacığı bilgisayarın sabit diskinde bir yerde saklanır. Böylece program, ihtiyaç duyduğunda bu kodları çalıştırmak için nereden çalıştıracağını bilir.

Python, bir modülden fonksiyon çağırmak için farklı yollar sunar. Bunlardan çok yaygın kullanılan ikisini inceleyelim.

```
from math import sqrt
```

Eğer birden fazla fonksiyon çağırmamız gerekirse örneğin yaygın logaritma ve trigonometri kapsamındaki cos fonksiyonuna da ihtiyacımız varsa söz dizimi şu şekilde olacaktır:

```
from math import sqrt, log10, cos
```

Böylece her 3 fonksiyon da program açısından erişilebilir ve kullanılabilir duruma gelir. "math" modülü pek çok farklı fonksiyonu da içermektedir. Eğer çok sayıda modül kullanmamız gerekiyorsa birkaç modülün spesifik olarak ismini belirtmek yerine

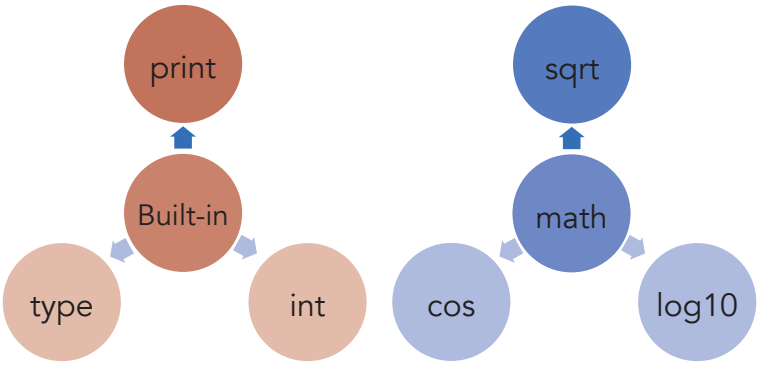
```
import math
```

ifadesini kullanarak "math" modülünün tamamına erişim sağlarız.

import modül adı

Böylece "math" modülündeki tüm fonksiyonlar erişilebilir hâle geldi. Ancak ilgili fonksiyonu çağırırken modülün adını da belirtmemiz gerekir. Modül adının "." ile fonksiyon adlarına birleştirilerek kullanıldığına dikkat ediniz. Bu yapıya birleşik (modül adı. fonksiyon adı) yapı diyoruz. Programcıların çoğu fonksiyon çağırma işleminde bu yapıyı tercih eder çünkü bu yapı programı daha basit ve anlaşılır kılmaktadır.

```
y = math.sqrt(x)
print(math.log10(100))
```



6.5. Yerleşik İşlevler

Bir süredir kullandığımız print, input, int, float, str, ve type isimli fonksiyonlar __builtins__ isimli modülde yer almaktadır. Bu modül Python için özel bir modüldür. Bu modüldeki fonksiyonlar import komutu ile erişim sağlanmadan doğrudan kullanılabilmektedir. Bu ismi kullanmamıza gerek olmamasına rağmen istersek kullanabiliriz.

```
>>> print("Merhaba")
Merhaba
>>> _ _ builtins _ _ .print("Merhaba")
Merhaba
>>> print
<built-in function print>
>>> _ _ builtins _ _ .print
<built-in function
>>> id(print)
9506056
>>> id( _ _ builtins _ _ .print)
9506056
```

Bu etkileşimli sıra print fonksiyonunun gerçekten yerleşik işlevler kütüphanesinde olup olmadığını kontrol etmemizi sağlar. Diğer bir fonksiyon ise "id" fonksiyonudur. id(x) ifadesi ile x isimli nesnenin hafızadaki adresini öğreniriz. id(print) ve id(__builtins__.print) aynı değer ile sonuçlandığı için aynı fonksiyonun nesnesini kullandığımızı anlarız. "dir" fonksiyonu ise bir modüldeki tüm fonksiyonları listelememizi sağlar. Yerleşik işlevlerin listesini görmek için dir(__builtins__) komutunu kullanırız.

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'Bytes Warning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'Depre cationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingP ointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
```

__builtins__ modulü bir de help (yardım) fonksiyonu içermektedir. Etkileşimli yorumlayıcıda spesifik fonksiyonlar hakkında anlaşılabilir bilginin sunulması sağlanır. Böylece her fonksiyona ilişkin ayrıntılı bilgi edinilebilir.

```
>>> help(input)
Help on built-in function input in module builtins:

input(...)
    input([prompt]) -> string

    Read a string from standard input.  The trailing newline is stripped.
    If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError.
    On Unix, GNU readline is used if enabled.  The prompt string, if given,
    is printed without a trailing newline before reading.

>>> help(sqrt)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> help(math.sqrt)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
>>> import math
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

6.6. Standart Matematik Fonksiyonları

math Modülü

Fonksiyon Adı	Açıklama	Örnek Kullanım ve Çıktı
sqrt()	İstenilen değerin karekök'ünün bulunmasını sağlar.	math.sqrt(16) → 4
exp()	e (Euler sabiti) sayısının istenilen kuvvetinin alınmasını sağlar.	math.exp(2) →7.389056
log()	log(x,y) fonksiyonumuz iki parametre alır. İlk parametremiz olan x logaritması, alınacak sayıyı; ikinci parametre olan y taban sayısını temsil etmektedir.	math.log(2,2) → 1.0
log10()	log(x,y) fonksiyonundan tek farkı taban olarak 10 sayısının sabit olmasıdır.	math.log10(10) → 1.0
cos()	cos(x), x derecesinin kosinüs değerini verir.	math.cos(45)→ 0.5253
pow()	pow(x,y) fonksiyonu x sayısının y. kuvvetinin alınmasını sağlar.	math.pow(2,2) → 4
degress()	degress(x) fonksiyonu x açısını radyandan dereceye çevirmeye yarar.	math.degress(45) → 2578.310078088
radians()	radians(x) fonksiyonu x açısını dereceden radyana çevirmeye yarar.	math.radians(45) → 0.7853981633
fabs()	fabs(x) fonksiyonu x değerinin mutlak değerinin alınması işlemini gerçekleştirir.	math.fabs(-5) → 5.0

```
'ImportWarning', 'IndentationError', 'IndexError', 'Interrup tedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryEr ror',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessL ookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEn codeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'Zero DivisionError',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float',
'format', 'frozenset', 'getattr ', 'globals', 'hasattr', 'hash', 'help',
'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit',
'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
'vars', 'zip']
>>>
```

Standart "math" modülü fonksiyonel bir hesap makinesi ile yapılabilen işlemlerin çoğunu içerir. Ayrıca pi (p) ve e (e) değerlerini de tanımlar. "math" modülünün içerdiği tüm fonksiyonları dir (math) komutu ile görebiliriz.

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'g amma', 'hypot',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>>
```

Çağıran tarafından gönderilen parametre gerçek parametre olarak bilinir. Fonksiyon tarafından tanımlanan parametre ise resmî parametredir. Bir fonksiyon çağrıldığında ilk gerçek parametre ilk resmî parametre değerine; ikinci gerçek parametre ikinci resmî parametre değerine atanır. Bu nedenle parametreleri doğru sırada göndermek çok önemlidir. Örneğin,

```
math.pow(8,2)
```

8 değerinin karesini alıp 64 sonucunu döndürürken

```
math.pow(2,8)
```

2 üzeri 8 değerini hesaplar ve 256 değerini döndürür.

Karekök fonksiyonunu asal sayıları belirlemek için de kullanabiliriz. Bütün olasılıkları (n ve n-1 için) denemek yerine n değerinin karekök sonucuna kadar olan değerleri denememiz yeterli olacaktır.

```
from math import sqrt
sonDeger = int(input("Hangi sayıya kadar asal sayılar listelensin? "))
deger = 2 # En küçük asal sayı
while deger <= sonDeger:
# İstenilen değere kadar dönmesi için döngü kuruluyor
   kontrol = True # Başlangıç aşamasında kontrol değişkeni True olarak belirlenir
   # 2 ile -1 arasındaki tüm değerlerin kontrolünün yapılması
   geciciDeger= 2
    kok = sqrt(deger) # Döngüde sırası gelen değerin karakökü hesaplanıyor
   while geciciDeger <= kok:
     if deger % geciciDeger == 0:
       kontrol = False # Asal sayı özelliği yitiriliyor ve kontrol False oluyor
       break # Kontrol döngüsünden çıkılıyor.
     geciciDeger += 1 # Bir sonraki kontrol sayısına geçiş
   if kontrol:
     print(deger, end= " ") # Şarta uyan değer Asal olarak kabul edilip yazdırılıyor.
   deger += 1 # Asal sayı kontrolü için sonraki sayı
print() # Kursor bir sonraki satıra alınıyor
```

Ekran Çıktısı

```
Hangi sayıya kadar asal sayılar listelensin? 51
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

6.7. time Fonksiyonları

Zamanla ilgili bilgi ve işlemlerin yer aldığı modül, "time" modülüdür. Bunlardan ikisi clock ve sleep fonksiyonlarıdır. Time.clock fonksiyonu ile programın belli bölümlerinin çalışma süresini ölçebiliriz. Programın ilk çağrıldığı andan itibaren geçen süreyi saniye olarak verir.

```
from time import clock
print("Adınızı Giriniz: ", end="")
baslangicZamani = clock()
```

```
ad = input()
zaman = clock() - baslangicZamani
print(ad, "bilgilerinizi", zaman, "zamanda girdiniz")
```

Ekran Çıktısı

```
Adınızı Giriniz: Oğuz
Oğuz bilgilerinizi 0.00411499999999998 zamanda girdiniz
```

Bir Python programının 1.000.000'a kadar bütün sayıların toplamını hesaplamasının ne kadar sürdüğünü öğrenmek için yine aynı fonksiyon kullanılır.

```
from time import clock
toplam = 0 # Toplam değişkeni tanımlanıp ilk değer olarak 0 veriliyor
basla = clock() # İşlem süresinin hesaplanması için süre başlatılıyor
for n in range(1, 1000001): # Toplamı alınacak sayılar için 1.000.000'e kadar döngü kuruluyor
  toplam += n
gecenZaman = clock() - basla # Geçen zaman hesaplanıyor
print("Toplam:", toplam, "Geçen Süre:", gecenZaman) # Sonuçlar yazdırılıyor
```

Ekran Çıktısı

```
Toplam: 500000500000 Geçen Süre: 0.16708000000000003
```

Bir Python programının 10.000'e kadar bütün asal sayıların toplamını hesaplamasının ne kadar sürdüğünü öğrenmek için yine aynı fonksiyon kullanılır.

```
# 10.000"e kadar olan asal sayıların adetini ve geçen zamanı bulan program
from time import clock
sonDeger = 10000
sayac = 0
zaman = clock() # Süre başlatılıyor
# En küçük asal sayı olan 2 den istenilen değere kadar döngü kuruluyor
for deger in range(2, sonDeger + 1):
# Sırayla sayılar ele alınıyor
```

```
  kontrol = True # Değerlerin kontrol edilmesi için ilk değer True veriliyor
# Asal olma özelliğinin kontrolü için bölenlerinin döngüsü kuruluyor
  for bolenSayi in range(2, deger):
    if deger % bolenSayi == 0:
      kontrol = False # Tam bölme işlemi oluştuysa kontrol False yapılıyor
      break # ve döngü sonlandırılıyor
  if kontrol:
    sayac += 1 # Asal olma özelliği sağlanmışsa sayac arttırılıyor
print() # Yeni satır başı
gecenZaman = clock() - zaman # İşlem tamamlandıktan sonra süre sonlandırılıyor
print("Adet:", sayac, " Geçen Zaman:", gecenZaman, " saniye")
```

Ekran Çıktısı

```
Adet: 1229 Geçen Zaman: 0.8711089999999999 saniye
```

Time.sleep() fonksiyonu ise programın çalışması sırasında belirtilen süre kadar durmasını sağlar. Örneğin geriye sayımda her sayıdan sonra 1 saniye beklemek için aşağıda görülen kod kullanılır.

```
from time import sleep
for sayac in range(10, -1, -1): # Range 10, 9, 8, ..., 0
print(sayac) # Sayac yazdırılıyor
sleep(1) # 1 saniye bekleme işlemi yapılıyor
```

6.8. Rastgele Sayılar

Rastgele sayılar; birçok programlama dilinde, oyun ve simülasyonlarda kullanılır. Bütün rastgele sayılar üreten algoritmalar, aslında gerçek rastgele sayılar üretmez. Sözde rastgele sayılar üreten bu algoritmalar uzun süre kullanımdan sonra aynı seriyi üretmeye başlar. Gerçek rastgele değerler, farklı sıralamada gelir ve bu sıralamayı tekrarlamaz. Python standart kütüphanesinde, Mersenne Twister algoritmasına dayalı olarak çalışan sözde rastgele değer üretmek mümkündür.

Mersenne Twister algoritmasına ilişkin daha fazla bilgi almak için https://docs.python.org/2/library/random.html adresini ziyaret ediniz.

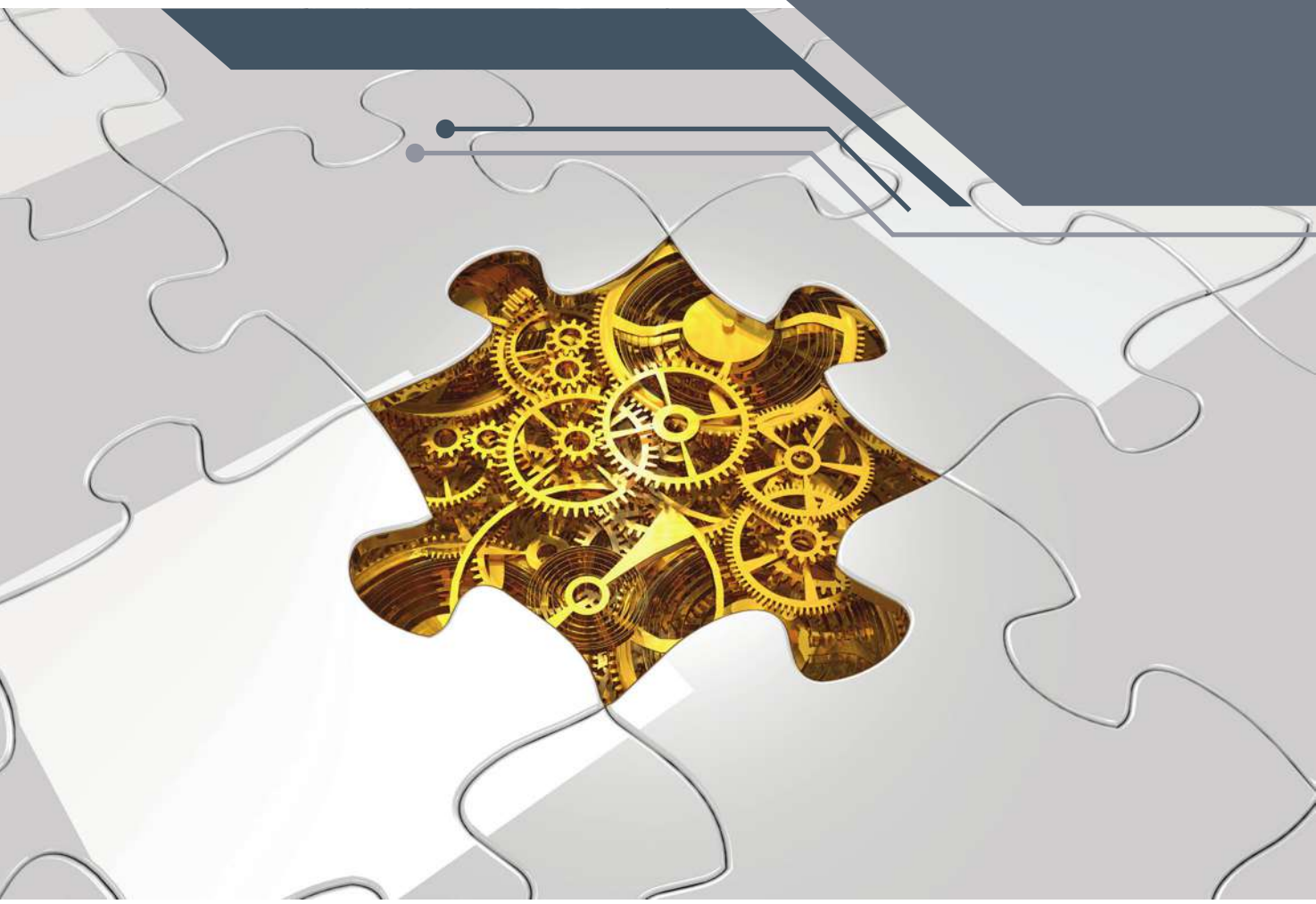
Aşağıdaki tabloda rastgele değer üreten fonksiyonlar ve özellikleri açıklanmıştır.

Rastgele sayı üretme fonksiyonları	
random	0<= x <1 arasında sözde rastgele ondalık bir değer döndürür.
randrange	Belirli bir aralıkta sözde rastgele tam sayı bir değer döndürür.
seed	Rastgele değer dizisi için bir değer alır.
choice	Değerler arasından rastgele bir değer seçer.

random.seed fonksiyonu, üretilecek sözde rastgele değerler için bir başlangıç değeri belirler. random.random veya random.randrange fonksiyonları her çağırıldığında bir sonraki sözde değerler için yeni bir değer döndürür. Aşağıdaki örnek, 1 ile 100 arasında rastgele değer döndürür.

```
from random import randrange, seed
for i in range(0, 100): # 100 adet rastgele sayı için döngü kuruluyor
   print(randrange(1, 1001), end=" ") # 1..1001 aralığında üretilen rastgele sayı yazdırılıyor
print()
```

7. FONKSİYON YAZMA



Bu bölümde;

- ✓ Python programlama dilinde programları alt programlar hâlinde yazmayı öğrenebilecek,
- ✓ Fonksiyon tanımla ve çağırma süreçlerini kavrayacak,
- ✓ Fonksiyonlara değer gönderme ve fonksiyonları döndürme süreçlerini uygulayabileceksiniz.

7.1. Fonksiyon Kavramı

Program yazarken kod satırları uzayabilir ve yazılan program karmaşık bir hâl alabilir. Bu durumu ortadan kaldırmak için problemi alt problemler hâlinde ele almak ve fonksiyon yazmak gerekir. Böylesine bir çözüm yolu birçok yarar sağlar:

- Programın yönetimi kolaylaşır.
- Daha doğru çözüm üretilebilir.
- Daha kolay hata ayıklama yapılabilir.
- Kod satırlarını değiştirmek/genişletmek kolaylaşır.

Python programlama dilinde bir fonksiyon için iki durum söz konusudur:

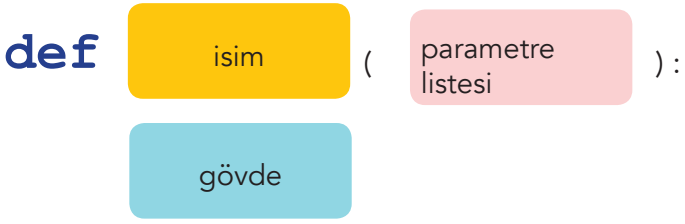
- Fonksiyon tanımlama: Fonksiyonun nasıl davranacağını tanımlayan kod satırları.
- Fonksiyon çağırma: Program içinde fonksiyonun çağırılması ile kod satırlarının çalışması.

Her fonksiyonun bir kez tanımlanması ancak farklı şekillerde çağırılması söz konusudur.

7.1.1. Fonksiyon Tanımlama

Fonksiyon tanımlamak için dikkat edilmesi gereken dört durum vardır:

- def: Bu ayrılmış sözcük ile fonksiyon tanımlama başlar.
- İsim: Fonksiyon için bir isim verilmelidir. Aynı değişken tanımlamada olduğu gibi.
- Parametre: Fonksiyon içinde kullanılan değerleri ifade eder.
- Gövde: Fonksiyon için gerekli olan kod blokundan oluşur.



7.1.2. Fonksiyon Yazma

Aşağıdaki örnekte def kelimesi fonksiyon tanımlama için kullanılmıştır. Fonksiyon ismi double olarak belirlenmiştir. Kullanıcıdan n ile bir değer istenmiştir. Fonksiyona ait kod bloku bir satırdan oluşmakta olup bu, girinti ile ötelenmiştir.

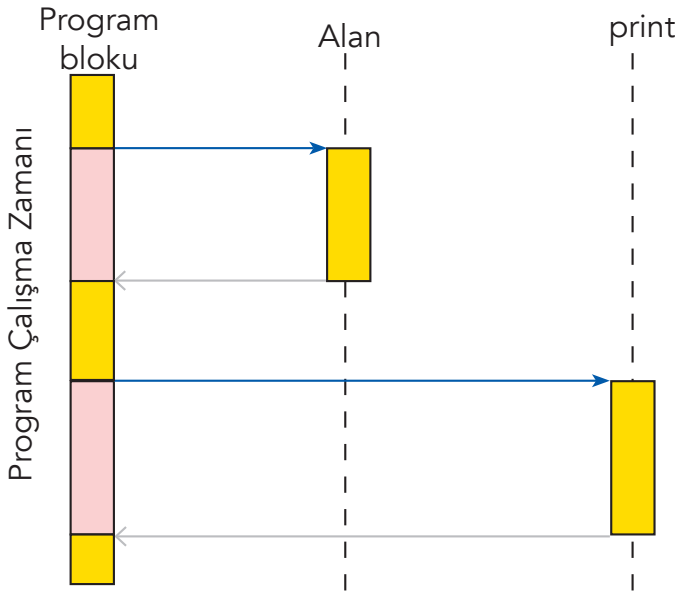
```
def double(n):
    return 2 * n
x = double(3)
print(x)
```

7.1.3. Fonksiyon Çağırma

Aşağıdaki örnekte fonksiyon, 5 değeri ile a değişkeni içerisine çağırılmıştır. return komutu ile gelen değer, print() komutu ile yazdırılmıştır.

```
# karenin alanını hesaplayan program
def Alan(a)
     return a*a
hesapla=Alan(5)
print(hesapla)
```

Fonksiyon şu şekilde çalışır



Program alan alt programına 5 değerini göndererek dallanma yapar. Alan alt programından hesapla değişkenine return komutu ile gelen değer aktarılır ve bu değer print komutu ile ekrana yazdırılır.

7.2. Fonksiyon Kullanımı Örnekleri

Örnek

```
def say():
     for i in range(1, 11):
          print(i, end=" ")
     print()
print("10"a kadar sayılıyor. . .")
say()
print("Tekrar 10"a kadar sayılıyor. . .")
say()
```

Ekran Çıktısı

```
10"a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
Tekrar 10"a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
```

Düşünelim/Deneyelim

Bir fonksiyon yazarak, program içinde kaç defa çağırılabileceğini deneyiniz.

Örnek

```
def say _ n (n):
     for i in range(1, n + 1):
          print(i, end=" ")
     print()
print("10'a kadar sayılıyor. . .")
say _ n(10)
print("5'e kadar sayılıyor . . .")
say _ n(5)
```

Ekran Çıktısı

```
10'a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
5'e kadar sayılıyor. . .
1 2 3 4 5
```

Düşünelim/Deneyelim

Sabit değerle çağırılan bir fonksiyon yazınız.

Örnek

```
def say _ n (n):
     for i in range(1, n + 1):
          print(i, end=" ")
     print()
for i in range(1, 10):
say _ n(i)
```

Ekran Çıktısı

```
1
1 2
1 2 3
1 2 3 4
```

```
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

Düşünelim/Deneyelim

Bir fonksiyon yazarak bir döngü içinde kaç defa çağırılabileceğini deneyiniz.

7.3. Değer Gönderme ile İlgili Olası Sorunlar

say_n()	Fonksiyon çağırıldığında eksik parametre hatası verir.
say_n(3, 5)	Fonksiyon çağırıldığında fazla parametre hatası verir.
say_n(3.2)	Tam sayı olmadığı için çalışma zamanı hatası verir.

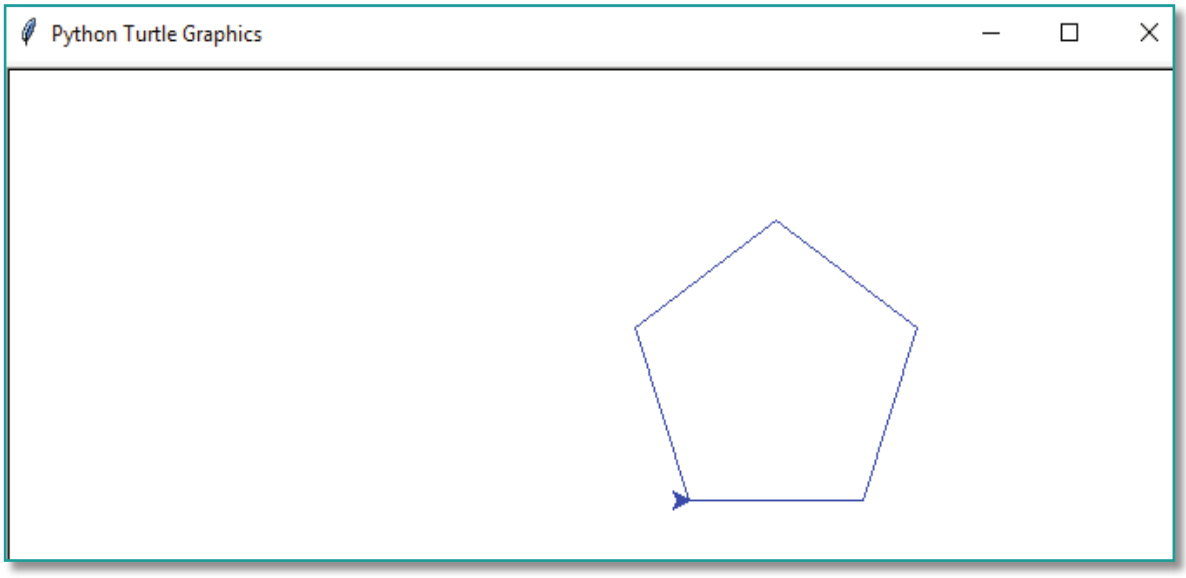
7.4. Çoklu Değer Gönderme Örnekleri

7.4.1. Grafik Ortamda Beşgen Çizimi

```
# turtle modülünü kullanarak beşgen çizimi
import turtle
import random
# turtle modülü programa ekleniyor
def polygon(sides,length,x,y,color):
#polygon adında ve içerisine 5 değer alabilen bir fonksiyon tanımlanıyor
  turtle.penup()
# Çizimi yapacak kalemin yönleri belirleniyor
  turtle.setposition(x,y)
# Parametre olarak gelen x,y değerleri başlangıç noktası olarak belirleniyor
  turtle.pendown()
# Çizimi yapacak kalemin yönleri belirleniyor
  turtle.color(color)
# Parametre olarak gelen renk atanıyor
```

```
  turtle.begin _ fill()
  for i in range(sides):
    turtle.forward(length)
    turtle.left(360//sides)
    turtle.end _ fill()
polygon(5,100,50,50,"blue")   # Tanımlanan fonksiyonun istenilen
parametrelerle çağırılması
```

Ekran Çıktısı



Poligon fonksiyonu, 4 parametre alıyor: kenar sayısı, her kenarın uzunluğu, x ve y ise poligonun koordinatları ve poligonun rengi.

7.4.2. En Büyük Ortak Çarpan Fonksiyonu

```
# Girilen 2 sayıdan en büyük ortak bölen program
sayi1=int(input("Lütfen ilk sayıyı giriniz: "))
sayi2=int(input("Lütfen ikinci sayıyı giriniz: "))
def gcd(s1,s2):
 min=s1 if s1<s2 else s2
  ebop=1
  for i in range(1,min+1):
    if s1%i==0 and s2%i==0:
      ebop=i # En büyük ortak bölen aktarılıyor
  return ebop
```

Ekran Çıktısı

```
Lütfen ilk sayıyı giriniz: 24
Lütfen ikinci sayıyı giriniz: 18
En büyük ortak çarpan : 6
```

7.5. Yerel Değişken

Yerel değişkenler fonksiyonların içinde tanımlanıp sonlandırılan değişken türüdür. Fonksiyon içerisine girildiğinde tanımlanıp hafızada yer ayırırlar ve fonksiyondan çıkıldığında hafızadan silinirler.

```
x=2
print("1. x =",x)
def fun1():
     x=10
     print("2. x =",x)
print("3. x =",x)
def fun2():
     x=20
     print("4. x =",x)
print("5. x =",x)
fun1()
fun2()
print("6. x =",x)
```

Ekran Çıktısı

```
1.x=2
3.x=2
5.x=2
2.x=10
4.x=20
6.x=2
```

7.6. Fonksiyon Yazarken Fonksiyon Sıralamasını Belirleme

Bir programın içerisinde fonksiyon tanımı, kullanımından önce ifade edilmelidir. Aksi takdirde program hata verecektir. Python yorumlayıcısı, bir kod blokunu satır satır çalıştırır. Fonksiyonu çalıştırmadan önce yukarıda tanımına rastlamaz ise program çalışmayacaktır.

7.6.1. Girilen İki Değerin En Büyük Ortak Böleni

```
def gcd(sayi1,sayi2):
  min=sayi1 if sayi1<sayi2 else sayi2
  ebop=1
  for i in range(1,min+1):
    if sayi1 % i== 0 and sayi2 % i== 0:
      ebop=i # En büyük ortak bölen aktarılıyor
  return ebop
def SayiGir():
  return int(input("Lütfen bir sayı giriniz : "))
def main():
  s1=SayiGir()
  s2=SayiGir()
  print("gcd(",s1, ",",s2, ") = ",gcd(s1,s2),sep="")
main()
```

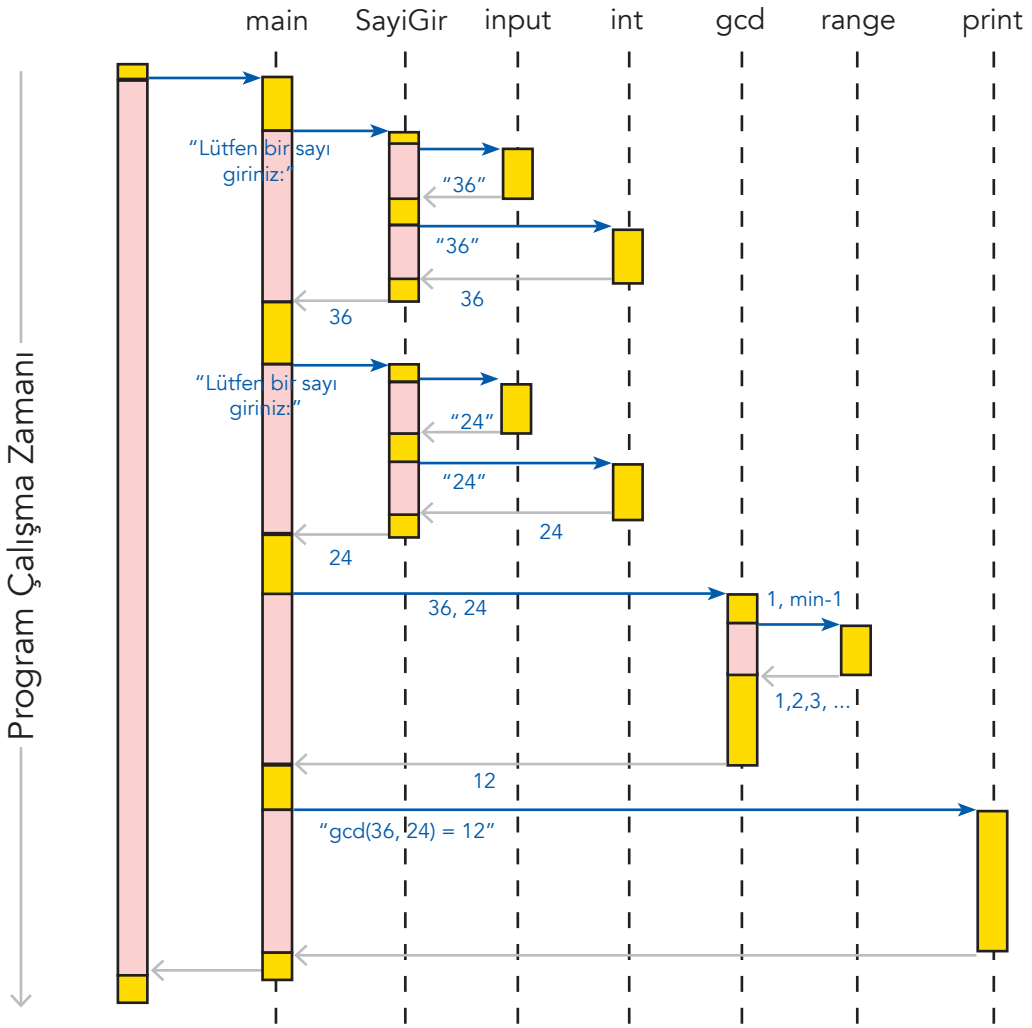
Ekran Çıktısı

```
Lütfen bir sayı giriniz: 36
Lütfen bir sayı giriniz: 24
gcd (56,32) = 12
```

Kodlarda main() fonksiyonunda EBOB'u bulunacak iki sayı kullanıcıdan istenmiş ve bu iki sayı gcd() fonksiyonuna gönderilmiştir. gcd() fonksiyonu içerisinde küçük olan sayıya kadar döngü kurulmuş ve döngü içerisinde iki sayıya aynı anda bölünen sayı EBOB olarak belirlenmiştir.

main () ifadesi, ana fonksiyonu ve diğer fonksiyonları çağırır.

Programın Çalışma Sırası



Program çalıştığında ilk olarak main() fonksiyonu çalıştırılır. SayıGir() fonksiyonuna iki kere dallanma yaparak s1 ve s2 değişkenlerine girilen sayıları aktarır (s1=36, s2=24). Üçüncü aşamada gcd fonksiyonuna s1 ve s2 değişkenleri parametre olarak gönderilir. for döngüsü içerisinde bu parametreler kullanılarak en büyük ortak bölen değeri bulunur ve return deyimi ile çağırıldığı yere gönderilir. print komutu ile gönderilen değer ekrana yazdırılır.

7.7. Parametre Gönderme

Bir fonksiyonun içerisinde parametre alacağı belirtilmişse mutlaka parametre gönderilmelidir.

```
def artir(x):
    print("Değişkenin artırılması yapılıyor, x =",x)
    x+=1
    print("Artırma sonucu değer, x =",x)
    return x
```

```
def main():
    x=5
    artir(x)
    print("Artırma sonrası, x =",x)
    print("Artırma sonrası, x =",artir())
main()
```

Ekran Çıktısı

```
Değişkenin artırılması yapılıyor, x = 5
Artırma sonucu değer, x = 6
Artırma sonrası, x = 5
Traceback (most recent call last):
  File "/Users/macbook/Desktop/Fwd _ _ Bilgisayar _ Bilimi _ Taslak _ Kitap/Pyton Kitap Kodlar/artirmaFonk.py", line 11, in <module>
    main()
  File "/Users/macbook/Desktop/Fwd _ _ Bilgisayar _ Bilimi _ Taslak _ Kitap/Pyton Kitap Kodlar/artirmaFonk.py", line 10, in main
    print("Artırma sonrası, x =",artir())
TypeError: artir() missing 1 required positional argument: "x"
>>>
```

Program başladığında artir() fonksiyonuna parametre olarak 5 değeri gönderilir ve işlem yaptırılır. İkinci kez artir() fonksiyonu çağırıldığında parametre verilmediğinden hata kodları alınır.

7.8. Fonksiyon Yazarken Tanımlayıcı Bilgileri Ekleme

Kod satırları için açıklamalar eklemek programı geliştirirken yarar sağlayacaktır.

- Fonksiyonun amacı
- Alınacak parametrenin görevi
- Geri dönüş değeri, açıklama satırlarında belirtilebilir.

Ayrıca, fonksiyon yazarı, değiştirilme tarihi varsa referanslar da eklenebilir.

```
# Yazar: Oğuz IŞIK
# Son düzenleme: 2017-01-06
# Yayınlanan bir örnekten uyarlanmıştır
def gcd (s1,s2)
# Girilen iki değer arasındaki EBOB'u bulur
```

7.9. Fonksiyon Örnekleri

7.9.1 Asal Sayıların Bulunması

```
from math import sqrt
def asal(n):
     kok=round(sqrt(n))+1

     for deneme in range(2,kok):
          if n % deneme==0:
               return False
          else:
               return True
def main():
  en _ buyuk=int(input("Asal sayıları hangi değere kadar
gösterelim? "))
  for deger in range(2,en _ buyuk+1):
    if asal(deger):
      print(deger,end=" ")
      print()
main()
```

Ekran Çıktısı

```
Asal sayıları hangi değere kadar gösterelim? 15
3
5
7
9
11
13
15
```

7.9.2 Hesaplama Yapan Fonksiyon

```
def yardim():
  print("Topla  : Girilen iki sayıyı toplar")
  print("Fark Al : Girilen iki sayının farkını alır")
  print("Yazdır : İşlem yapılan en son değeri ekrana yazdırır")
  print("Yardım : Bu ekranı görüntüler")
 print("Çıkış  : Programdan çıkışı sağlar")
def menu():
```

```
  return input("=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===")
def main():
  result=0.0
  done= False
  while not done:
    secim=menu()
    if secim== "T" or secim== "t":
     sayi1=float(input("Sayı 1: "))
      sayi2=float(input("Sayı 2: "))
      sonuc=sayi1+sayi2
      print(sonuc)
    elif secim== "F" or secim== "f":
      sayi1=float(input("Sayı 1: "))
      sayi2=float(input("Sayı 2: "))
      sonuc=sayi1-sayi2
      print(sonuc)
    elif secim== "Y" or secim== "y":
      print(sonuc)
    elif secim== "A" or secim== "a":
      yardim()
    elif secim== "Ç" or secim== "ç":
      done= True
main()
```

Ekran Çıktısı

```
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===T
Sayı Giriniz #1: 12
Sayı Giriniz #2: 13
25.0
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===F
Sayı Giriniz #1: 23
Sayı Giriniz #2: 21
2.0
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===Ç
```

Bu programda dört işlem harf seçimine göre yapılmaktadır. main() fonksiyonu çağırıldığında yapılacak işlem secim değişkenine aktarılarak kontrol edildikten sonra istenilen işlem yapılacaktır.

7.9.3 Kısıtlı Veri Girişi

```
def arasındami(ilk,son):
  if ilk>son:
    ilk,son=son,ilk
  deger=int(input("Lütfen belirtilen aralıkta bir değer girin " \
        +str(ilk)+ "..." +str(son)+ ": "))
  while deger<ilk or deger>son:
    print(deger, "Bu değerler arasında değil",ilk, "...",son)
    deger=int(input("Tekrar deneyin: "))
  return deger
def main():
  print(arasındami(10,20))
  print(arasındami(20,10))
  print(arasındami(5,5))
  print(arasındami(-100,100))
main()
```

Ekran Çıktısı

```
Lütfen belirtilen aralıkta bir değer girin 10...20: 4
4 Bu değerler arasında değil 10 ... 20
Tekrar deneyin: 21
21 Bu değerler arasında değil 10 ... 20
Tekrar deneyin: 16
16
Lütfen belirtilen aralıkta bir değer girin 10...20: 10
10
Lütfen belirtilen aralıkta bir değer girin 5...5: 4
4 Bu değerler arasında değil 5 ... 5
Tekrar deneyin: 6
6 Bu değerler arasında değil 5 ... 5
Tekrar deneyin: 5
5
Lütfen belirtilen aralıkta bir değer girin -100...100: -101
-101 Bu değerler arasında değil -100 ... 100
Tekrar deneyin: 101
101 Bu değerler arasında değil -100 ... 100
Tekrar deneyin: 0
0
```

Programda istenilen değerler arası veri girilmediğinde hata mesajı, girildiğinde onay mesajı verilmektedir.

7.9.4 Ağaç Çizimi

```
def tree(yukseklik):
  satir=0 # Ağacın çizilmesine başlanıyor
  while satir<yukseklik: # Girilen yükseklik değerine kadar döngü
kuruluyor
    sayac=0
    while sayac<yukseklik-satir:
      print(end=" ")
      sayac+=1
    sayac=0
    while sayac<2*satir+1:
      print(end="*")
      sayac+=1
    print()
    satir+=1
def main():
  yukseklik=int(input("Ağacın yüksekliğini giriniz: "))
  tree(yukseklik)
main()
```

Ekran Çıktısı

```
Ağacın yüksekliğini giriniz: 4
     *
    ***
   *****
  *******
```

Çizilmesi istenilen ağaç yüksekliğinin parametre olarak alan tree() fonksiyonu while döngüsüyle problemin çözümünü gerçekleştirir.

8. FONKSİYONLAR 2

Bu bölümde;

- ✓ Fonksiyonlarda kullanılan global ve yerel değişkenler hakkında bilgi sahibi olabilecek,
- ✓ Fonksiyonların tekrarlı kullanımlarını kavrayabileceksiniz.

8.1. Global Değişkenler

Fonksiyonların içerisinde tanımlanan değişkenler yerel değişkenlerdir. Bu değişkenlerin bazı özellikleri vardır:

- Bu değişkenler, hafızada aktif kullanıldıkları sürece yani sadece fonksiyon çalışırken korunur. Fonksiyondan çıkıldığında bu değişkenler de program tarafından unutulur ve hafızada ayrılan yerde başka değişkenler tarafından kullanılır.
- Aynı değişken adı çakışma olmaksızın farklı fonksiyonlarda kullanılabilir. Bir fonksiyon sonlanmadan diğeri çalışmaya başlayamayacağı için aynı değişken kullanılması sorun yaratmaz.
- Yerel değişkenler geçicidir; fonksiyonlar çağırıldığında kaybolur. Bazen bu çalışma sürecinden bağımsız bir değişkene ihtiyaç duyulur.
- Global değişken: Ne zaman hangi fonksiyon çağırılırsa çağırılsın, program tarafından tanınan ve hafızada sürekli yeri olan bir değişken.
- Bir değişken, bir nesneye atandığı zaman tanımlanır. Bir fonksiyona atanan değişken o fonksiyon için yereldir. Ancak global olarak tanımlanır ise programın tümü tarafından tanınır ve kullanılır.

8.2. Örnekler

8.2.1. Hesap Makinesi Örneği

```
# Kullanıcı seçimine göre Toplama ve Çıkarma işlemi yapan program kodları
def Yardim():
  print("Topla  : Girilen iki sayıyı toplar")
  print("Fark Al : Girilen iki sayının farkını alır")
  print("Yazdır : İşlem yapılan en son değeri ekrana yazdırır")
  print("Yardım : Bu ekranı görüntüler")
  print("Çıkış  : Programdan çıkışı sağlar")
def Menu():
  return input("=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===")
# Programda kullanılmak üzere global değişken tanımlanması
sonuc = 0.0
sayi1 = 0.0
sayi2 = 0.0
def SayiGir():
  global sayi1, sayi2 # sayi1 ve sayi2 nin global değişken olarak bildirilmesi
  sayi1 = float(input("Sayı Giriniz #1: "))
  sayi2 = float(input("Sayı Giriniz #2: "))
def Yazdir():
  print(sonuc)
def Topla():
```

```
  global sonuc
  sonuc = sayi1 + sayi2
def FarkAl():
  global sonuc
  sonuc = sayi1 - sayi2
def main():
  durum = False
  while not durum:
    secim = Menu() # İşlem yapılacak Menü Tasarımı
    if secim == "T" or secim == "t": # Toplama
      SayiGir()
      Topla()
      Yazdir()
    elif secim == "F" or secim == "f": # Çıkarma
      SayiGir()
      FarkAl()
      Yazdir()
    elif secim == "Y" or secim == "y": # Yazdırma
      Yazdir()
    elif secim == "A" or secim == "a": # Yardım
      Yardim()
    elif secim == "Ç" or secim == "ç": # Çıkış
      durum = True
main()
```

Ekran Çıktısı

```
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===T
Sayı Giriniz #1: 12
Sayı Giriniz #2: 43
55.0
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===F
Sayı Giriniz #1: 44
Sayı Giriniz #2: 32
12.0
=== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===Ç
```

8.2.2. Varsayılan (Default) Parametreler

Parametreli fonksiyonlar çağırılırken bir değer gönderilmesi gerekir. Bazı durumlarda bu değer gönderilmeden de fonksiyon çalıştırılmak istenirse, fonksiyonun tanımlama aşamasında gönderilmesi istenen parametreye varsayılan olarak bir değer verilmesi gerekir. Örneğin a=input() ya da a=input("Adınızı giriniz")

```
def gerisayim(n=5):
  for sayac in range(n, -1, -1):
    print(sayac)

gerisayim()
print()
gerisayim(8)
```

Ekran Çıktısı

```
5
4
3
2
1
0

8
7
6
5
4
3
2
1
0
```

Yukarıdaki programda gerisayim() fonksiyonu iki kere çağırılmıştır. İlk çağırımında parametre verilmediği için fonksiyonun tanımında verilen default değer ile işlem yapılmış ve 5'ten geriye doğru sayma işlemi gerçekleşmiş ve ekrana yazılmıştır. İkinci çağırılmasında ise 8 değeri parametre olarak gönderilmiş ve bu değer ile işlem yapılmıştır.

8.2.3. Varsayılan ve Diğer Parametreler

```
def AralikTopla(n, m=100): # Tek değer varsayılan atama
     toplam = 0
     for deger in range(n, m + 1):
          toplam += deger
```

Yukarıdaki AralikTopla() fonksiyonunun ikinci (m) parametresine değer girilmez ise varsayılan olarak 100 değeri alınır.

```
def AralikTopla(n=0, m=100): # İki değer varsayılan atama
     toplam = 0
     for deger in range(n, m + 1):
          toplam += deger
```

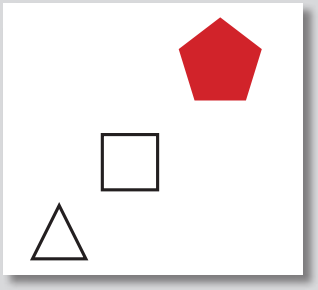
Yukarıdaki AralikTopla() fonksiyonunun birinci (n), ikinci (m) parametrelerine değer girilmez ise varsayılan olarak sırasıyla 0 ve 100 değerleri alınır.

```
def AralikTopla(n=0, m): # Tek değer varsayılan atama
     toplam = 0
     for deger in range(n, m + 1):
     toplam += deger
```

Yukarıdaki AralikTopla() fonksiyonunun birinci (n) parametresine değer girilmez ise varsayılan olarak 0 değeri alınır.

8.2.4. Gelişmiş Poligon Çizimi

```
import turtle
import random
# Program fonksiyona gönderilen parametreler ile çokgen çizer.
# Uzunluk paremetresi girilerek herbir kenarın uzunluğu belirlenir.
# Çizim x ve y parametrelerine girilen koordinat noktalarından başlar.
# Bir sonraki parametre çizimin kenar rengini belirler. (Varsayılan değer olarak siyah).
# Çizilen çokgenin içine dolgu olup olmayacağı belirlenir(Varsayılan False).
def Cokgen(kenarSayisi, uzunluk, x, y, renk="black", dolgu=False):
  turtle.penup()
  turtle.setposition(x, y)
  turtle.pendown()
  turtle.color(renk)
  if dolgu:
    turtle.begin _ fill()
  for i in range(kenarSayisi):
    turtle.forward(uzunluk)
    turtle.left(360//kenarSayisi)
  if dolgu:
    turtle.end _ fill()
```



```
# Adım adım çizim işlemi iptal edilerek çizim hızlandırılıyor
  turtle.hideturtle()
  turtle.tracer(0)
# Fonksiyonlar örnek çizimler için kullanılıyor
Cokgen(3, 30, 10, 10) # Üçgen çizimi
Cokgen(4, 30, 50, 50, "blue") # Kenar rengi mavi olan Kare çizimi
Cokgen(5, 30, 100, 100, "red", True) # Dolgusu kırmızı olan beşgen
çizimi
turtle.update()
turtle.exitonclick() # Fare tuşuna tıklandığında çıkış işlemi
yapılacaktır.
```

8.2.5. Öz Yineleme

Bir fonksiyonun kendisini çağırarak çözüme gitmesine özyineleme, böyle çalışan fonksiyonlara da özyinelemeli fonksiyonlar denilir. Özyinelemeli algoritmalarda, tekrarlar fonksiyonun kendi kendisini kopyalayarak çağırması ile elde edilir. Bu kopyalar işlerini bitirdikçe kaybolur. Bu yönteme en uygun örnek faktöriyel problemidir.

```
#Özyineleme ile faktöriyel hesaplama
def faktoriyel(n):
#Gelen n değerinin faktöriyeli alır.
  if n == 0:
    return 1
  else:
    return n * faktoriyel(n-1)
def main():
# Fonksiyonumuza çeşitli değerler ile test edelim
  print(" 0! = ", faktoriyel(0))
  print(" 1! = ", faktoriyel(1))
  print(" 6! = ", faktoriyel(6))
  print("10! = ", faktoriyel(10))
main()
```

Sonuç Ekranı

```
 0! = 1
 1! = 1
 6! = 720
10! = 3628800
```

Çalışma Prensibi

```
faktoriyel(6) =  *  faktoriyel(5)
              =  6  *  5  *  faktoriyel(4)
              =  6  *  5  *  4  *     faktoriyel(3)
              =  6  *  5  *  4  *  3  *     faktoriyel(2)
              =  6  *  5  *  4  *  3  *  2  *    faktoriyel(1)
              =  6  *  5  *  4  *  3  *  2  *  1  *    faktoriyel(0)
              =  6  *  5  *  4  *  3  *  2  *  1  *    1
              =  6  *  5  *  4  *  3  *  2  *  1
              =  6  *  5  *  4  *  3  *  2
              =  6  *  5  *  4  *  6
              =  6  *  5  *  24
              =  6  *  120
              =  720
```

Öz yinelemeli fonksiyonlar kendilerini çağırarak işlem yapan fonksiyonlardır. Yukarıdaki örnekte parametre olarak gelen değerden başlayarak birer azaltıp kendini çağırma işlemi yapar. Bu çağırma işlemi, döngü parametre değeri, 0 olasıya kadar devam eder. 0 olduğunda kendini çağırma işlemi durur ve geriye doğru değerler çarpılarak sonuç bulunur.

8.2.6. Öz Yineleme Olmadan Faktöriyel Hesaplama

```
def faktoriyel(n):
  sonuc = 1
  while n:
    sonuc *= n
    n -= 1
  return sonuc
def main():
  print(" 0! = ", faktoriyel(0))
  print(" 1! = ", faktoriyel(1))
  print(" 6! = ", faktoriyel(6))
  print("10! = ", faktoriyel(10))
main()
```

Sonuç Ekranı

```
0! = 1
1! = 1
6! = 720
10! = 3628800
```

8.2.7. Fibonacci Sayıları

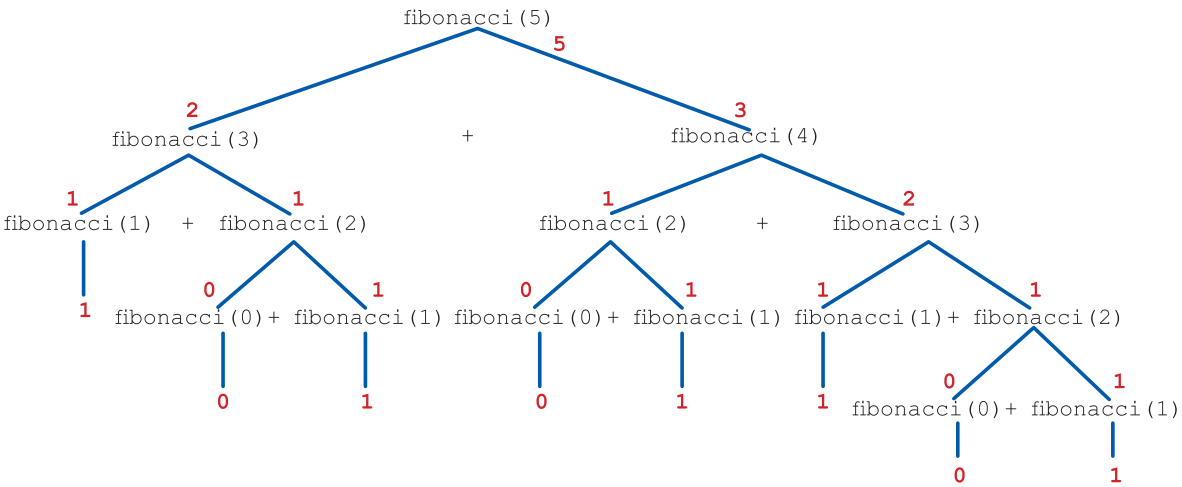
```
def fibonacci(n):
  if n <= 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fibonacci(n - 2) + fibonacci(n - 1)
sira=int(input("Görmek istediğiniz fibonacci sıra numarasını
giriniz.:"))
print(fibonacci(sira))
```

Sonuç Ekranı

```
Görmek istediğiniz fibonacci sıra numarasını giriniz.:22
17711
```

Fibonacci Sayıları: Her sayının kendisinden önce gelen iki sayının toplamı şeklinde yazılıp devam ettiği sayı dizisine Fibonacci Sayı Dizisi denir.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, …



8.3. Fonksiyonları Tekrar Kullanılabilir Yapma

Fonksiyonlar programlar içerisinde defalarca kullanılabilir. Aynı amaçlı fonksiyonların birden fazla program tarafından da kullanılması istenebilir. Bu durumda tanımlanan fonksiyon, amacına uygun çağrışım yapacak bir isim verildikten sonra kullanılmak istenen programlarda "from" komutu ile dosya adı yazılarak çağırılabilir.

Kullanımı

from <dosya adı> import <fonksiyon adı>

Örnek

```
# Asal sayının kontrol edildiği fonksiyon tanımlama
from math import sqrt

def AsalKontrol(n):
# Fonksiyona gelen değer asal ise geriye True, değilse False döner.
  bolen= 2
  kok = sqrt(n)
  while bolen <= kok:
    if n % bolen == 0: # Kalan kontrolü yapılıyor
      return False # Tam bölünme işlemi gerçekleşti. Asal Değil
    bolen += 1 # Bir sonraki bölen değerine geçiliyor.
  return True # Tüm değer kontrollerinden sonra kalanlı bölme
gerçekleşmediğinde, True değeri dönüyor.
```

Yukarıda yazılan kodlar, gönderilen sayının asal olup olmadığını kontrol eder. Gelen sayı asal ise geriye True, değilse False gönder. Kodlar yazıldıktan sonra dosya Kontrol.py adı ile kaydedilmelidir.

Aşağıdaki kod, kendisi ile aynı dizinde bulunan Kontrol.py dosyasında bulunan AsalKontrol fonksiyonunu kullanıyor.

```
from Kontrol import AsalKontrol
# Kontrol dosyasındaki AsalKontrol fonksiyonu programa ekleniyor
sayi = int(input("Bir sayı giriniz.: "))
if AsalKontrol(sayi):
  print(sayi, "ASAL")
else:
  print(sayi, "ASAL değil")
```

Diğer bir yol ise şu şekildedir:

```
import Kontrol
sayi = int(input("Bir sayı giriniz.: "))
if Kontrol.AsalKontrol(sayi):
     print(num, "sayısı ASAL sayıdır.")
else:
     print(num, "sayı ASAL değildir.")
```

9. NESNELER

Bu bölümde;

- ✓ Python dilinde kullanılan nesneler hakkında bilgi sahibi olacak,
- ✓ Nesneler içeren program yazabilecek,
- ✓ Dizi nesneleri kullanabilecek,
- ✓ Dosya nesneleri ile dosyalarda veriler üzerinde işlemler yapabilecek,
- ✓ Grafik nesneleri ile programlar geliştirebileceksiniz.

9.1. Nesne Kavramı

Nesne, kendine has özellikleri olan ve bu özellikler doğrultusunda bulunduğu duruma bağlı olarak çeşitli tutumlar sergileyen somut ya da soyut varlıklardır. Bu tanım, "somut" kısmıyla gerçek yaşamdaki nesneler için de geçerlidir. Donanım açısından baktığımızda kişisel bir bilgisayar; ana kart, işlemci, video kart, sabit disk ve kontrol ünitesi, bunları barındıran bir kasa, klavye, fare ve ekrandan oluşur. Video kart; çip, hafıza ve diğer elektronik bileşenleri barındıran karmaşık bir yapıdır. Somut kısmıyla nesnelere, donanım birimleri örnek olarak verilebilir.

Ancak gerçek yaşamda "nesne" olarak nitelendirmediğimiz fonksiyon, değişken, dizi gibi kavramlar programlama ortamında nesne olarak tanımlanabilirler. Bu da tanımın "soyut" diye ifade ettiği bölümü oluşturur.

Günümüzde yazılım geliştirme daha çok yazılımların donanım gibi kullanıldığı bileşenlere dayanmaktadır. Bir yazılım sistemi var olan yazılım oluşturma blokları üzerine inşa edilir. Python, farklı yapıdaki blokları ve bileşenleri desteklemektedir.

Python nesne yönelimli bir programlama dilidir. Nesne yönelimli programlama dili programcının nesneleri tanımlamasına, oluşturmasına ve yönetmesine olanak sağlar. Nesneler, veri ve fonksiyonları bir araya toplar. Diğer değişkenler gibi Python nesnelerinin de tipi ve sınıfı vardır.

9.2. Nesneleri Kullanmak

Bir nesne bir sınıfa örnek olarak verilebilir. Aslında başından beridir nesneleri kullanıyoruz ama detaylı bir biçimde şimdi inceleyeceğiz. Tam sayılar, reel sayılar, diziler ve fonksiyonlar Python için birer nesnedir. Fonksiyonların dışında genel olarak nesneleri pasif veri olarak kullandık. Bir değişkene tam sayı atayarak daha sonra o değişkenin değerini kullanabiliriz. "+" operatörü ile iki reel sayıyı ya da iki kelimeyi toplayabiliriz. Fonksiyonlara nesne yollayabilir ve sonucu nesne olarak alabiliriz.

Tipik bir nesne iki bölümden oluşur: veri ve metotlar. Örnek değişken ait olduğu nesne tarafından temsil edilen değişken anlamına gelir ve nesne de bir sınıf örneğidir. Örnek değişkenler için diğer isimler, özellik ve alanları da kapsamaktadır. Metotlar fonksiyon gibidir ve operatör olarak da ifade edilir. Bir nesne için örnek değişkenler ve metotlar nesnenin üyeleri olarak bilinir. "Nesneyi kullanan kod, nesne istemcisidir ve nesne istemcilere servis sunmaktadır." şeklinde açıklanır. Bir nesne tarafından sunulan servisler basit fonksiyonlara göre daha ayrıntılıdır. Bunun nedeni ise nesneler değişken içinde kolay veri korur çünkü nesnelerin örnek değişkenler içinde veri koruması daha kolaydır.

9.3. Dizi Nesneleri

Dizi nesnelerinin nasıl oluşturulacağını str örneği ile inceleyelim. Nesneler veri ve diziyi oluşturan veri diziyi oluşturan karakterlerin sıralaması hâline gelir.

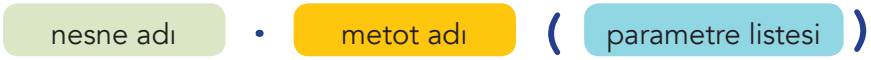
Şimdi str metotlarını inceleyelim.

```
ad = input("Adını yaz: ")
print("Merhaba " + ad.upper() + ", nasılsın?")
```

Bu kodlama ile kullanıcı tarafından girilen dizideki bütün karakterler büyük harfe çevirilir.

```
Adını yaz: Filiz
Merhaba FİLİZ, nasılsın?
```

print ifadesinde yer alan ad.upper() komutu ile bir metot çağırılmaktadır. Genel olarak bir metot çağırmanın biçimi şu şekildedir:



- Bir önceki örnekte "name" bir dizi nesnesini çağırmaktadır.
- "." (nokta) nesneye bağlı olarak çağırılacak metot ile bağ kurulduğunu ifade eder.
- "metot adı" çağırılan ve çalışacak olan metot adıdır.
- "parametre listesi" metoda gönderilen ve virgülle ayrılmış parametre listesidir. Bazı metotlar parametre istemez, o zaman bu liste boş olabilir.

Metoda gönderilen parametre listesi, fonksiyona gönderilen parametre listesi ile tamamen aynı biçimde davranır. Bu nedenle metotlar da geri değer döndürebilir.

Aşağıdaki örnekte rjust komutu metni sağa hizalamak için kullanılmaktadır.

```
kelime = "ABCD"
print(kelime.rjust(10, "*"))
print(kelime.rjust(3, "*"))
print(kelime.rjust(15, ">"))
print(kelime.rjust(10))******ABCD
```

Ekran Çıktısı

```
******ABCD
ABCD
>>>>>>>>>>ABCD
      ABCD
```

- kelime.rjust(10, "*") 10 karakterlik bir alan içinde boşluklara "*" değeri vererek "ABCD" dizisini sağa dayalı yazar.
- kelime.rjust(3, "*") komutu belirtilen değer dizi boyutundan daha küçük olduğu için bir işlem yapamaz.
- kelime.rjust(15, ">") 15 karakterlik bir alan içinde boşluklara ">" değeri vererek "ABCD" dizisini sağa dayalı yazar.
- kelime.rjust(10) ise boş alanlara basılacak varsayılan karakterin boşluk olduğunu belirtmektedir.

Aşağıdaki örnek, metodu dizi kullanarak çağırabildiğimizi göstermektedir:

```
>>> "aBcDeFgHiJ".upper()
"ABCDEFGHIJ"
>>> "PYTHON çok güzel bir dildir.".rjust(35,"-")
"-------PYTHON çok güzel bir dildir."
```

Bu söz dizimi daha önce kullandığımız bazı örneklerle benzerlik göstermektedir.

```
>>> "{0} {1}".format(23, 9)
```

```
"23 9"
>>> s = "{0} {1}"
>>> s.format(23, 9)
"23 9"
```

9.3.1 str Nesnesi İçin Yöntemler

Yöntem	Açıklama
upper	Metnin tüm harflerini büyük harfe dönüştürür.
lower	Metnin tüm harflerini küçük harfe dönüştürür.
rjust	Karakter dizisini sağa yaslar.
ljust	Karakter dizisini sola yaslar.
center	Karakter dizisini ortalar.
strip	Karakter dizisinin sağında ve solunda bulunan boşluk karakterlerini yok etmemizi sağlar.
startswith	Karakter dizilerinin ön ekini sorgulamamızı sağlar.
endswith	Karakter dizilerinin ön ekini sorgulamamızı sağlar.
count	Bir karakter dizisi içinde belli bir karakterin kaç kez geçtiğini sorgular.
find	Bir karakter dizisi içinde belli bir karakteri bulur.
format	Biçimlendirilmiş değerleri, dizi konum parametrelerine yerleştirir.

strip ve count Fonksiyon Örneği

```
# Ön ve arka plandaki boşlukları ve sayaç alt dizilerini ayırır
s = " ABCDEFGHBCDIJKLMNOPQRSBCDTUVWXYZ "
print("[", s, "]", sep="")
s = s.strip()
print("[", s, "]", sep="")
# Alt dizinin sayılarını sayar "BCD"
print(s.count("BCD"))
```

Ekran Çıktısı

```
[ ABCDEFGHBCDIJKLMNOPQRSBCDTUVWXYZ ]
[ABCDEFGHBCDIJKLMNOPQRSBCDTUVWXYZ]
3
```

9.3.2 getitem Kullanımı

"str" sınıfı __getitem__ isimli bir metot ile karakterin dizideki sırasını verir. Metodun isminin "__" ile başlıyor olması, bu metodun dâhilî kullanımı olduğu ve istemcilerin kullanamayacağı anlamına gelir. İstemciler, bu metodu özel bir söz dizimi ile kullanabilirler.

```
>>> s = "ABCEFGHI"
>>> s
"ABCEFGHI"
>>> s._ _getitem_ _(0)
"A"
>>> s._ _getitem_ _(1)
"B"
>>> s._ _getitem_ _(2)
"C"
>>> s[0]
"A"
>>> s[1]
"B"
>>> s[2]
"C"
```

Dizi nesnelerinde ilk karakterin konumu ya da indeks değeri "0" olduğundan her bir karakterin konumu, köşeli ayraç ([ ]) içerisinde gösterilir.

Dizilerle kullanılabilen başka bir metot ise "__len__" metodudur. Böylece dizideki karakter sayısı elde edilir.

```
>>> s
"ABCEFGHI"
>>> s = "ABCEFGHI"
>>> s
"ABCEFGHI"
>>> len(s)
8
>>> s._ _len_ _()
8
```

"len(s)" ve "s.__len__()" söz dizimleri fonksiyonel olarak aynıdır. İstemciler çağırırken global "len" fonksiyonunu kullanmalıdır.

Karakter Yazma Örneği

```
s = "Hayatta En Hakiki Mürşit İlimdir"
print(s)
for i in range(len(s)):
  print("[", s[i], "]", sep="", end="")
print() # Yeni satır başı
```

```
for ch in s:
  print("<", ch, ">", sep="", end="")
print() # Yeni satır başı
```

Ekran Çıktısı

```
Hayatta En Hakiki Mürşit İlimdir
[H][a][y][a][t][t][a][ ][E][n][ ][H][a][k][i][k][i][ ][M][ü][r][ş][i][t][ ][İ][l][i][m][d][i][r]
<H><a><y><a><t><t><a><        ><E><n><        ><H><a><k><i><k><i><><M><ü><r><ş><i><t>< ><İ><l><i><m><d><i><r>
```

Diziler değişmez nesnelerdir. Bu nedenle bir dizi nesnesinin içeriğini değiştiremeyiz.

```
s = "ABCDEFGHIJKLMN"
s[3] = "S" # Kurala aykırı, dizi sabit
```

Dizi değişmezliği, bir şeridin belirli bir diziyi değiştirmediği bir yöntem anlamına gelir.

```
s = " ABC "
s.strip() # s değişmez
print("<" + s + ">") # < ABC > yazılır , <ABC> değil
```

Ön ve arka plandaki boşlukları s değişkenine bağlı dizi kadar çıkarmak için yeniden atamanız gerekir:

```
s = " ABC "
s = s.strip() # Yeniden atama yapılır
print("<" + s + ">") # <ABC> yazılır
```

"strip" metodu yeni bir dizi döndürür ancak mevcut dizi değiştirilmez. Bir diziden boşlukları etkili bir biçimde temizleyebilmek için istemci, strip metoduna gönderilen değişkene tekrar atama yapmak zorundadır. Mantıksal olarak değerlendirildiğinde boş dizi ('') yanlış; tüm diğer diziler ise doğru olarak işlenir.

9.4. Dosya Nesneleri

Şu ana kadar çalıştırdığımız tüm programlar, sonlandıklarında tüm verilerini kaybetti. Oysaki bazı durumlarda bu verilerin saklanması gerekebilir. Örneğin kaydetmenize olanak sağlamayan bir kelime işlemci programı düşünün. Bu durumda tekrar erişip düzenleme, çıktı alma vb. hiçbir işlemi yapmak mümkün olmaz.

Sistemlerin çoğu saklanması gereken veriyi dosya biçiminde kaydeder ve her bir programın da dosya türünü belirten özel uzantısı vardır.

Python kapsamında veri saklama ve geri çağırma işlemlerini "file (dosya)" nesnesi ile gerçekleştiririz. "io" modülündeki "TextIOWrapper" ile bu işlemi yaparız. Dosya işlemleri yaygın kullanıldığı için "io" modülü dâhilinde kullanılan fonksiyon ve sınıflar, "import" komutu kullanılmadan çalıştırılabilir.

```
f = open("dosyam.txt", "r")
```

ifadesi bir dosya oluşturarak "f" isimli bir dosya nesnesi döndürür. İlk parametre dosya adını; ikinci parametre ise dosyanın durumunu ifade eder. Dosya durumu aşağıdaki gibi olabilir.

- 'r' yalnızca okunabilir.
- 'w' dosyayı yazmak için açar, yeni dosya oluşturur.
- 'a' dosyaya yeni veri eklenerek değiştirme yapılabilir.

"f" isimli bir dosya nesnesi oluşturmak ve "dosyam.txt" isimli dosya içeriğini okuyabilmek için

```
f = open("dosyam.txt", "r")
```

söz dizimi kullanılır. Eğer dosya yoksa ya da programı kullanan kişinin dosyaya erişim için gereken izinleri bulunmuyorsa bu komut hata verecektir.

"f" isimli bir dosya nesnesi oluşturmak ve "dosyam.txt" isimli dosyaya yazabilmek için

```
f = open("dosyam.txt", "w")
```

söz dizimi kullanılır. Dosya yoksa fonksiyon, disk üzerinde yeni bir dosya oluşturur. Aynı isimli bir dosya zaten varsa dosyadaki eski veriler yenileri ile değişecektir. Bu, dosyanın içerisinde önceden oluşturulmuş içeriğin silineceği anlamına gelir. "f" isimli bir dosya nesnesi oluşturmak ve "dosyam.txt" isimli dosyaya erişmek ve veri ekleyebilmek için

```
f = open("dosyam.txt", "a")
```

söz dizimi kullanılır. Dosya yoksa yeni bir dosya oluşturulur. Aynı isimli bir dosya varsa bu dosya tekrar düzenlemek için erişime açılır. Böylece dosyanın mevcut içeriği korunmuş olur. Bu fonksiyon ikinci parametre unutularak çağırılırsa varsayılan değer olarak "r" atanır.

```
f = open("dosyam.txt") ile f = open("dosyam.txt", "r")
```

aynı işlemi gerçekleştirir. "w" ya da "a" izni ile erişilmiş ve yazabileceğiniz bir dosya nesneniz varsa write metodunu kullanarak dosya üzerinde işlem yapabilirsiniz.

```
f.write("kaynak")
```

komutu 'kaynak' verisini dosya içerisine yazar. Aşağıdaki 3 komut

```
f.write("kaynak")
f.write("dosya")
f.write("veri")
```

dosyaya 'kaynakdosyaveri' verisini ekler. Veriyi ayırarak saklamak istiyorsak ona göre düzenlememiz gerekir.

```
f.write("kaynak\n")
f.write("dosya\n")
f.write("veri\n")
```

Bu işlem her kelimeyi ayrı bir satırda saklar. Böylece okuma işlemi yapmamız gerektiğinde işimiz kolaylaşmış olur. Dosyaya okuma izni ile erişilmişse

```
print(line.strip())
```

komutu, dosyadaki her bir satırı okur ve yazdırır. Dosya nesnesinin ok komutunu kullanarak bir dosyanın tüm içeriğini tek bir komutla bir dizi içerisine aktarabiliriz.

```
icerik = f.read()
in = open("bilgiTerimleri.txt", "r")
```

"open" metodu, bir dosyayı okuma ve yazma için açar; böylece programın dosya ile etkileşimi sağlanır. Dosya ile işi bittikten sonra programın dosyayı uygun biçimde kapatması gerekir. Daha önce kapatılmamış bir dosyaya tekrar erişim sırasında sorun yaşanabilir. Bu nedenle, açılan her dosya işlemler bittikten sonra mutlaka "close" metodu kullanılarak kapatılmalıdır.

9.4.1. Dosya Okuma ve Yazma İşlemleri

```
f = open("veriler.dat") # f adında dosya nesnesi
for line in f: # Her satırı metin olarak oku
print(line.strip()) # Sondaki yeni satır karakterini sil
f.close() # Dosyayı kapat
with open("veriler.dat") as f: # f adında dosya nesnesi
for line in f: # Her satırı metin olarak oku
print(line.strip()) # Sondaki yeni satır karakterini sil
f.close() # Dosyayı kapat
```

9.4.2. Dosya Okuma ve Yazma İşlemlerinde with/as kullanımı

"with/as" ifadesi ile nesnelere ilişkin işlemler yürütülür.

- "object-creation" ifadesi bir nesne oluşturur ve döndürür. Bu işlem başarısız olursa söz dizimi çalışmaya devam etmez.
- "as" ifadesi yaratılan nesne ile değişkenin bağlantısını kurar.
- "object" ile yaratılan nesne ilişkilendirilir.
- "block" ifadesi kapsamında farklı kodlar bulunur.

"with/as" ifadesi "TextIOWrapper" gibi sınıflarla çalışabilir; böylece başlama ve bitiş için belirli bir protokol sağlanmış olur. Sadece belirli sınıflar bu işlemi yürütmek için uygundur. Bu sınıfların ilk değer atama için "__enter__" ve sonlandırma için "__exit__" metotları vardır.

Sayıları Kaydetme Örneği

```
# Python "da dosyaya yazma ve dosyadan okuma programı
def Listeleme(dosyaAdi):
# Parametre olarak gelen dosyada bulunan kayıtları listeleme.
# Okumak için dosyanın açılması
  with open(dosyaAdi) as f: # f adında bir dosya nesnesi oluşturuldu
for satir in f: # Satır satır okuma işlemi için döngü kuruldu
     print(int(satir)) # int veri türüne dönüştürme ve ekrana yazdırma
def Kaydet(dosyaAdi):
```

```
# Parametre olarak gelen dosyada bulunan kayıtları kaydetme.
  with open(dosyaAdi, "w") as f: # f adında yazma modunda bir dosya nesnesi oluşturuldu
     sayi = 0
     while sayi != 999: # Kullanici 999 giresiye kadar dönen döngü kuruluyor
       sayi = int(input("Lütfen sayi giriniz..(Çıkış için 999):"))
       if sayi != 999:
          f.write(str(sayi) + "\n") # String veri türüne dönüşüm ve dosyaya kayıt
       else:
         break # Döngüden çıkış
def main():
# Ana Program başlangıcı. Menülü seçim işlemleri ve çıkış.
  kontrol = False
  while not kontrol:
    secim = input("K)aydet L)isteleme S)onlandır: ")
    if secim == "K" or secim == "k":
      Kaydet(input("Kayıt Edilecek Dosya Adı Girin:"))
    elif secim == "L" or secim == "l":
      Listeleme(input("Kayıtların Okunacağı Dosya Adını Girin<:"))
    elif secim == "S" or secim == "s":
      kontrol = True
main()
```

Ekran Çıktısı

```
K)aydet L)isteleme S)onlandır: k
Kayıt Edilecek Dosya Adı Girin:sayilar
Lütfen sayi giriniz..(Çıkış için 999):1
Lütfen sayi giriniz..(Çıkış için 999):54
Lütfen sayi giriniz..(Çıkış için 999):66
Lütfen sayi giriniz..(Çıkış için 999):44
Lütfen sayi giriniz..(Çıkış için 999):23
Lütfen sayi giriniz..(Çıkış için 999):999
K)aydet L)isteleme S)onlandır: l
Kayıtların Okunacağı Dosya Adını Girin<:sayilar
1
54
66
44
23
K)aydet L)isteleme S)onlandır: s
```

"io" modülünden seçilen Python dosya sınıfını "TextIOWrapper" olarak görüyoruz. Bu sınıfta işlem gören dosyalar metin türündedir. Metin dosyaları, karakter veri saklar ve basit bir editörle kolayca oluşturulup düzenlenebilir. Python kapsamında dizi ve dosya nesnelerini bir arada kullanarak güçlü dosya işleme programları yazabiliriz. Bir dosyayı açıp, içeriğini okuyup tamamen değiştirip başka bir dosyaya yazabiliriz.

```
from convertupper import capitalize
capitalize("declaration.txt")
```

9.4.3. TextIOWrapper Yöntemleri

open	Metin dosyasını açma komutu
read	Metin dosyasına bir dizi okuyan komut
write	Metin dosyasına bir dizi yazan komut
close	Dosyayı kapatma komutu. Dosyaya yazarken close yöntemi kullanılırsa dosyaya gönderilen tüm verilerin dosyaya kaydedilmesi sağlanır.

Örnek

Nesneler metotların yanı sıra veri de içerir. "TextIOWrapper" nesneleri tam sayı, dizi ve mantıksal ifadeler saklayabilir.

```
>>> f = open("temp.dat", "w")
>>> f.name
"temp.dat"
>>> f._CHUNK_SIZE
8192
>>> f.mode
"w"
>>> f.encoding
"cp1252"
>>> f.line_buffering
False
```

name", "_CHUNK_SIZE", "encoding" ve "line_buffering" ifadeleri "f" nesnesinin örnek değişkenleridir. Bu değişkenlerin önceden kullandıklarımızdan farkı "." ile belirli bir nesne ile ilişkilendirilmiş olmalarıdır. Bu isimler metot değil veriyi ifade ettiği için sonunda ayraç kullanılmamaktadır. "f" ve "g" isimli iki farklı nesnemiz varsa bu nesneler birbirinden farklı davranabilir.

```
x = 2
```

ifadesinde x değişkenine "2" değeri atanırken

```
obj.x = 2
```

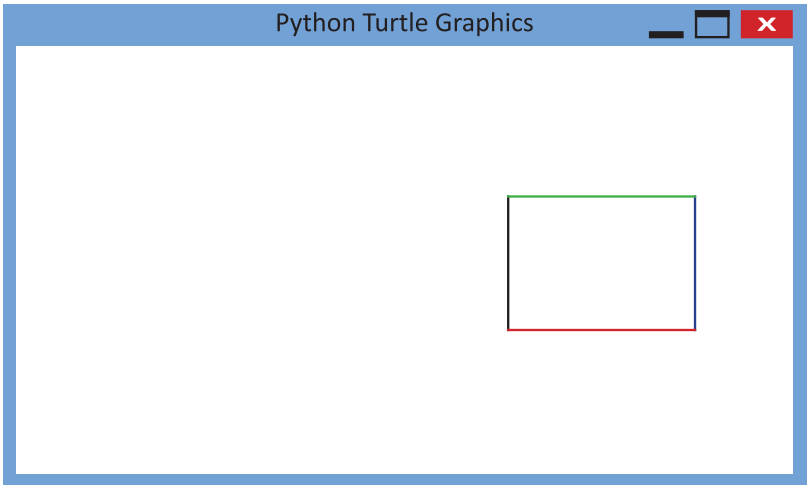
ifadesinde "obj" isimli nesnenin örnek x değişkenine atama yapılmaktadır.

9.5. Turtle Grafik Nesneleri

Grafik çizme işleminde "Turtle" nesnesinin, çizim işlemi için kalemi modellediğini biliyoruz.

Kendi "Turtle" nesnelerimizi tasarlayarak kullanabiliriz. Birden fazla kalem kullanmamız gerekirse bu, çok kullanışlı olacaktır.

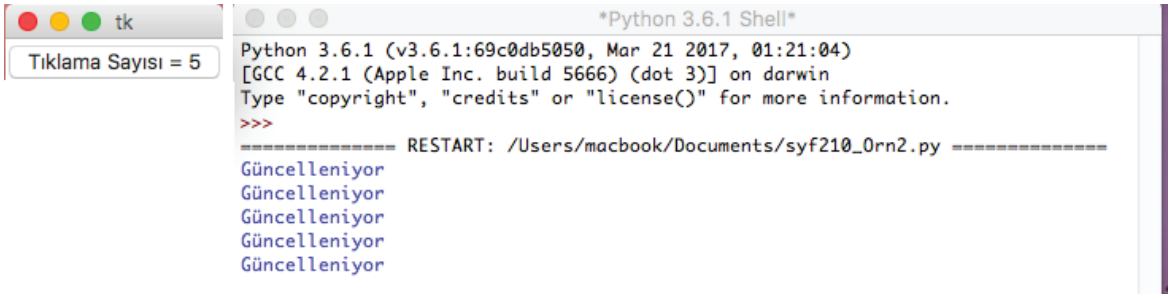
```
# Pencere içerisine diktörtgen bir kutu çizimi
from turtle import Turtle, mainloop
t = Turtle() # Adı t olan yeni bir Turtle() nesnesi oluşturuluyor
t.pencolor("red") # t nesnesinin çizim kalemi rengi kırmızı olarak ayarlanıyor
t.forward(200) # t nesnesinin kalemi 200 birim ileri oynatılıyor ve dikdörtgenin alt kenarı çiziliyor
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("blue") # t nesnesinin rengi mavi olarak değiştiriliyor
t.forward(150) # Sağ kenarın çizilmesi için Turtle 150 birim yukarı ilerletiliyor
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("green") # Turtle nesnesinin rengi yeşil olarak değiştiriliyor
t.forward(200) # Dikdörtgenin üst kenarının çizilmesi için Turtle 200 birim ilerletiliyor.
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("black") # Turtle nesnesinin rengi siyah olarak değiştiriliyor
t.forward(150) # Son kenar olan sol kenarın çizilmesi için Turtle 150 birim ilerletiliyor.
t.hideturtle() # Turtle gizleniyor
mainloop() # Kullanıcıdan veri bekleniyor
```



9.5.1. Turtle Grafik & tkinter Nesneleri

"tkinter" modülü Tk araç kiti ile grafiksel kullanıcı arayüzleri oluşturma sürecinde farklı sınıflar sunar. Tk MicrosoftWindows, Apple Mac, and Linux işletim sistemleri ile kullanılabilir. "tkinter" modülü "Turtle" modülünden daha kapsamlı ve karmaşıktır. Aslında "Turtle" modülü "tkinter" modülünün sunduğu bileşenler üzerine inşa edilmiştir.

```
from tkinter import Tk, Button
sayac = 0 # Tıklama sayısının hafızada tutulacağı değişken tanımlanıyor
def update():
# Grafikte bulunan butona tıklandığında sayaç artırma işlemi
    global sayac, b
    sayac += 1
    b.config(text="Tıklama Sayısı = " + str(sayac))
    print("Güncelleniyor")
root = Tk()
b = Button(root)
b.configure(background="yellow", text="Tıklama Sayısı = 0", command=update) # Ekrana buton nesnesi oluşturuluyor
b.pack()
root.mainloop()
```



9.5.2. Buton Test Etme

Tk: Bu sınıf bir grafik pencereyi temsil eder.

```
root = Tk()
```

ifadesi, "root" isimli bir nesne oluşturur. Bu nesne, uygulamanın ana grafik penceresini ifade eder.

```
root.mainloop()
```

ifadesi, pencerenin yanı sıra grafik programını başlatmak için "mainloop" metodunu çağırır. Bu metot, hareket sürecini başlatarak kullanıcının görsel dönüt almasına olanak sağlar.

Button: Kullanıcının basabileceği bir grafik butonu temsil eden sınıftır. Bir buton, araç kitinin sağladığı pek çok grafikten biridir.

```
b = Button(root)
```

ifadesi, "b" isimli bir buton nesnesi oluşturur ve nesneyi "root" pencere ile ilişkilendirir.

```
b.configure(background="yellow    ", text="Tıklama Sayısı = 0",
command=update)
```

ifadesi, butonu sarı arka plan, metin rengi ve basıldığında gerçekleşecek işlem açısından konfigüre eder.

"Button" nesnesi ile yazı tipi ve rengi, sola ya da sağa hizalı yazma, yatay ya da dikey konumlama ve çerçeve kalınlığı gibi seçenekleri de değiştirebiliriz.

```
b.pack()
```

ifadesi, butonun pencerede görünür iyi bir noktada yer almasını sağlar.

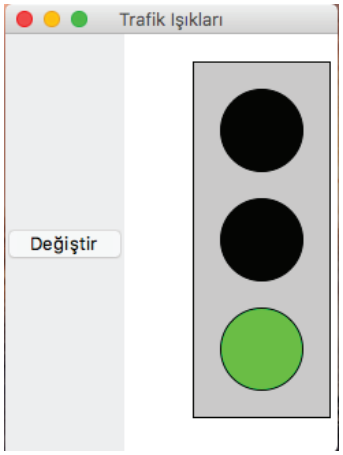
```
b = Button(root, background ="yellow", text="Tıklama Sayısı
= 0", command=update)
```

ifadesinde gerekli olan pencere parametresi ile birlikte (root) 3 parametrenin daha gönderildiğini görüyoruz. Bu örnekte "sayac" değişkeni global olarak tanımlanmalıdır çünkü fonksiyonda tekrar atama yapılmaktadır. Ayrıca, "b" değişkeni de global olmalıdır.

Trafik Işıkları Örneği

```
# "Değiştir" butonuna basıldığında trafik ışıklarını sırasıyla
yakan program
from tkinter import Tk, Canvas
from tkinter.ttk import Button, Frame
def ButonaBasildiginda(): # Her bir tıklamada ışıkların sırasıyla
yanması
    global renk
    if renk == "red":
        renk = "green"
        canvas.itemconfigure(kirmiziLamba, fill="black") # Kırmızı
ışık kapatılıyor
        canvas.itemconfigure(yesilLamba, fill="green") # Yeşil ışık
yanıyor
    elif renk == "green":
        renk = "yellow"
        canvas.itemconfigure(yesilLamba, fill="black") # Yeşil ışık
kapatılıyor
        canvas.itemconfigure(sariLamba, fill="yellow") # Sarı ışık
yanıyor
    elif renk == "yellow":
        renk = "red"
        canvas.itemconfigure(sariLamba, fill="black") # Sarı ışık
kapatılıyor
```

```
            canvas.itemconfigure(kirmiziLamba, fill="red") # Kırmızı
ışık yanıyor
# Kullanılacak değişkenlerin tanımlanması
renk = "red" # Açık olarak gelecek ilk trafik ışığı
renk root = Tk() # Ana Pencere"nin oluşturulması
root.title("Trafik Işıkları") # Pencere başlığı
frame = Frame(root) # Nesnelerin birlikte tutulması için grafiksel
bileşen ( widget ) oluşturuluyor
frame.pack() # Pencere içerisine frame yerleştiriliyor
# Grafiksel bileşenlerin yerleştirileceği çizim alanı ( canvas )
oluşturuluyor
canvas = Canvas(frame, width=150, height=300)
# frame"in içerisinde çizim arayüzü oluşturuluyor
# Trafik ışıkları oluşturuluyor, zemin rengi gri olarak ayarlanıyor
canvas.create _ rectangle(50, 20, 150, 280, fill="gray")
# Kırmızı Lamba
kirmiziLamba = canvas.create _ oval(70, 40, 130, 100, fill="red")
# Sarı Lamba
sariLamba = canvas.create _ oval(70, 120, 130, 180, fill="black")
# Yeşil Lamba
yesilLamba = canvas.create _ oval(70, 200, 130, 260, fill="black")
# Grafiksel butonun oluşturulması ve işlevsellik kazandırılması
Butona Basildiginda # Fare ile tıklama yapıldığında fonksiyon
çağrılıyor
button = Button(frame, text="Değiştir", command=ButonaBasildiginda)
# Oluşturulan katmanın ilk satır ve ilk sütününa buton,
# birinci satır ikinci sütununa da çizim alanı yerleştiriliyor.
button.grid(row=0, column=0)
canvas.grid(row=0, column=1)
# Grafiksel arayüz oluşturuluyor
root.mainloop()
```



Mevcut kodlama içinde tkinter paketinden ve tkinter.ttk paketinden ayrı sınıflar kullanılmaktadır. "Tk" sınıfı bir grafik penceresi sunar.

```
root = Tk()
```

komutu "root" isimli bir Tk nesnesi oluşturur ve "root" nesnesi uygulamanın ana grafik penceresi ile bağlantı kurar.

```
root.title("Trafik Işıkları")
```

pencerenin başlık çubuğundaki metni belirler.

```
root.mainloop()
```

"mainloop" yöntemini çağırarak grafik programı başlatır.

"Frame" sınıfı diğer grafik nesnelerini barındırmak için görünmeyen bir depo (widget) oluşturur.

```
frame = Frame(root)
```

komutu bir çerçeve nesnesi oluşturarak nesneyi grafik penceresi ile ilişkilendirir.

```
frame.pack()
```

komutu ise grafik penceresinin tamamını doldurur. "widget" görsel programlamada bir kütüphanedeki grafik bileşenlere verilen isimdir.

"Canvas" sınıfı grafik pencere içinde bir çizim alanı oluşturur.

```
canvas = Canvas(frame, width=300, height=300)
```

komutu çerçevenin deposunda yer alan "canvas" isimli bir nesne oluşturur. "canvas" nesnelerinin boyutları 300 px x 300 px olarak yükseklik ve genişlik anahtar kelimeleri ile belirlenir. Canvas üzerindeki koordinat sisteminin merkezi (0;0), pencerenin çizim alanının sol üst köşesinde yer alır ve y ekseni yukarı değil aşağı doğru şekillenir. Diğer bir ifade ile, soldan sağa doğru x değeri arttıkça, y değeri yukarıdan aşağıya doğru artmaktadır.

9.6. Nesne Değişkenliği ve Örtüşme

```
from fractions import Fraction
# Bazı kesir tanımlamaları yapılıyor
f1 = Fraction(1, 2)
f2 = Fraction(1, 2)
f3 = f1
# İlişkilendirmeler
print("f1 =", f1)
print("f2 =", f2)
print("f3 =", f3)
# Pay ve paydalar ayrı ayrı inceleniyor
print("f1--> Pay, Payda:", f1.numerator, f1.denominator)
print("f2--> Pay, Payda:", f2.numerator, f2.denominator)
print("f3--> Pay, Payda:", f3.numerator, f3.denominator)
# Kesirler karşılaştırılıyor
```

```
print("f1 == f2 ?", f1 == f2)
print("f1 == f3 ?", f1 == f3)
print("f1 ile f2 aynı değer mi?", f1 is f2)
print("f1 ile f3 aynı değer mi?", f1 is f3)
```

Ekran Çıktısı

```
f1 = 1/2
f2 = 1/2
f3 = 1/2
f1--> Pay, Payda: 1 2
f2--> Pay, Payda: 1 2
f3--> Pay, Payda: 1 2
f1 == f2 ? True
f1 == f3 ? True
f1 aynı f2 değer mi? False
f1 aynı f3 değer mi? True
```

```
f1 = Fraction(1, 2)
```

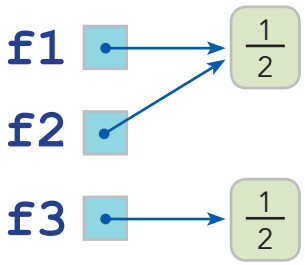
Komutu "Fraction (kesir)" sınıfını çağırarak yeni bir kesir nesnesi oluşturur. Bu komut ile paya 1 ve paydaya 2 değeri atanır. "f1" değişkeni de bu yeni kesir nesnesine atanır.

```
fraction object. The statement
f2 = Fraction(1, 2)
```

komutu aynı şekilde davranarak "f2" nesnesi de bu kesir nesnesine atanır.

```
f3 = f1
```

komutu ile "f3" değişkeni diğerleri ile aynı kesir nesnesine atanır. Ancak, söz dizimi "Fraction" sınıf yapısını içerdiğinden yeni bir kesir nesnesi oluşturmaz. Bu aşamada iyi kesir nesnemiz ve buna bağlı 3 değişkenimiz vardır. Aşağıda nesneler ve kesir arasındaki ilişki görülmektedir.



Görüldüğü gibi f1 ve f3 aynı nesneyi işaret etmektedir. Bu durum "örtüşme" olarak ifade edilir yani f1 ile f3 örtüşmektedir (f1 aliases f3). Kesirlerin mantıksal olarak eşitliğini karşılaştırmak için kesir nesnesine ait __eq__ yöntemi "==" operatörü ile kullanılır.

```
print("f1 == f2?", f1 == f2)
print("f1 == f3?", f1 == f3)
```

komutlarının üçü de mantıksal olarak birbirine eş değer değişkenler olduğunu açığa çıkarmaktadır. "__eq__" yöntemi (==) ile kesirlerin pay ve payda değerleri eşitlik konusunda karar verebilmek için ayrı ayrı karşılaştırılmaktadır.

Bazen mantıksal eşitlik yeterli olmaz ve her iki değişkenin de aynı nesneye ait olup olmadığını bilmek isteriz.

```
print("f1 ile f2 aynı değer mi?", f1 is f2)
print("f1 ile f3 aynı değer mi?", f1 is f3)
```

komutları f1 ve f2'nin iki farklı nesneyi, f1 ve f3'ün ise aynı nesneye işaret ettiğini gösterir. Bu durumda f1 ve f3 örtüşmektedir.

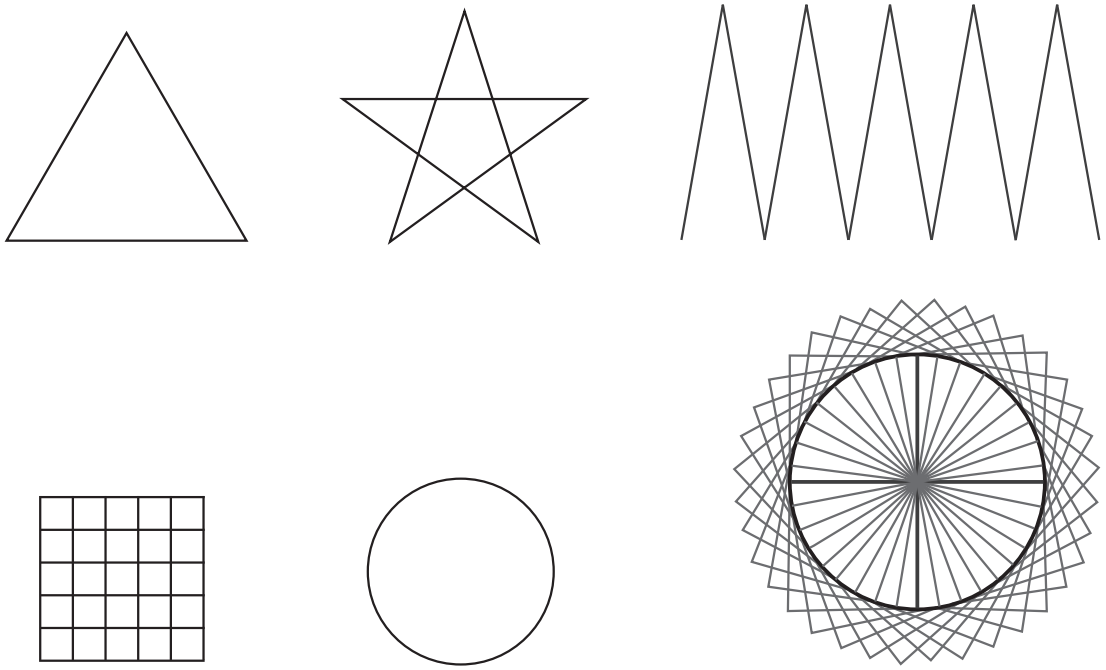
Python "id" isimli bir; fonksiyonu vardır ve bu fonksiyon, her bir nesne için özel oluşturulmuş bir tam sayı değeri döndürür (Çoğu Python uygulamalarında bu değer programın nesneyi yerleştirdiği hafızanın başlangıç adresidir.). a ve b nesne ise bu nesnelerin eş değerliği aşağıdaki biçimde sorgulanır.

```
id(a) == id(b)
```

Nesnelerin örtüşüp örtüşmediğine bakılırken değişkenlerin türü önemli değildir. Tam sayı değeri olarak 3 her zaman 3'tür. Dizi olarak "Fred" kelimesi "Free" olarak değişemez. Kesir sınıfı örnekleri de bu şekilde değişkendir. Değişken nesneler için örtüşme önemli bir konu olabilir. Python'un Turtle grafik kütüphanesindeki nesneler değişkendir. Programcılar kaplumbağa nesnesini hareket ettirebilir ve yönünü ve çizim rengini değiştirebilirler. Her bir işlem kaplumbağanın durumunu değiştirir ve grafik pencere içinde kaplumbağanın çizim sürecini etkiler.

Düşünelim/Deneyelim

Python Turtle grafikleri kullanarak aşağıdaki şekilleri oluşturunuz.



10. LİSTELER

Bu bölümde;

- ✓ Listelerin kullanım amacı konusunda bilgi sahibi olacak,
- ✓ Listeleri içeren programları yazabilecek,
- ✓ Listeler üzerinde işlemler yapabilecek,
- ✓ Listeler ve fonksiyonları bir arada kullanan programlar geliştirebilecek,
- ✓ Farklı boyutları olan listeleri içeren programları yazabileceksiniz.

10.1. Liste Kavramı

Bu konuya kadar kullanılan değişkenler, bir değeri temsil ediyordu. Ancak listeler ile daha fazla değeri bir değişkene aktarmak mümkün. Aşağıdaki örnekte beş değer için beş değişken kullanılmıştır.

```
def main():
    print("Lütfen 5 adet sayı giriniz: ")
    n1=float(input("1. sayı: "))
    n2=float(input("2. sayı: "))
    n3=float(input("3. sayı: "))
    n4=float(input("4. sayı: "))
    n5=float(input("5. sayı: "))
    print("Girdiğiniz Sayılar :",n1,n2,n3,n4,n5,sep=" - ")
    print("Ortalama :",(n1+n2+n3+n4+n5)/5)
main()
```

Ekran Çıktısı

```
Lütfen 5 adet sayı giriniz:
1. sayı: 23
2. sayı: 35
3. sayı: 55
4. sayı: 67
5. sayı: 98
Girdiğiniz Sayılar : - 23.0 - 35.0 - 55.0 - 67.0 - 98.0
Ortalama : 55.6
```

5 değil de 25 değere ihtiyaç duyulsa idi bir önceki soruya 20 değer daha eklemek gerekecekti. Oysaki değişken kullanımı konusunda alternatif yaklaşımlar ile bu durum daha pratik bir şekilde çözülebilir.

```
def main():
    toplam=0.0
    girilecekSayıAdeti=5
    print("Lütfen ",girilecekSayıAdeti, " adet sayı giriniz: ")
    for i in range(0, girilecekSayıAdeti):
        sayi=float(input("Lütfen " +str(i+1)+ ". sayıyı giriniz: "))
        toplam+=sayi
    print("Ortalama :",toplam/girilecekSayıAdeti)
main()
```

Ekran Çıktısı

```
Lütfen  5  adet sayı giriniz:
Lütfen 1. sayıyı giriniz: 12
Lütfen 2. sayıyı giriniz: 23
Lütfen 3. sayıyı giriniz: 34
Lütfen 4. sayıyı giriniz: 65
Lütfen 5. sayıyı giriniz: 76
Ortalama : 42.0
```

10.1.1. Listeleri Kullanmak

Listeler, belirli bir sırada çoklu değer tutmak için kullanılır. Listeler bir anlamda string veri türüne benzemektedir. Ancak, string veri türünün aksine herhangi bir liste içerisinde herhangi bir Python nesnesi korunabilir. Bir liste, tüm değişkenler gibi yerel ve global olarak kullanılabilir. Listenin kullanılmadan önce tanımlanması gerekmektedir.

Liste kullanırken değerler köşeli ayraç içerisinde virgül ile ayrılarak yazılır.

```
lst=[2,-3,0,4,-1]
A=[]
```

Örnek

```
lst=[2,-3,0,4,-1]
print([2,-3,0,4,-1])
print(lst)
```

Ekran Çıktısı

```
[2,-3,0,4,-1]
[2,-3,0,4,-1]
```

Örnek

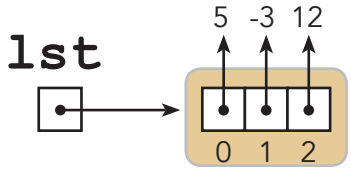
```
lst=[2,-3,0,4,-1]
lst[0]=5
print(lst[1])
lst[4]=12
print(lst)
print([10,20,30][1])
```

Ekran Çıktısı

```
-3
[5,-3,0,4,12]
20
```

Listelerde,

- Köşeli ayraç içerisindeki sayıya indeks denmektedir.
- lst[0] değeri ilk değerdir.
- lst[1] değeri serideki ikinci değerdir.
- 0, başlangıç indeksi olduğu için n elemanlı bir serinin indeks değeri n-1 olur.



- A[-1] değeri, A dizisindeki son elemanı ifade eder.
- A[-2] ise sondan bir önceki değeri ifade eder. Bu şekilde devam eder.

Örnek

```
def main():
      data=[10,20,30,40,50,60]
      print(data[-1])
      print(data[-2])
      print(data[-3])
      print(data[-4])
      print(data[-5])
      print(data[-6])
main() # Ana program çalıştırılıyor
```

Ekran Çıktısı

```
60
50
40
30
20
10
```

10.1.2. Listelerin Farklı Kullanımları

```
karisikListe=[24.2,4,"Python","Bilisim",19,-0.03,"kelime"]
```

Örnekte görüldüğü gibi bir listede tam sayı, ondalık sayı, string ve hatta fonksiyonlar olabilir. Liste içerisinde liste bile yer alabilir.

```
col=[23,[9.3,11.2,99.0],[23],[],4,[0,0]]
print(col)
```

Köşeli ayraç içerisinde ilgili değerler ile işlem yapılabilir.

```
nums=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10 , 11 , 12, 13, 14, 15]
print(nums[3])
nums[2]=(nums[0]+nums[9])/2;
nums[1], nums[4]=sqrt(x), x+2*y
```

String içerisindeki değerlere de köşeli ayraçlar ile erişilebilir.

```
>>>s= "ABCEFGHI"
>>>s[0]
"A"
>>>s[1]
"B"
```

Köşeli ayraç içerisindeki değer bir tam sayı olmalıdır.

```
a[x]
```

Köşeli ayraç içerisinde aritmetik bir işlem yapılabilir.

```
a[x+3]
```

Geri dönen değer de kullanılabilir. Bunlar da tam sayı olmalıdır.

```
a[max(x,y)]
```

10.1.3. Değerler Arası Gezinme

Bir listedeki değerlerin her birini ziyaret etmeye **gezinme** denmektedir. Bir liste aslında tekrarlanan bir nesne gibidir ve bu nedenle elemanları arasında gezinmek için for döngüsünü kullanabiliriz.

```
karisikListe=[24.2,4,"Python","Bilisim",19,-0.03,"kelime"]
```

```
for item in collection:
print(item) # Her bir eleman yazdırılıyor
```

Örnek

```
print(len([2,4,6,8]))
a=[10,20,30]
print(len(a))
>>>4
```

```
>>>3 değerlerini ekrana yazar.
```

Örnek

```
nums=[2,4,6,8]
for i in range(len(nums)-1,-1,-1):
print(nums[i])
>>>8
>>> 6
>>> 4
>>> 2 değerlerini ekrana yazar.
```

Örnek

```
nums=[2,4,6,8]

for i in reversed(nums):

print(nums[i])
```

Örnek

```
def my _ reversed(lst):

for i in range(len(lst)-1,-1,-1):

yield(lst[i])
```

Len(lst)-1 listedeki son değerin indeks değeridir. İkinci argüman ise döngüyü sonlandırmak içindir. Son –1 değeri ise geri saymak içindir.

10.2. Liste Oluşturmak

Python, liste oluşturmak için birçok yöntem sunmaktadır. Birleştirme yöntemi ile var olan iki liste tek bir liste yapılabilir. Bunun için + operatörü kullanılır.

```
>>> a = [2, 4, 6, 8]
>>> a
[2, 4, 6, 8]
>>> a + [1, 3, 5]
[2, 4, 6, 8, 1, 3, 5]
>>> a
[2, 4, 6, 8]
>>> a = a + [1, 3, 5]
>>> a
[2, 4, 6, 8, 1, 3, 5]
>>> a += [10]
```

```
>>> a
[2, 4, 6, 8, 1, 3, 5, 10]
>>> a += 20
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
  a += 20
TypeError: "int" object is not iterable
```

a = [2, 4, 6, 8] ifadesi ile a listesine değer atanır.

a + [1, 3, 5] ifadesi ile değerler [2, 4, 6, 8, 1, 3, 5] şeklini alır. Ancak atama gerçekleşmez.

a = a + [1, 3, 5] ifadesi ile değerler değişir.

a += [10] ifadesi ile liste [2, 4, 6, 8, 1, 3, 5, 10] şeklini alır.

a += 20 ifadesi bir liste olmadığı ve tam sayı olduğu için işlem yapmaz.

Bir değişkenin değeri listeye eklenmek istenirse köşeli ayraç kullanılmalıdır.

```
>>> x = 2
>>> a = [0, 1]
>>> a += [x]
>>> a
[0, 1, 2]
```

Örnek

```
# Kullanıcıdan alınan sayılardan liste oluşturan program
def ListeOlustur():
    sonuc = []
    girilenSayi = 0
    while girilenSayi >= 0:
        girilenSayi= int(input("Sayı giriniz ( Çıkış için -1): "))
        if girilenSayi>= 0:
            sonuc += [girilenSayi]
    return sonuc
def main():
    sutun = ListeOlustur()
    print(sutun)
main()
```

Ekran Çıktısı

```
Sayı giriniz ( Çıkış için -1): 2
Sayı giriniz ( Çıkış için -1): 3
Sayı giriniz ( Çıkış için -1): 4
Sayı giriniz ( Çıkış için -1): 55
```

```
Sayı giriniz ( Çıkış için -1): 66
Sayı giriniz ( Çıkış için -1): 77
Sayı giriniz ( Çıkış için -1): -1
[2, 3, 4, 55, 66, 77]
```

10.3. List Fonksiyonu ile Tam Sayı Liste Yapmak

Liste oluşturulmak istenirken bazen kullanıcılardan veri almak yerine liste elemanları otomatik olarak oluşturulmak istenebilir. Bu durumlarda list() fonksiyonuyla beraber range() fonksiyonu kullanılabilir. list() fonksiyonu otomatik olarak belirlenen değerlerden liste oluştururken range() fonksiyonu ise başlangıç ve bitiş değerleri arasında artım miktarına göre değerler oluşturur.

```
def main():
     a = list(range(0, 10))
     print(a)
     a = list(range(10, -1, -1))
     print(a)
     a = list(range(0, 100, 10))
     print(a)
     a = list(range(-5, 6))
     print(a)
main()
```

Ekran Çıktısı

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

20 ile 50 Arasındaki Asal Sayılardan Liste Yapmak

```
from math import sqrt
def AsalKontrol(n):
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    geciciDeger = 3
```

```
    kok = sqrt(n)
    while geciciDeger <= kok:
        if n % geciciDeger == 0:
            return False
        geciciDeger += 2
    return True
def AsalAralikBelirleme(ilk, son):
    for deger in range(ilk, son + 1):
        if AsalKontrol(deger):
            yield deger
def main():
    asal = list(AsalAralikBelirleme(20, 50))
    print(asal)
if __name__ == "__main__":
    main() # Programı çalıştır
```

Ekran Çıktısı

```
[23, 29, 31, 37, 41, 43, 47]
```

10.4. * Operatörü ile Liste Oluşturma

Daha önceki konularımızda * operatörü matematikteki çarpma işlemini yerine getirmek için kullanılıyordur. Liste oluşturulurken * operatörü solundaki liste elemanını sağındaki adet kadar çoğaltır.

* Operatörü string'ler ile benzer sonuç verir.
>>> 'abc' * 3
'abcabcabc'

```
def main():
     a = [0] * 10
     print(a)
     a = [3.4] * 5
     print(a)
     a = 3 * ["ABC"]
     print(a)
     a = 4 * [10, 20, 30]
     print(a)
     n = 3 # çarpım değişkeni tanımlanıyor
```

```
    a = n * ["abc", 22, 8.7]
    print(a)
main()
```

Ekran Çıktısı

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[3.4, 3.4, 3.4, 3.4, 3.4]
["ABC", "ABC", "ABC"]
[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]
["abc", 22, 8.7, "abc", 22, 8.7, "abc", 22, 8.7]
```

Örnek

Listedeki değerlerin ortalamasını bulmak.

```
def main():
    toplam = 0.0
    girilecekSayiAdeti = 5
    sayilar = []
    print("Lütfen", girilecekSayiAdeti, " adet sayi giriniz :")
    for i in range(0, girilecekSayiAdeti):
        sayi = float(input(str(i+1) + " . sayıyı giriniz : "))
        sayilar += [sayi]
        toplam += sayi

    for sayi in sayilar:
        print(end="Girilen Sayılar: ")
        print(sayi, end="")
        print() # Alt satıra geçiş
    print("Ortalama :", toplam/girilecekSayiAdeti)
main()
```

Ekran Çıktısı

```
Lütfen 5  adet sayi giriniz :
1 . sayıyı giriniz : 12
2 . sayıyı giriniz : 23
3 . sayıyı giriniz : 43
4 . sayıyı giriniz : 54
```

```
Girilen Sayılar: 12.0
Girilen Sayılar: 23.0
Girilen Sayılar: 43.0
Girilen Sayılar: 54.0
Girilen Sayılar: 55.0
Ortalama : 37.4
```

10.5. Liste Üyeliği

in operatörü ile listenin elemanları üzerinde işlem yapılabilir. True veya false değeri döndürür.

```
liste = list(range(0, 21, 2))
for i in range(-2, 23):
    if i in liste:
        print(i," eleman, ", liste," listesinin bir üyesidir.")
    if i not in liste:
       print(i," eleman ", liste, "listesinin bir üyesi değildir.")
```

Ekran Çıktısı

```
-2 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
-1 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
0 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
1 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
2 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
3 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
4 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
5 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
6 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
7 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
8 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
9 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
10 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
11 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
12 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
13 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
14 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
15 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
16 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
```

```
17 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
18 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
19 eleman [[0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
20 eleman, [0,2,4,6,8,10,12,14,16,18,20]] listesinin bir üyesidir.
21 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
22 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
```

10.6. Listeye Değer Atama ve Eşitleme

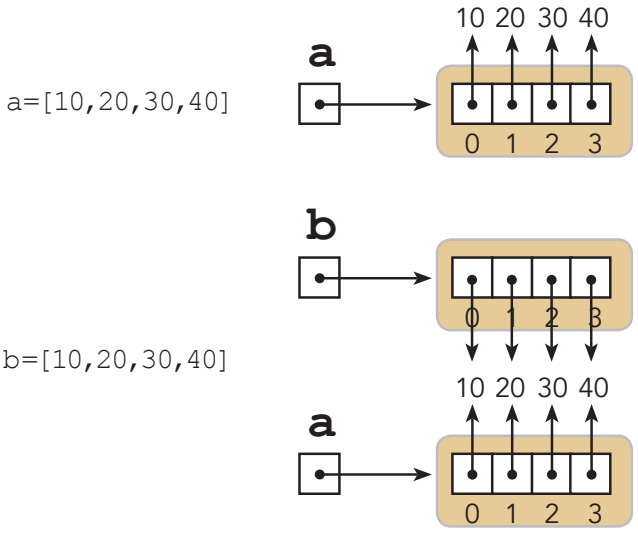
Örnek

```
a=[10,20,30,40]
b=[10,20,30,40]
print("a =",a)
print("b =",b)
b[2]=35
print("a =",a)
print("b =",b)
```

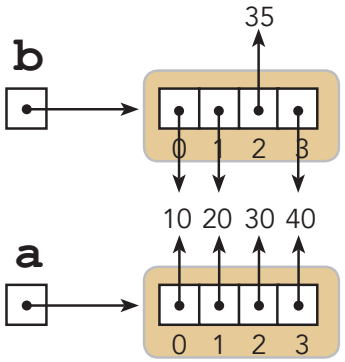
Ekran Çıktısı

```
a=[10,20,30,40]
b=[10,20,30,40]
a=[10,20,30,40]
b=[10,20,35,40]
```

Programın çalışma süreci aşağıda verilen resimdeki gibidir.



b[2] = 35



Örnek

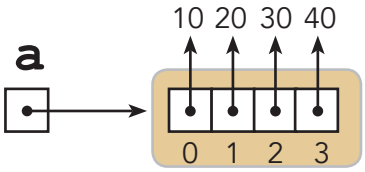
```
a=[10,20,30,40]
b=a
print("a =",a)
print("b =",b)
b[2]=35
print("a =",a)
print("b =",b)
```

Ekran Çıktısı

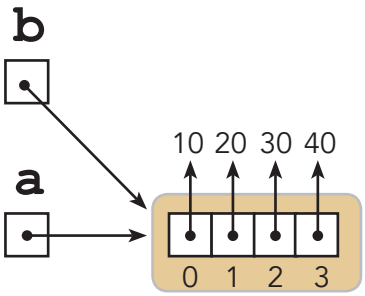
```
a=[10,20,30,40]
b=[10,20,30,40]
a=[10,20,30,40]
b=[10,20,35,40]
```

Programın çalışma süreci aşağıda verilen resimdeki gibidir.

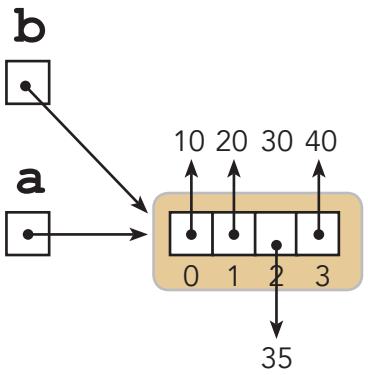
a=[10,20,30,40]



b=a



b[2] = 35



Örnek

```
a=[10,20,30,40]
b=[10,20,30,40]
print(a,"değeri ",b," değerine eşit mi?",sep="",end=" ")
print(a==b)
print(a,"değeri ",b," değeri ile aynı mı?",sep="",end=" ")
print(a is b)
c=[100,200,300,400]
d=c
print(c,"değeri ",d," değerine eşit mi?",sep="",end=" ")
print(c==d)
print(c,"değeri ",d," değeri ile aynı ma?",sep="",end=" ")
print(c is d)
```

Ekran Çıktısı

```
[10, 20, 30, 40]değeri [10, 20, 30, 40] değerine eşit mi? True
[10, 20, 30, 40]değeri [10, 20, 30, 40] değeri ile aynı mı? False
[100, 200, 300, 400]değeri [100, 200, 300, 400] değerine eşit mi? True
[100, 200, 300, 400]değeri [100, 200, 300, 400] değeri ile aynı ma? True
```

İki değişkenin değerlerinin aynı olması (== ile karşılaştırma) ile değişkenlerin aynı olması (is ile karşılaştırma), farklı durumlardır. Örnekte a ve b değişkenlerinin değerleri aynıdır dolayısıyla == ile karşılaştırma yapıldıklarında True değerini verir. Ancak değişken değerlerinin atamaları ayrı yapıdığı için is ile karşılaştırma yapıldığında False değeri verir. Bu durum c ile d değişkenlerinde ayrılır. Çünkü d değişkenine c değişkeni atanmış ve birbirlerine eşitlenmiştir. Dolayısıyla is ile karşılaştırma yapıldığında True değerini döndürür.

Örnek

```
def ListeKopyala(lst):
    result=[]
    for item in lst:
        result+=[item]
    return result
def main():
    a=[10,20,30,40]
    b=ListeKopyala(a)
    print("a =",a, " b =",b)
  print(a, "listesi ile ",b, "listesi değerleri eşit mi?",sep="",end="")
    print(a==b)
    print(a, " ile ",b, " listeleri aynı mı?",sep="",end="")
    print(a is b)
    b[2]=35
    print("a =",a, " b =",b)
main()
```

Ekran Çıktısı

```
a = [10, 20, 30, 40]  b = [10, 20, 30, 40]
[10, 20, 30, 40]listesi ile [10, 20, 30, 40]listesi değerleri eşit mi?True
[10, 20, 30, 40] ile [10, 20, 30, 40] listeleri aynı mı?False
a = [10, 20, 30, 40]  b = [10, 20, 35, 40]
```

Bu örnekte a adında bir liste tanımlanmış ve bu listenin ListeKopyala() fonksiyonu ile kopyası b değişkenine oluşturulmuştur. İki liste değerleri birbirine eşittir. Fakat birbirlerine eşitlenme durumu False'dur.

Örnek

```
print(list(range(11)))
print(list(range(10,101,10)))
print(list(range(10,-1,-1)))
```

Ekran Çıktısı

```
[0,1,2,3,4,5,6,7,8,9,10]
[10,20,30,40,50,60,70,80,90,100]
[10,9,8,7,6,5,4,3,2,1,0]
```

10.7. Listenin Sınırları

a=[10,20,30,40] listesinde a[0], a[1], a[2] ve a[3] değerleri bulunurken a[4] değeri bulunmamaktadır.

```
a=[10,20,30,40]
print(a[4]) # listenin sınırı dışından bir erişim yapılmaya çalışılmaktadır.
```

Bu durumda IndexError hatası oluşur.

```
# Değerleri 0 olan 100 elemanlı liste oluşturuluyor
v=[0]*100
x=int(input("Bir sayı giriniz: "))
# Girilen değer liste sınırları içerisinde mi?
if 0<=x<len(v):
     v[x]=1 # Girilen indis değeri 1 olarak değiştiriliyor
else:
     print("Girdiğiniz değer liste sınırları arasında değil")
```

10.8. Dilimleme

Bir liste dilimlenerek başka bir liste oluşturulabilir. Bir listeyi dilimlemek için aşağıdaki ifadeyi kullanmak gerekir.

list [başlangıç : bitiş : artım miktarı]

- List: liste değişkenini,
- başlangıç: listenin başlangıç indeksini,
- bitiş: listenin son elemanının indeksini,
- artım miktarı: verilen aralıktaki eleman sayısını belirtir.

```
Lst= [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst)          # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[0:3])     # [10, 20, 30]
print(Lst[4:8])     # [50, 60, 70, 80]
print(Lst[2:5])     # [30, 40, 50]
print(Lst[-5:-3])   # [80, 90]
print(Lst[:3])      # [10, 20, 30]
print(Lst[4:])      # [50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[:])       # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[-100:3]) # [10, 20, 30]
print(Lst[4:100])   # [50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[2:-2:2]) # [30, 50, 70, 90]
print(Lst[::])      # [10, 30, 50, 70, 90, 110]
```

```
print(Lst[::-1])    # [120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

10.8.1. Dilimleme İçin Farklı Kullanımlar

```
>>>a=[34,-19,20,8,12]
>>>print(a)
[34,-19,20,8,12]
>>>print(a[0:1])
[34]
>>>print(a[0])
34
>>>b=[[2,5],7,[11,4]]
>>>print(b)
[[2,5],7,[11,4]]
>>>print(b[0:1])
[[2,5]]
>>>print(b[0])
[2,5]
```

10.8.2. Dilimleme Örnekleri

Örnek

```
a=[1,2,3,4,5,6,7,8]
print("Listenin Soldan Başlanarak Dilimlenmesi",a)
for i in range(0,len(a)+1):
     print("<",a[0:i], ">",sep="")
print("-----------------------------------")
print("",a)
for i in range(0,len(a)+1):
     print("<",a[i:len(a)+1], ">",sep="")
```

Ekran Çıktısı

```
Litenin Soldan Başlanarak Dilimlenmesi [1, 2, 3, 4, 5, 6, 7, 8]
<[]>
<[1]>
<[1, 2]>
<[1, 2, 3]>
```

```
<[1, 2, 3, 4]>
<[1, 2, 3, 4, 5]>
<[1, 2, 3, 4, 5, 6]>
<[1, 2, 3, 4, 5, 6, 7]>
<[1, 2, 3, 4, 5, 6, 7, 8,]>
----------------------------------
Litenin Sağdan Başlanarak Dilimlenmesi [1, 2, 3, 4, 5, 6, 7, 8]
<[1, 2, 3, 4, 5, 6, 7, 8,]>
<[2, 3, 4, 5, 6, 7, 8,]>
<[3, 4, 5, 6, 7, 8,]>
<[4, 5, 6, 7, 8,]>
<[5, 6, 7, 8,]>
<[6, 7, 8,]>
<[7, 8,]>
<[8,]>
<[]>
```

Örnek

```
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:5]=["a", "b", "c"] #[30, 40, 50] değerleri ["a", "b", "c"] değerleri ile değiştiriliyor
print(lst)
print("--------------------")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:6]=["a","b"] #[30, 40, 50, 60] değerleri ["a", "b"] değerleri ile değiştiriliyor
print(lst)
print("--------------------")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:2]=["a", "b", "c"] # 2 nolu indexten başlayarak ["a", "b", "c"] değerleri ekleniyor
print(lst)
print("--------------------")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:5]=[] # [30, 40, 50] değerleri yerine [] atanıyor. (değerler siliniyor)
print(lst)
```

Ekran Çıktısı

```
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", "c", 60, 70, 80]
================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", 70, 80]
================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", "c", 30, 40, 50, 60, 70, 80]
================
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 60, 70, 80]
================
```

Listeler üzerinde değerleri değiştirme, değerler ekleme veya değerleri silme işlemleri yapılabilir.

10.9. Listeden Eleman Çıkarma

Listeden bir eleman silmek için del komutu kullanılır.

Örnek

```
>>>a=list(range(10,51,10))
>>>a
[10,20,30,40,50]
>>>del a[2]
>>>a
[10,20,40,50]
```

Örnek

```
>>>b=list(range(20))
>>>b
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
>>>del b[5:15]
>>>b
[0,1,2,3,4,15,16,17,18,19]
```

Örnek

```
>>>c=list(range(20))
>>>c
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
>>>del c[1],c[18]
>>>c
[0,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
```

10.10. Listeler ve Fonksiyonlar

Liste elemanları üzerinde, Python diline ait fonksiyonları aynı zamanda programcının yazdığı kodları kullanarak işlem yapabiliriz. Bu örneğimizde random() fonksiyon sınıfına ait metotlar ile eleman ataması yapıldı. Bu işlemler de programcı tarafından yazılan fonksiyonlar ile gruplanmıştır.

```
import random
def Topla(lst):
    sonuc=0
    for eleman in lst:
        sonuc+=eleman
    return result
def SifirAta(lst):
    for i in range(len(lst)):
        lst[i]=0
def RastgeleDegerAta(n):
    sonuc=[]
    for i in range(n):
        RastgeleDeger=random.randrange(100)
        sonuc+=[RastgeleDeger]
    return sonuc
def main():
    a=[2,4,6,8]
    print(a)
    print(sum(a))
    SifirAta(a)
    print(a)
    print(sum(a))
    a=[]
    print(a)
    print(sum(a))
   a=RastgeleDegerAta(10)
    print(a)
    print(sum(a))
main()
```

Ekran Çıktısı

```
2, 4, 6, 8]
20
[0, 0, 0, 0]
0
[]
0
[97, 28, 13, 38, 32, 59, 94, 60, 17, 55]
493
```

10.11. Listede Kullanılan Yöntemler

Listede Kullanılan Yöntemler	
count	Bir listede eleman sayısını geri döndürür. Listeyi değiştirmez.
insert	Verilen indeks numarasına yeni bir eleman yerleştirir. Listenin uzunluğunu bir arttırır. Listeyi değiştirir.
append	Listenin sonuna bir eleman ekler. Listeyi değiştirir.
index	Listede verilen elemanın en düşük indeks numarasını geri döndürür. Eğer eleman listede yoksa hata üretir. Listeyi değiştirmez.
remove	Listedeki ilgili elemanı siler. İlgili eleman bulunmaz ise hata verir. Listeyi değiştirir.
reverse	Listedeki elemanları fiziksel olarak ters çevirir. Listeyi değiştirir.
sort	Artan değer şeklinde listeyi sıralar. Listeyi değiştirir.

Listelerde, değişen veri yapıları olduğu için __getitem__ and __setitem__ yöntemleri de kullanılır.

```
x=lst[2] ifadesi x=list._ _getitem_ _(lst,2) ile
lst[2]=x ifadesi list._ _setitem_ _(lst,2,x)
```

ile benzer işlem yapar.

```
lst=["one","two","three"]
lst+=["four"]
```

=

```
lst=["one","two","three"]
lst.append("four")
```

10.12. Çok Boyutlu Listeler

Bir liste aslında tek boyutlu bir veri yapısıdır. İki boyutlu listeler, dikdörtgen olarak elemanlar dizisidir ve matriks olarak bilinir.

```
100 14 8 22 71
0 243 68 1 30
90 21 7 67 112
115 200 70 150 8
```

```
matrix=[[100, 14, 8, 22, 71],
        [0, 243, 68, 1, 30],
        [90, 21, 7, 67, 112],
        [115, 200, 70, 150, 8]]
```

```
>>>print(matrix)
[[100, 14, 8, 22, 71],[0, 243, 68, 1, 30],[90, 21, 7, 67, 112],
[115, 200, 70, 150, 8]]

print (matrix [2][3])
```

10.13. Çok Boyutlu Diziler

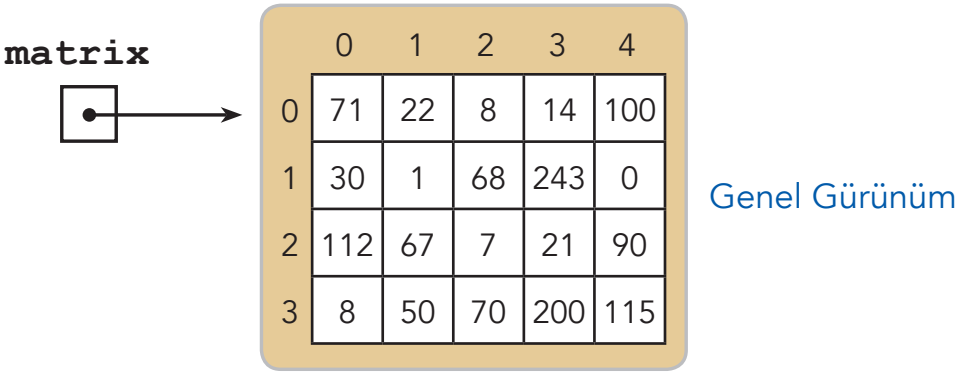
Aşağıdaki örnek iki boyutlu bir diziyi ekrana yazdırmaktadır.

```
matrix=    [[100,14,8,22,71],
      [0,243,68,1,30],
      [90,21,7,67,112],
      [115,200,70,150,8]]
for row in matrix:
      for elem in row:
            print("{:>4}".format(elem),end="")
      print()
```

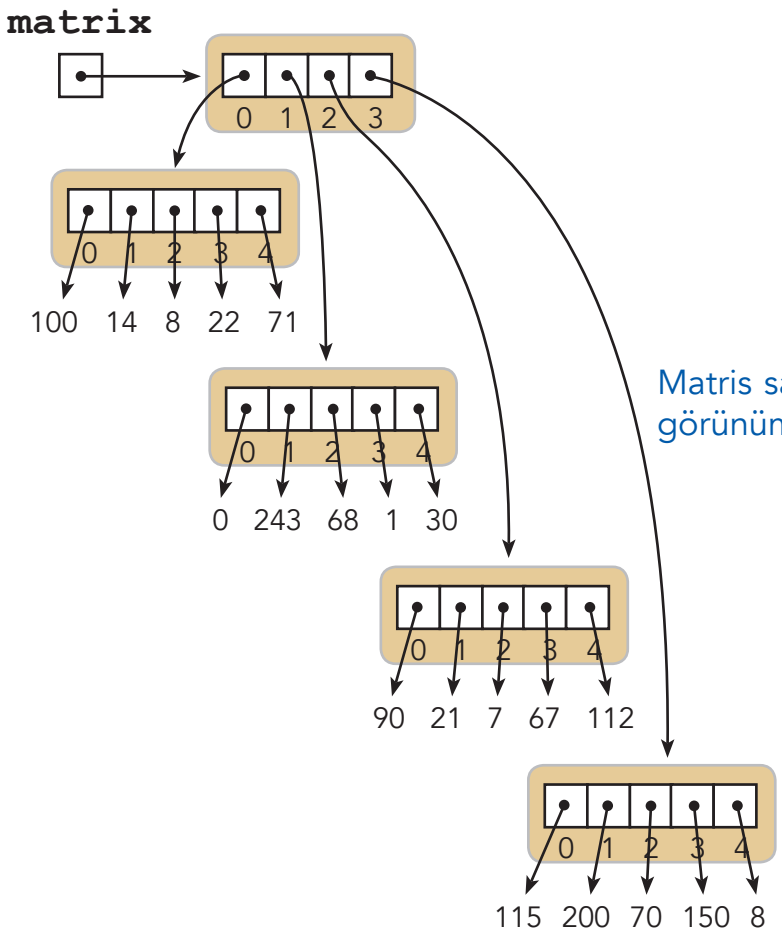
Ekran Çıktısı

```
100   14    8   22   71
  0  243   68    1   30
 90   21    7   67  112
115  200   70  150    8
```

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.



Genel Gürünüm



Matris satırlarının adım adım görünümü

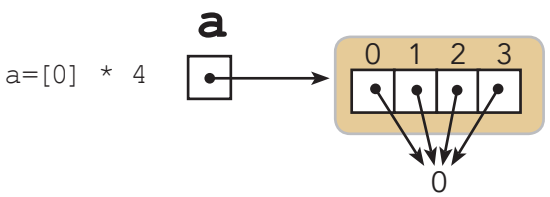
Örnek

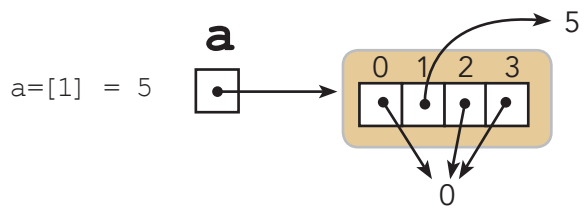
```
a=[[0]*4]*3
print(a)
a[1][2]=5
print(a)
```

Ekran Çıktısı

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 5, 0], [0, 0, 5, 0], [0, 0, 5, 0]]
```

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.

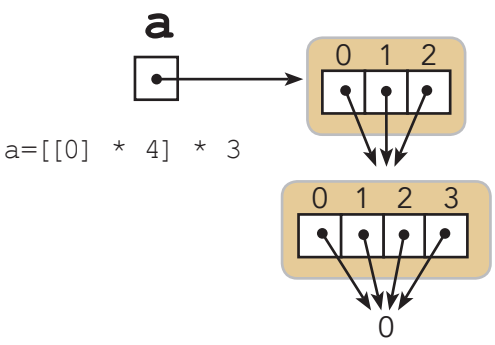


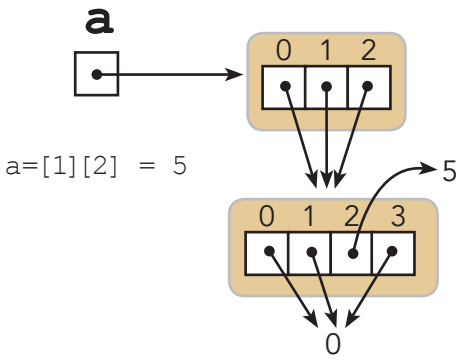


Örnek

`A=[[0] * 4a=[[0]*4 for x in range(3)]`	=	`a=[]`
		`for x in range(3):`
		`  a.append([0]*4)`

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.





10.14. _ Sembolü

Kullanım durumları, aşağıdaki örneklerde gösterilmektedir.

```
a=[[0]*4 for _ in range(3)]
```

veya

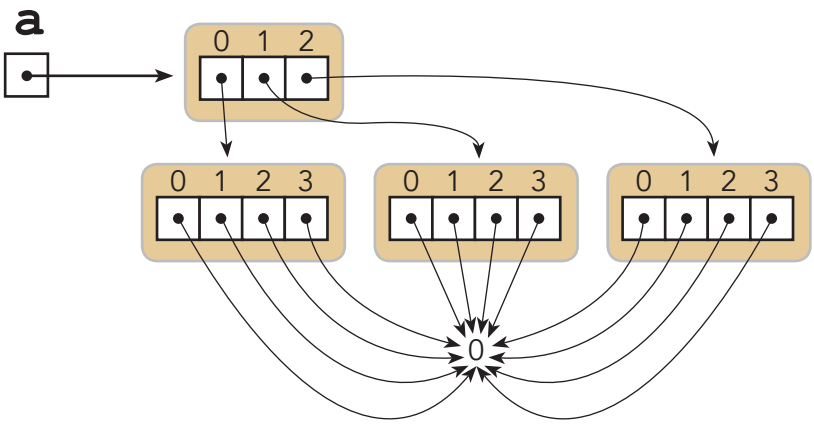
```
a=[[0 for _ in range(4)] for _ in range(3)]
```

Ekran Çıktısı

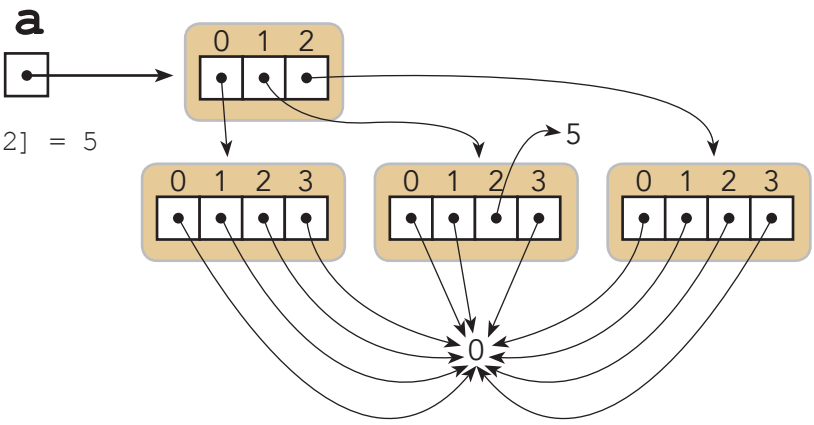
```
>>>10+4
14
>>> _
14
>>>100-60
40
>>>2+ _
42
>>>
```

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.

```
a=[[0 for _ in range(4)] for _ in range(3)]
```




```
a=[1][2] = 5
```



10.15. Liste Oluşturma Tekniklerinin Özeti

- Sıralı atama
  - L=[2,4,6,8,10,12,14,16,18,20]
- Düzensiz atama
  - L=[]
  - for i in range(2,21,2):
  - L+=[i]
  - L=[2,4,6,8,10,12,14,16,18,20]
- Range kullanımı
  - L=list(range(2,21,2))
- Liste oluşturma
  - L=[x for x in range(1,21) if x%2==0]
- Yöntemlerin birleşimi
  - L=list(range(2,9,2))+[10,12,14]+[x for x in range(16,21,2)]

SÖZLÜK

-A-

algoritma : Bir problem durumunu çözmek için ilgili adımların mantıksal sıralanması.

anlam bilimsel hata : Programın, programcının istediğinden farklı bir şey yapmasına neden olan hata.

-B-

bug : Programda oluşan bir hata.

-Ç-

çalışma zamanı hatası : Program çalıştırılana kadar ortaya çıkmayan, çalıştırıldıktan sonra oluşan ve programın çalışmasına devam etmesini engelleyen hata.

-D-

derleme : Yüksek seviyeli bir dilde yazılmış programı, düşük seviyeli bir dile çevirme işlemidir.

düşük seviyeli dil : Bilgisayarın kolay bir şekilde yürütmesi için tasarlanmış programlama dili. "Makine dili" veya "birleştirici dil" adı da verilmektedir.

-H-

hata ayıklama : Herhangi bir programlama hatasını bulma ve ortadan kaldırma sürecidir.

hedef kod : Derleyicinin programı (kaynak kodu) çevirmesiyle ortaya çıkan kod serisi.

-K-

kaynak kod : Yüksek seviyeli bir dildeki programın derlemeden önceki hâli.

-P-

print : Python yorumlayıcısının bir değeri ekranda göstermesini sağlayan komut.

problem çözme : Problemi formüle etme, probleme çözüm bulma ve problemin çözümü ifade etme süreci.

program : Bilgisayar tarafından gerçekleştirilecek eylemleri ve hesaplamaları belirtip art arda gelen komutlar.

-S-

söz dizimi : Programın yapısı.

söz dizimi hatası : Programın ayrıştırılmasını imkânsız hâle getiren hata (Dolayısıyla yorumlanmasını da imkânsız hâle getirir.).

-T-

taşınabilirlik : Bir programın birden fazla bilgisayar türünde yürütülebilmesi özelliği.

token : Programın söz dizimsel yapısının temel ögelerinden biri. Doğal dillerdeki kelimeye benzetilebilir.

-Y-

yorumlama : Yüksek seviyeli dilde yazılmış bir programı satır satır çevirerek yürütme.

yüksek seviyeli dil : İnsanlar tarafından rahatlıkla okunabilmesi, yazılması ve anlaşılması için geliştirilmiş Python benzeri programlama dili.

KAYNAKÇA

- Aksoy, A. **Yeni Başlayanlar İçin Python**. İstanbul: Abaküs Kitap. 2016.
- Downey, A., Elkner, J. & Meyers, C., **How to Think Like a Computer Scientist: Learning with Python (2nd Edition)**, Boston, MA: Green Tea Press, 2008.
- Halterman, R. L., **Fundamentals of Python Programming**, 2016. http://python.cs.southern.edu/pythonbook/pythonbook.pdf adresinden alınmıştır. Aralık, 2006.
- Levitin, A. & Levitin, M., **Algorithmic Puzzles**, New York: Oxford University Press, 2011.
- Sprankle, M. &Hubbard, J., **Problem Solving and Programming Concepts (9th Ed)**. New Jersey: Pearson Education, 2012.
- Spraul, V. A., **Think Like a Programmer**, San Francisco: No Starch Press, 2012.
- Burlu, K. **Bilişimin Karanlık Yüzü**. Nirvana Yayınları, 2011.
- Clarke, Richard A. & Knake, Robert K., **Siber Savaş**. İstanbul Kültür Üniversitesi - İKÜ Yayınevi, 2010.
- **Cyberpower and National Security (National Defense University), Franklin D. Kramer (Ed), Stuart H. Starr (Ed.), Larry Wentz (Ed.), 2009.**
- Floridi, L., **"Information ethics: On thephilosophicalfoundation of computerethics"**, **Ethicsand İnformation Technology**, 1(1), 33-52, 1999.
- G. Canbek, Ş. Sağıroğlu, **Bilgi ve Bilgisayar Güvenliği: Casus Yazılımlar ve Korunma Yöntemleri**, Grafiker Ltd. Ş. Aralık, 2006.
- Hoyk, R. & Hersey, P., **Theethical Executive: Becomingaware Of Therootcauses Of Unethical Behavior.** Stanford: Stanford University Press, 2008.
- Kayak, S., **"BÖTE Bölümü Öğrencilerinin İnternet Etiği Algılarının İncelenmesi"**, **Eğitim Teknolojileri Araştırmaları Dergisi,** 2011.
- Namlu, A. G. & Odabaşı, H. F., **"Unethical Computer Using Behavior Scale: A study of reliabilityandvalidity on Turkish University Students"**, **Computers & Education**, 48(2), 205-215, 2007.
- Quinn, M. J., **"On Teaching Computer Ethics Within A Computer Science Department", Science and Engineering Ethics**, 12, 335-343, 2006.
- Siegfried, R. M., **"Student Attitudes On Software Piracy And Related Issues Of Computer Ethics"**, **Ethicsand Information Technology**, 6, 215-222, 2004.
- Turilli, M. & Floridi, L., **"Theethics of Information Transparency", Ethicsand Information Technology**, 11(2), 105-112, 2009.