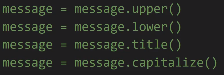
For/while döngülerinde x -= 1, x+= 1, x \*= 1, x /= 1, x %= 1, x \*\*= 1, x //= 1 (integer bölmesi) operatörleri sıkça kullanılır.

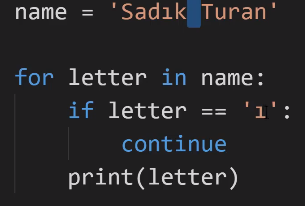
Bir variablenin başına “\*” eklediğimizde örn. “\*z” z değeri list haline dönüşür. Bunun yanında çoklu variableleri farklı değerlere eşlemek istiyorsak şu metodu uygularız, örn. x, y, z = 1, 3, 5.

Komut olan “is” objelerin aynı adresi tutup tutmadığını belirtir (“is not” komutu tersidir). “del” komutu ile değer sildirebiliriz. “in” komutu ile bir variablenin başka bir variable içerisinde barınıp barınmadığını sorgularız (“not in” komutu tersidir).



Something.strip() komutu stringteki (baş ve son) komutları kaldırırken something.split() komutu string kelimelerini ayrı ayrı yazdırmayı sağlar. Split metodu aynı zamanda belirli bir harfi referans alarak da stringi bölebilir. Message = ‘’.join(word) metodu ile stringleri birleştirebiliriz. “.find(‘some’) komutu ile kelime stringe var mı var ise hangi indexten başladığını belirtir. “.center(number, ‘sign’)” komutu ile stringi özelleştirebiliriz, .rjust/.ljust ile sağ ve sola entegre edebiliriz.

Break ve continue komutlarında break komutu istenen veriye ulaşılınca sistemin durdurulmasını, continue komutu ise istenen verinin atlanarak kod işlemlerinin devam etmesini sağlar. Bu komut ile 100’ e kadar olan tek sayıların toplami gibi bir kod yazılabilir. Örn.



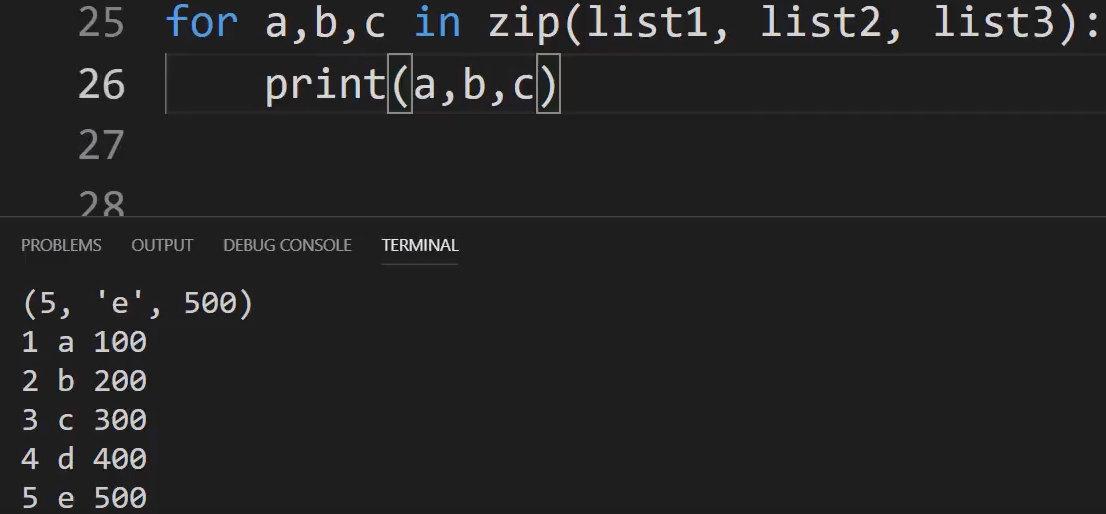
Range() komutu ile list için değerler girebiliriz, range(x, y, z)’de x başlangıç noktasını, y son noktayı (bu değere ulaşmaz), z ise artış miktarını ifade eder.

f-string komutu ile köşeli parantez kullanarak parantezin içerisine istediğimiz fonksiyonu sığdabiliriz.



Enumerate komutu ile yazdığımız stringin index numarası ve bu numaraya karşılık gelen harf (0, ‘a’) olacak şekilde gösterilir.

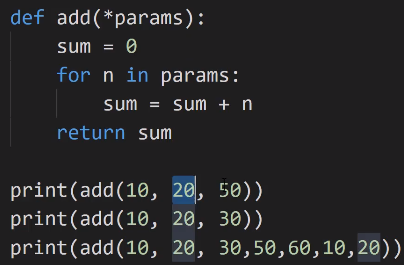
Zip(x, y,…) metodu ile x ve y’ı birleştirebiliriz. Birleşim outputta şu şekilde gözükür: [(a, b), (c, d)]



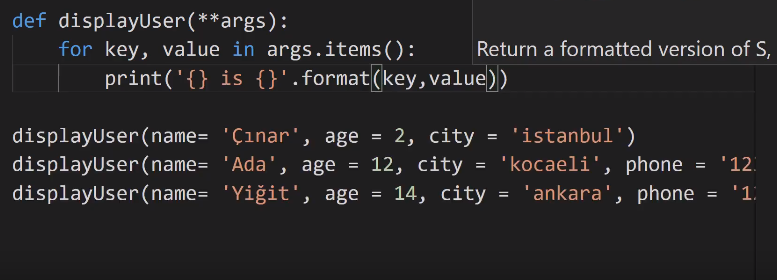
Örnekte olduğu gibi for döngüsü içerisinde a, b, c değerlerini listlere karşılık tutarak listx değerlerini teker teker de yazdırabiliriz.

Ctrl + K ve Ctrl + C komutlarına basarak multiline comment işlemi yapabiliriz, olanı commentten çıkarmak içinse Ctrl + K ve Ctrl + U kullanılır.

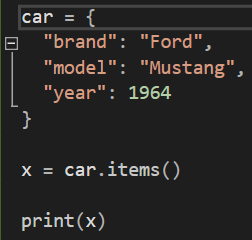
Function içerisinde def something(\*some) yazmak “some” ifadesinin bir tuple (list) olduğunu something(\*\*some) yazmak ise “some” ifadesinin bir dictionary olduğunu tanımlar. “(\*args, \*\*kwargs)” parametreleri bu yapıyı ifade ederken sıkça kullanılır fakat bu ifadelerin yalnızca isim olduğunu unutmamak gerek. Kaç adet parametre girileceği belirsiz olduğu zaman bu komutu kullanmak oldukça mantıklıdır.



Şekil 1: Bunu returnde sum() fonksiyonunu kullanarak da yapabiliriz.



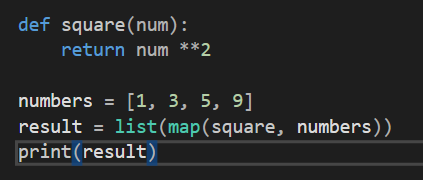
Yukarıdaki örnekte anladığım kadarıyla bir dictionary oluşturduğumuz için key ve value adında iki farklı parametre oluşturmamız lazım, “value” değerleri ifade ederken “key” ise stringleri kapsıyor. “.format(key, value)” komutu süslü parantezlerin içini doldurmak içindir, key ve value terimleri parantezlere yazılsa da olur.





Şekil : Örnek alternatifi

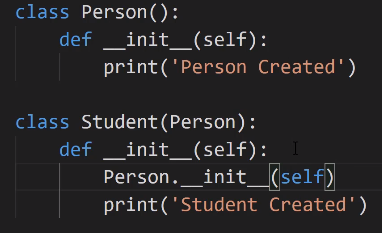
Anladığım kadarıyla “.items()” komutu tek seferde (bir satırda) integer ve stringlerin ifade edilmesine olanak tanıyor list içerisindeki tuple’lar olarak.



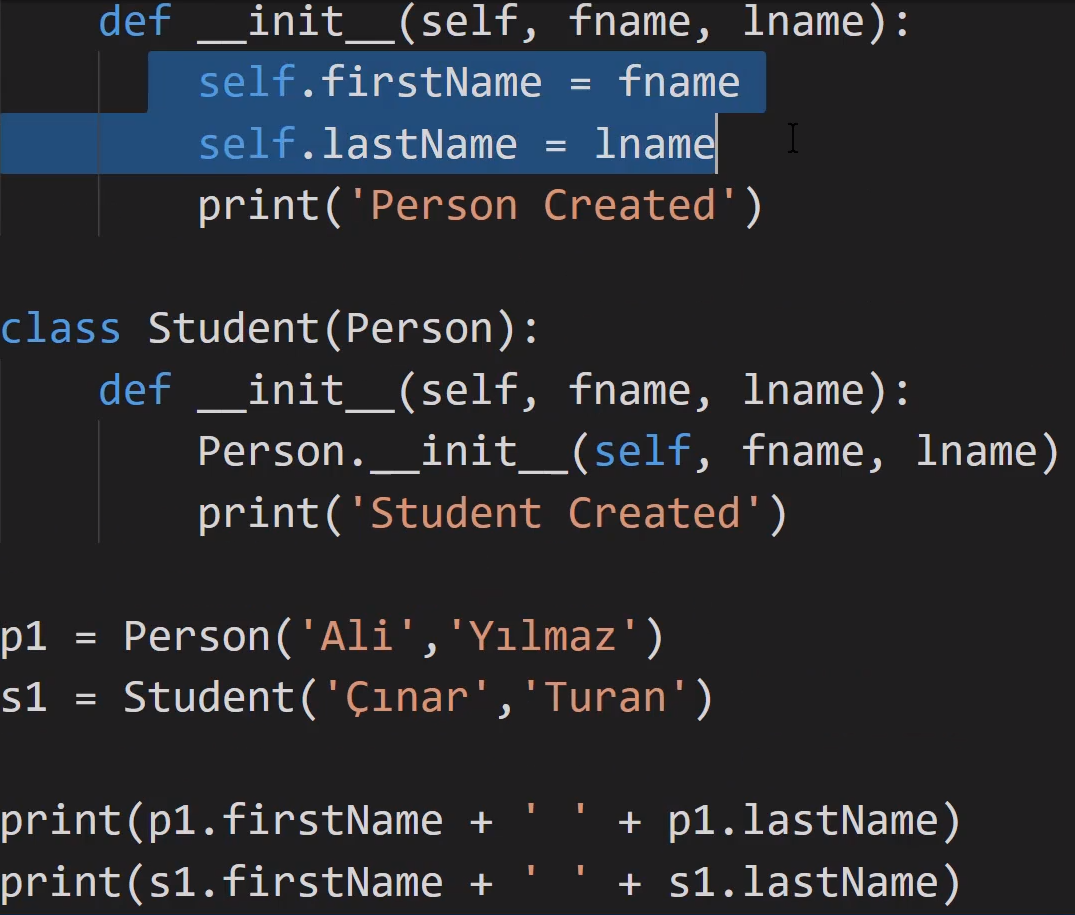
“map()” fonksiyonunu 1’den fazla parametreler içeren bir değeri list komutu içerisine yazarak list olarak oluşturabilir veya for döngüsü içerisine yazarak alt alta yazdırabiliriz. Örnekte gözüktüğü gibi fonksiyon değerden önce yazılır.

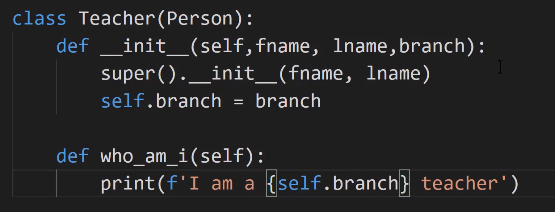
“def some()” fonksiyonunun içerisi boş bırakılacaksa (self) yazılmalıdır aksi takdirde function ilerde boş parantez ile çağırıldığında system error verir.

Bir def fonksiyonu içerisine başka bir def fonksiyonunu çağırırken çağırdığımız def fonksiyonunun içerisine some.init(self) olarak önceki fonksiyonu belirtmemiz gerekir aksi takdirde ikinci fonksiyon çağırılan fonksiyonu ezecektir.



Eğer bu fonksiyona parameter atayacaksak (self sonrası) bu parametreleri ikinci parametrenin yalnızca \_\_init\_\_ kısmında tanımlamamız yeterli olacaktır. Bunun alternatifi olarak “super().\_\_init\_\_(fname, lname)” komutunu da kullanabiliriz. "self.firstName = fname” komutu ile alınacak komutu “fname”’e integre ederiz.





Printing işlemini manuel yapmak için “\_\_str\_\_(self): return f”{} …”” komutuyla print() fonksiyonun yaptığı işi tanımlayabiliriz. Bunun yanında length belirtmek için “\_\_len\_\_(self): return self.duration” komutunu kullabiliriz. Bunun gibi birçok özel metodu bulunmaktadır. [**Kaynak**](https://www.informit.com/articles/article.aspx?p=453682&seqNum=6)

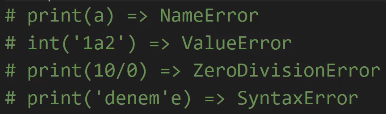
Kodumuza uyguladığımız import ile ilgili bilgi almak istersek “help(library)” komutu ile library nasıl kullanılır görebilirz, bu libray’nin neleri kapsadığını genel hatlarıyla öğrenmek istersek de “dir(library)” komutu işimize yarayacaktır. Library’nin belirli bir fonksiyonuyla ilgili bilgi almak istersek “help(lib.some)” (some belirli parametre) komutuyla bireysel olarak görüntüleyebiliriz. “import some” komutunu kullanırsak gerekli işlemleri yapıyorken some.something olarak yazılması gerektiği unutulmamalıdır. “import lib as some” komutuyla lib adındaki kütüphaneyi some adıyla kullanabiliriz.

Library kurmak için daha kullanışlı olan yöntem ise “from some import \*” komutudur. Bu sayede önceki yöntem gibi some.something olarak parametreyi ayarlamak yerine direkt something komutu işimizi görecektir (\* her şeyi import etmemizi sağlar, bunun yerine spesifik bir fonksiyonu import edebiliriz, birden fazla fonksiyon için virgül ile ayırırız.)

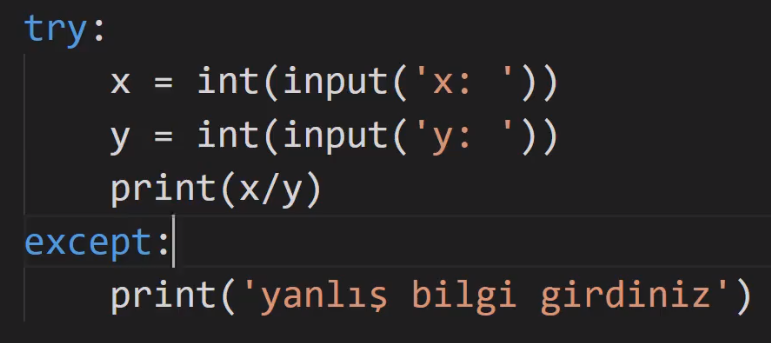
“import random” modülü rastgele sayılar oluşturmamıza olanak tanır. “random.uniform(x,y)” komutu ile x ve y sayıları arasında değerler oluşturabiliriz, bu fonksiyon int gibi komutlarla kullanıma uygundur (random.randint(x,y)” aynı işi yapacaktır). Bir liste oluşturduğumuzu ve bu listeden rastgele değer seçeceğimizi düşünelim, bu durumda “random.choice(list)” komutu oldukça işlevlidir. Bunu string üzerinden yaparsak sistem karşımıza rastgele harf yazdırır. “random.shuffle(list)” metodu listedeki (veya başka bir type) elemanların yerlerini her seferinde değiştirir, “random.sample(list, 3)” metodu ise listemizden rastgele 3 tane değer çekilmesini sağlar.

Hazır modülleri kullanabileceğimiz gibi kendi modüllerimizi de oluşturabiliriz. Dosya içerisine kodlama yapıldıktan sonra “import filename” komutuyla modül çağırılabilir. Bu modül pythonla beraber gelen built-in fonksiyonları gibi lib klasörüne atılırsa dosya yeri file path farketmeksizin istenildiği gibi çağırılabilir.

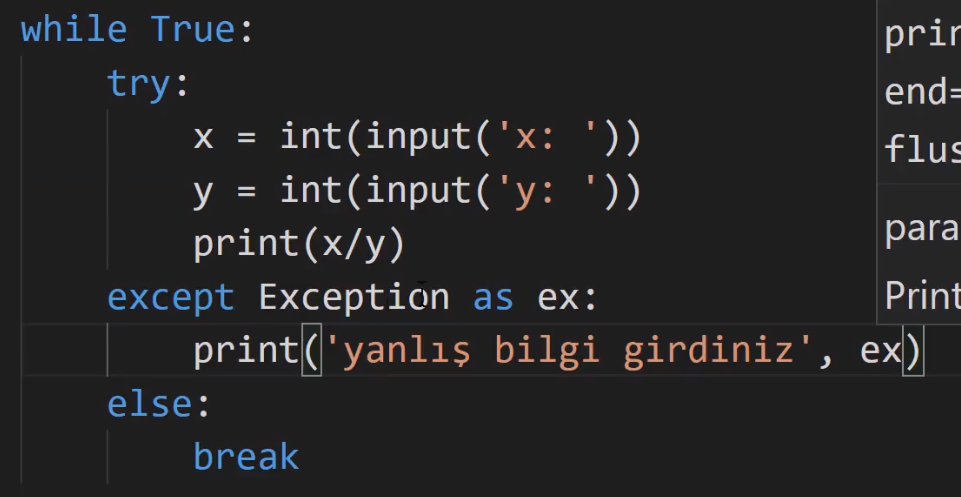
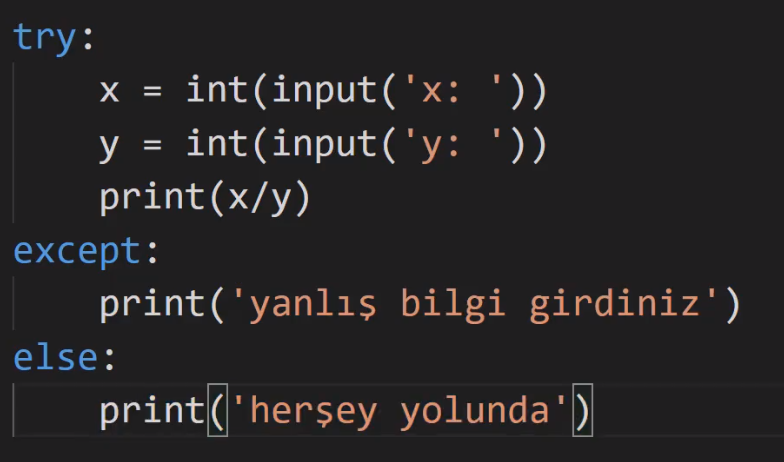
Python’da hataları bilmek de oldukça önemlidir. [**Kaynak**](https://docs.python.org/3/library/exceptions.html)



“try: except:” komutuna baktığımızda aşağıdaki örnekteki gibi bir kod bize error mesajı olarak verilecektir.

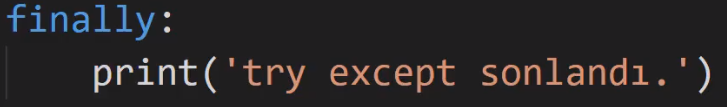


Fakat hatanın hangi türde olduğunu bu durumda tespit etmemiz mümkün değildir, bir başka örnek:



“try: except:” komutunu else ile örnekteki gibi destekleyebiliriz. Muhtemelen en mantıklı kodlama işlemi 2. Örnekte görebileceğimiz üzere “except Exception as some” komutudur, some kısmını printe entegre ettiğimiz takdirde hata türünü (eğer exception içerisindeyse, exception oldukça geniş bir base hata alanı) bize gösterecektir “yanlış bilgi girdiniz division by zero (vb)”. “try: except:” komutu aynı zamanda hatanın kod bozulmadan konsolda gösterilmesini sağlar (hataları kontrol yeteneği).

“finally:” komutu ne olursa olsun işlemin sonunda uygula anlamına gelmektedir.

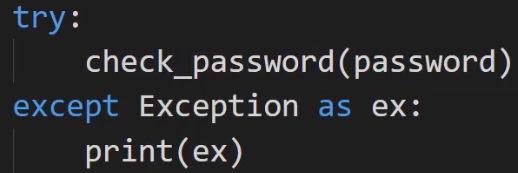
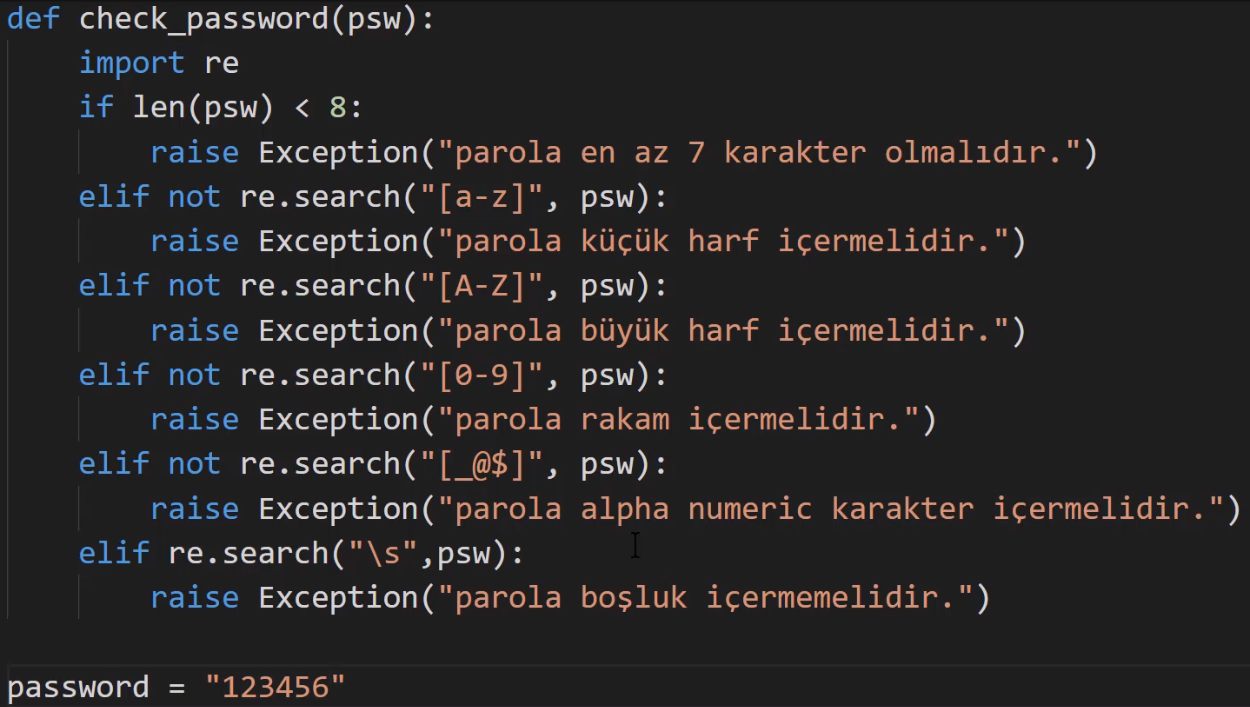


“while **True**” komutu ise sistem içeriden kendini kapatana kadar devam et anlamına gelmektedir. Eğer döngüyü kapatacak bir parametre bu aşamada kendini göstermez ise dögü sonsuza kadar devam eder.

“raise Exception(“something”)” komutu ile Exception’a ait hatalar aldığımız takdirde something stringini ekrana yazdırmamızı sağlar.

“elif not :” metodunu kullanarak eğer belirttiğimiz komut gerçekleşmemişse uygulanacak bir fonksiyon oluşturururuz.

“try: expect:” ve “raise Exception” komutlarına güzel bir örnek:



Yukarıdaki örnekte kurulan “import re” kütüphanesinden biraz bahsetmek gerekirse [**Kaynak**](https://docs.python.org/3/library/re.html) sitesine bakılabilir. Yukarıdaki örnekteki gibi lib.search() komutu ile kütüphanenin sahip olduğu komutları çağırarak kullanabiliriz, bu kütühane için “[a-z], [0-9]” ifadeleri “a’dan z ye kadar, sıfırdan dokuza kadar” anlamlarını taşır.

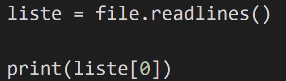
Dosya açmak veya oluşturmak için “open()” komutu kullanılır. Dosyayı bulunan path haricinde bir yerde oluşturmak için “C:/users/desktop” gibi alan adı belirterek yerleştirme yapabiliriz. “file.write(“something”) komutuyla dosya içerisine text yazdırabiliriz. Normal durumda oluşturulan dosya Türkçe karakterleri desteklemez, Türkçe karakterler ve daha nicesini dosyanın desteklemesi için “open()” komutunun sonuna “encoding = “utf-8”” komutu yazılır. “w” type ile oluşturulmuş dosya tekrardan çalıştırıldığında dosya içeriği silinip yeniden kurulur. “a” type ile oluşturulunca silinme olmaz, kod çalıştırıldıkça append edilen kod eklenmeye devam edilir. Eğer bir dosya oluşturup dosyaya ekleme yapmayacaksan “w” type kullanılır, bu komut yeni dosya oluşturur, öyle bir dosya varsa sistem error verir.

“open(“file.txt”) yazıp type belirtmezsek type “r” olarak atanır. Okunanlar printlenirken arada bir satır boşluk bırakılır, bunu engellemek için “print(i, end = “”)” komutunu kullanırız.





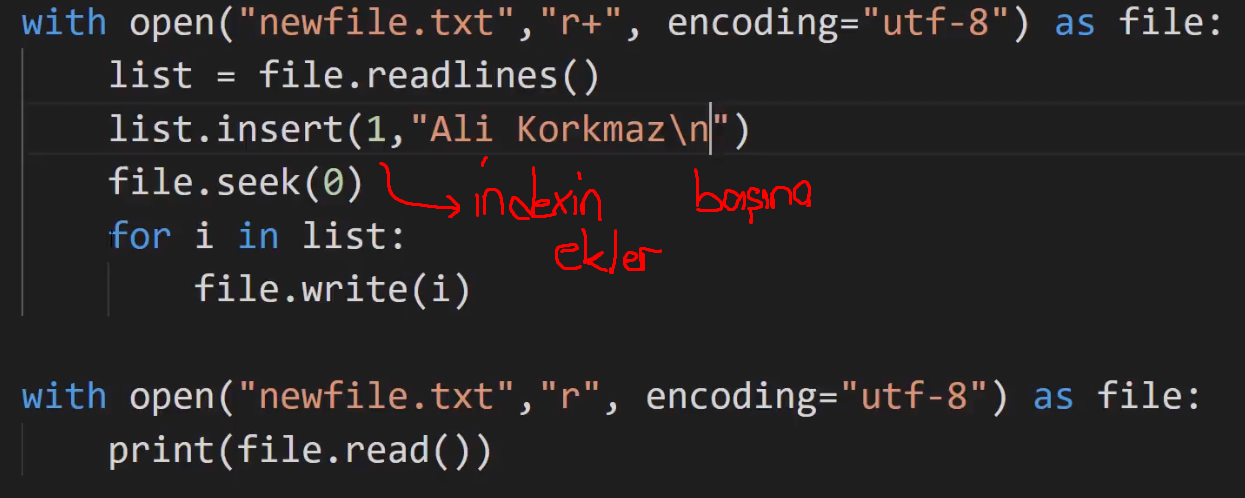
Dosya okuma işlemi yukarıda görüldüğü gibi iki farklı şekilde yapılabilir, ikinci yöntem çok daha pratiktir. Eğer .read(number) kısmında paranteze bir int girersek sistem ilk number karakteri okuyacaktır. “.readline()” komutuyla bir satırı okutabiliriz. “.readlines()” komutu ile dosya içerisindeki texti list şeklinde yazdırırız. Aşağıdaki örnekteki şekilde listeninilk değerini yazdırmış oluruz.



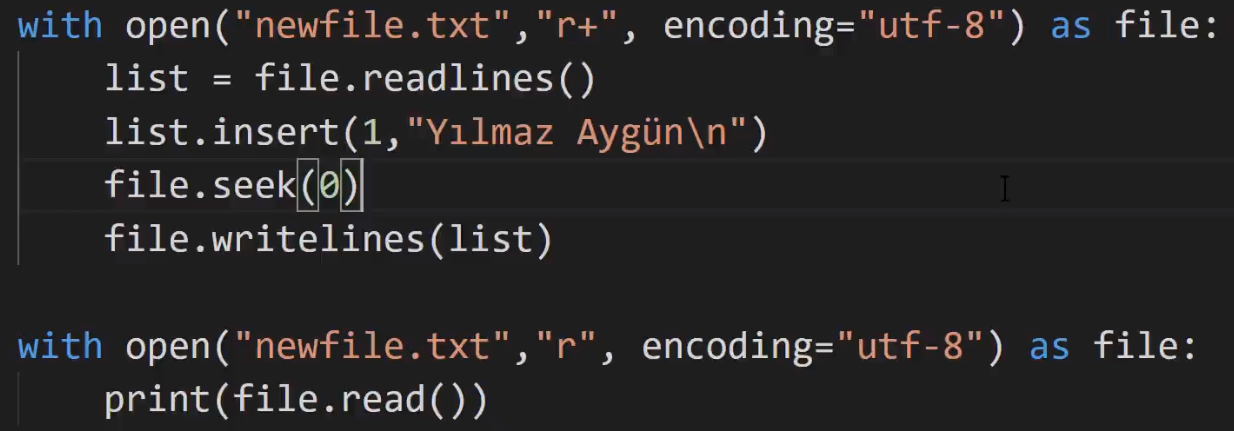
Okuma işleminin işleyiş şekli şudur; başlangıçta bulunan cursor bizim komuta göre ilerlemeye başlar ve bir daha geri aksi ifade edilmedikçe bir daha geri dönmez. Örneğin “.readline(10)” yaptığımızda texte 10 karakter var ise bu komutu tekrardan kullanmamız ekranda hiçbir şey görünmemesine yol açacaktır çünkü kodumuz texti zaten okumuştur.

“with open(…) as file:” komutunu kullanarak komutumuz kod bloğu halini alabilir, böylece “file.close()” komutunu kullanmamıza gerek kalmaz. “file.read()” komutuyla cursor’umuzun konumunu öğrenebiliriz, “file.seek(number)” komutu ile de cursor’umuzu number’ıncı konuma gönderebiliriz.

“r+” type’ı dosyayı güncelleme yapar ve yazma halinde cursor’ı en baştan başlatır, bunu ayarlamak için “.seek” komutunu kullanabiliriz. “a” type’ını kullanırsak ekleme işlemi yapar ve sayfanın en sonuna ekleme işlemini gerçekleştirir.



Dosyanın herhangi bir yerine kod eklemek için yukarıdaki metodlar kullanılabilir. “list.insert(index, “text”) ile texti insert eder ve ardından for dögüsü ile textimizi dosya içerisine yazdırabiliriz. Bunun yerine kullanabileceğimiz daha kullanışlı bir metod bulunur.

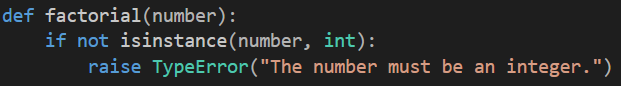


Sistem bir dosyayı okuyorken arada satır boşluğu bulunmamalıdır, aksi takdirde index 0 dan veri alınabiliyorken örneğin index 1 içerisinde herhangi bir parametre bulunmadığından değer okunamaz. Bunun önüne geçmek için boşluklar “param = param[:-1]” komutu ile kaldırılır. “list = line.split(“:”)” komutu ile line’ı string içerisindeki “:”’dan ikiye bölerek ayırma işlemini yaparız. Bu işlem, stringden değer çekmek için oldukça elzemdir.

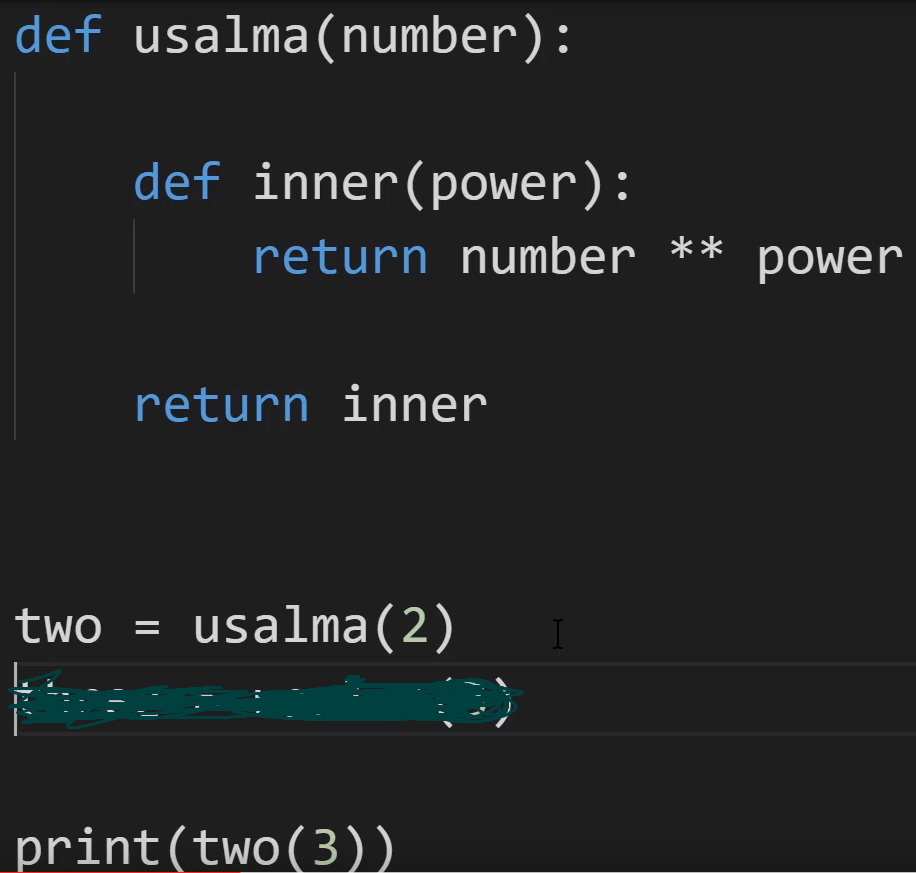
def fonksiyonu içerisine çağırmış olduğumuz başka bir def fonksiyonununu main scope’da çağıramayız çünkü belirli def fonksiyonu yalnızca birincil def fonksiyonu içerisinde tanımlı durumdadır.



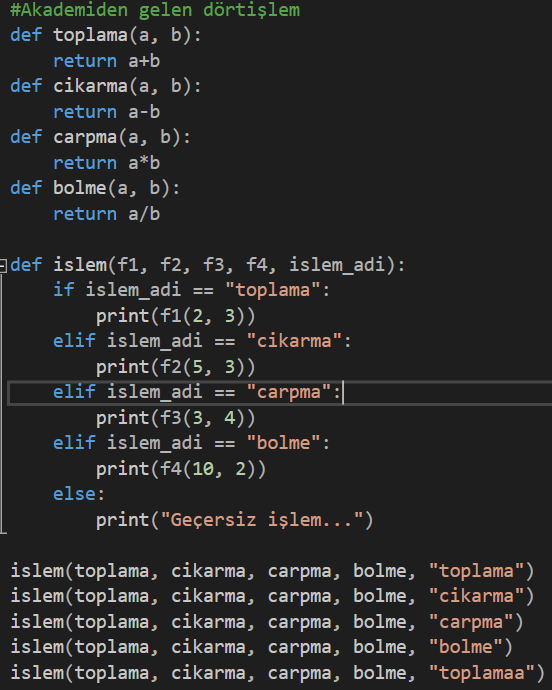
“isinstance(object, classinfo)” komutu ile objektimizin type’ının ikincil parametreden olup olmadığını kontrol ettiririz. Örneğin eğer girilen sayı bir integer değilse sistemimizin error vermesini istiyoruz, örnekteki gibi bir yapı kurarak bunu sağlayabiliriz.



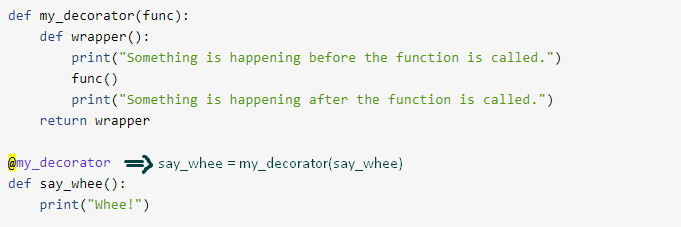
İçsel def fonksiyonunu tanımlamanın bir yolu örnekteki gibidir. Ana fonksiyon çağırılarak bir parametreye eşitlenir. Bu parametre ise “print(par)” olarak yazdırılır, “par” bu örnekte ifadesi içsel fonksiyondaki power kelimesine denk düşmektedir.



Bu örnekte görüldüğü gibi def fonksiyonları bir parametreye eşitleyerek kodlama yapabiliriz.

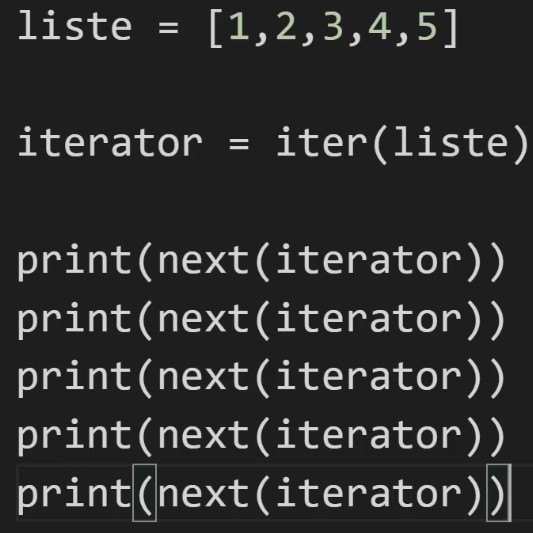


Time kütüphanesi “import time” komutu ile koda eklenir. “.ctime()” komutu ile (current time) şimdiki tarihi görüntüleyebiliriz. “.sleep(time)” komutu ile “time” içerisine yazdığımız saniye boyunca sistemin çalışmasını geciktiririz. “.time” ile sistem run edilmesinden ne kadar süre geçti öğrenebiliriz. Ayrıntı için [**Kaynak**](https://www.programiz.com/python-programming/time)**.**

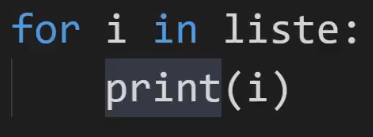


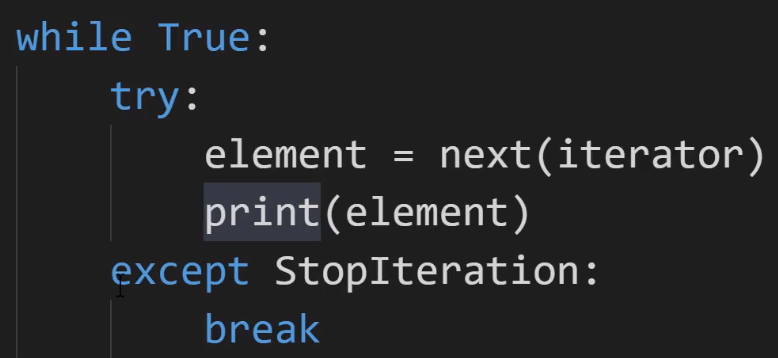
Bir objenin for döngüsünde döndürülebilmesi için kullanılan parametrenin iterable olması gerekir, (list gibi). List, iterable bir değerdir fakat iterator değildir. Bir fonksiyonun iterable olabilmesi için döngülendirilebilir olması şarttır. “print(dir(param))” komutu ile parametrenin iterable olup olmadığını öğrenebiliriz. Bir fonskiyonun iterator olabilmesi için “\_\_iter\_\_” ve “\_\_next\_\_” yapısallarını içermesi gerekir, bu fonksiyonlar sayesinde iterator bulunduğu konumu ve ne yapması gerektiğini bilir. Örneğin print(param) komutunu 3 defa yazdırdık, 4. yazdırışımızda sistem değerin 3. indexte olduğunu bilerek devam edecektir (0, 1, 2, 3).

Oluşturduğumuz bir listi veya başka bir type’ı (eğer uygunsa) “iter(list)” komutuyla iteratore dönüştürebiliriz.

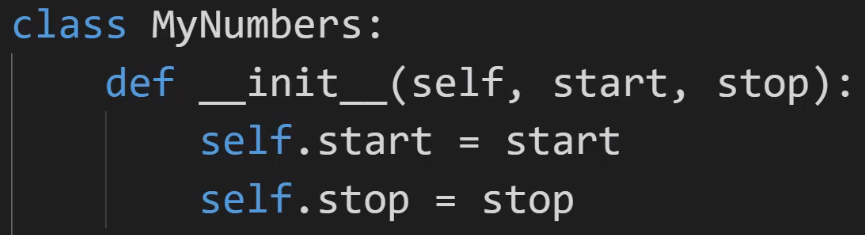


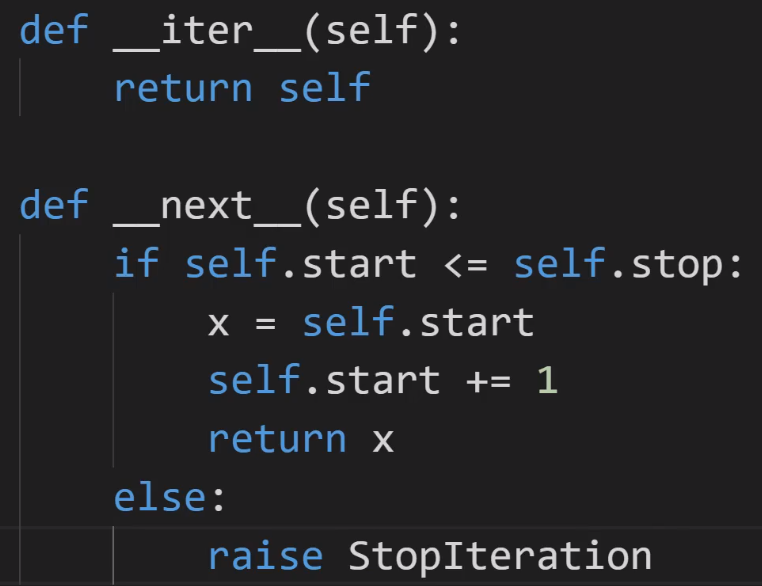
Aşağıdaki örnekte aynı örneğin for ile ifade edilişi ve bu for döngüsünün program içerisinde nasıl oluşturulduğunun kodu gösterilmektedir.

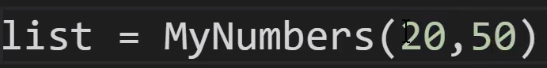




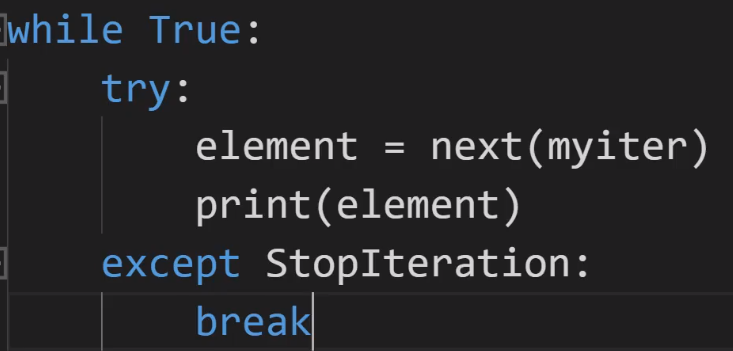
Bu bağlamda for döngüsünün yaptığı işi tamamen manuel ve kontrolün bizde olduğu şekilde yapmak istersek, örnek:



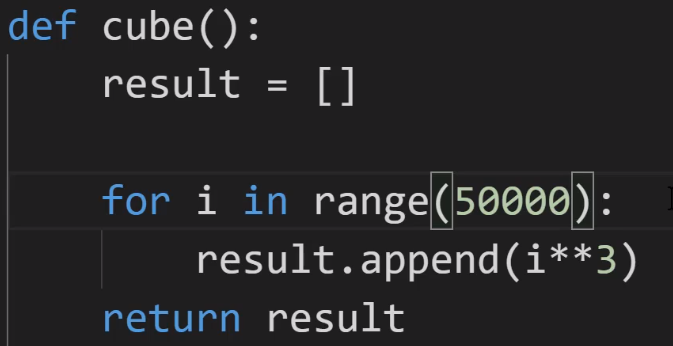


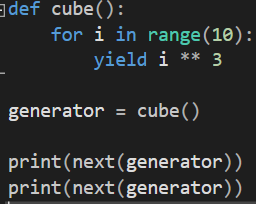




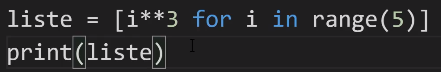


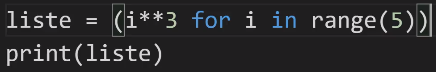
“yield param” komutu işlenen değerlerin bellekte tutulmadan hesaplanmasını sağlar, bu sayede işlemler bellekte yer kaplamaz fakat istenilen değerler işlenilerek print edilebilir. Aşağıdaki örnekte örneğin 50000 farklı değere sahip olan bir fonksiyonu işlemek çok zaman alırdı.





İkinci fonksiyonda range’in çok yüksek bir sayı almasının sistemde kaplayacağı yer açısından bir farkı olmaz çünkü kullanılan değerler kadar işlem yapılır. Bu komut dizinin **generator** olarak adlandırılır, kullanılma amacı ise eğer oluşturulan değerleri liste içerisinde saklamak gerekmiyorsa, elemanları o adna kullanıp daha kullanmayacaksak uygulamada kullanmak mantıklı olur. Comprehesive metodunu generator metodu ile birleştirbiliriz, bu işlemde normal comprehensivle aradaki tek fark liste olarak ifade edilen parametrenin köşeli parantezleri yerine normal parantez kullanıyor olmamızdır.



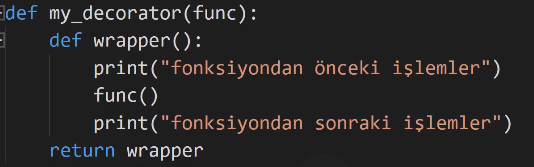


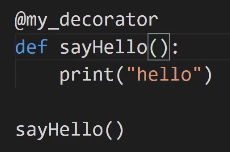
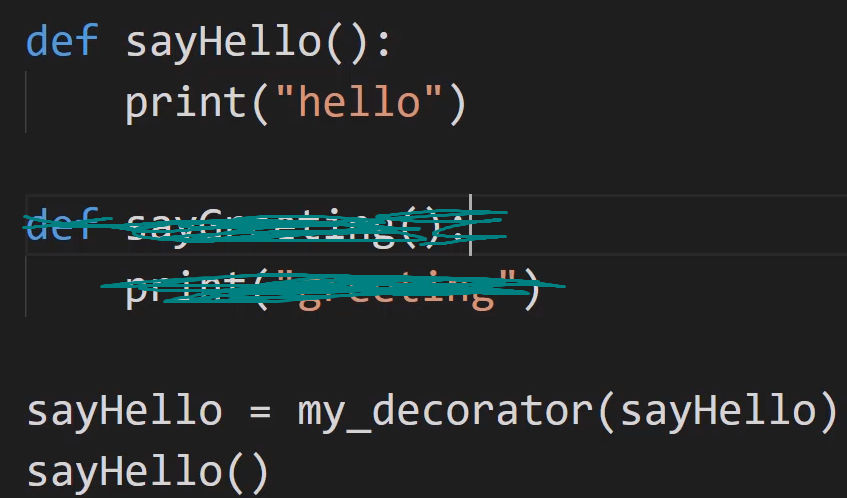
Tabiki ikinci ifadede “print(liste)” komutu yalnızca object ile ilgili bilgileri söyleyecektir, ilk örneğin yaptığını ikince yapabilmek için aşağıdaki eklentiyi ihtiyacımız olur.



“dir(param)” komutu aynı zamanda bir parametrenin sahip olduğu tüm yapısalları gösterir.

Decorator olarak adlandırılan “@functionName” formundaki yapılar bir def fonksiyonunun içerisinde parametre taşıdığı basit şekilde yazılmış haline denir. Aşağıdaki 2. ve 3. fotoğrafın işlevleri aynıdır.





Bu örnekte “sayHello()” komutunun içerisine bir parametre yazmak istersek o parametreyi “wrapper()” ve “func()” içerisine de girmemiz gerekir.

Ctrl + K, Ctrl + K ile istediğimiz satıra bookmark atayabiliriz.