****

**SAKARYA ÜNİVERSİTESİ**

**BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ**

**BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ TASARIMI**

**Python for Data Analysis 1.Rapor**

**B161200304**

**MUSTAFA ACAR**

**SAKARYA 2019**

**İÇİNDEKİLER**

1. BÖLÜM 2: Python Dilinin Temelleri, IPython ve Jupyter Notebook 1

1.1. Python Yorumlayıcısı 1

1.2. IPython Temelleri 2

1.3. Python Dilinin Temelleri 9

2. BÖLÜM 3: Yerleşik Veri Yapıları, Fonksiyonlar ve Dosyalar 29

2.1. Veri Yapıları ve Diziler 29

2.2. Fonksiyonlar 45

2.3. Dosyalar 53

3. BÖLÜM 4: NumPy Temelleri 56

3.1. NumPy ndarray 56

3.2. Evrensel Fonksiyonlar 66

3.3. Dizi Yöntemli Programlama 69

3.4. Dizilerle Dosya Giriş ve Çıkış 73

3.5. Doğrusal Cebir 74

3.6. Sözde Sayı Üretimi 75

3.7. Örnek 76

4. BÖLÜM 5: Pandas Temelleri 78

4.1. Pandas Veri Yapıları 78

4.2. Temel Fonksiyonlar 86

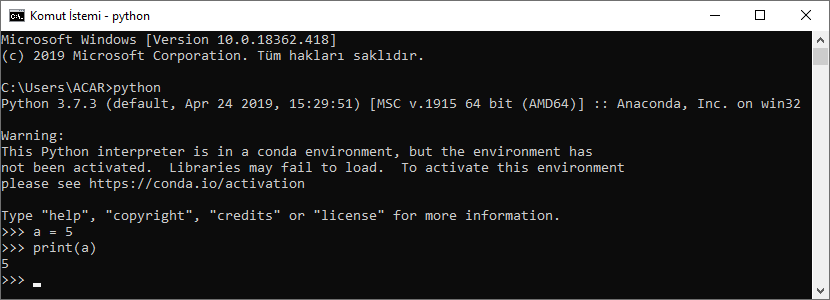
4.3. Tanımlayıcı İstatistiklerin Özetlenmesi ve Hesaplanması 102

**BÖLÜM 2**

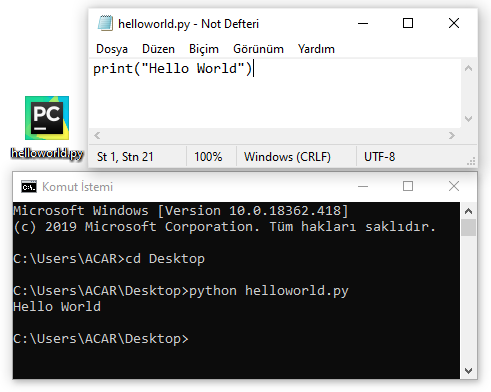
**Python Dilinin Temelleri, IPython ve Jüpyter Not Defteri**

**2.1 Python Yorumlayıcısı**

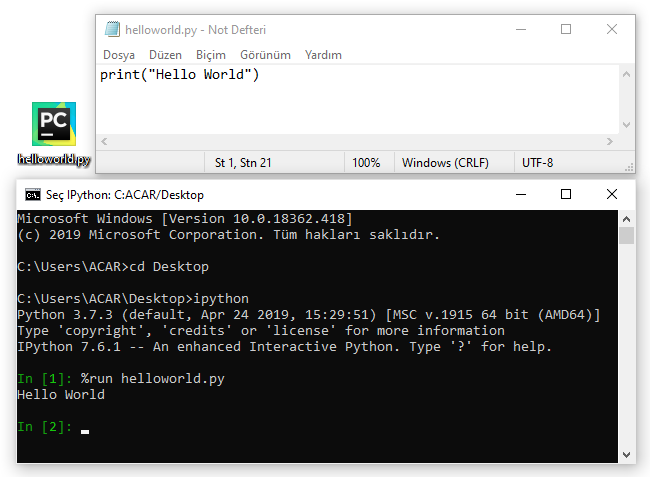
Python yorumlanmış bir dildir. Python yorumlayıcısı, kaynak kodu komut komut okuyup üzerinde çalışılan makinenin komut setine çevirerek çalıştıran bir programdır. Standart etkileşimli Python yorumlayıcısı “python” komutuyla komut satırından çağrılabilir:



Python programlarını komut istemi ile çalıştırmak kolaydır. Diyelim ki biz “helloworld.py” adında bir python dosyası oluşturduk. Aşağıdaki komutu çalıştırdığımızda kodlar yorumlanır. (“helloworld.py” dosyası geçerli çalışma terminal dizininde olmalıdır):



Bazı Python programcıları tüm Python kodlarını bu şekilde yürütürken, veri analizi veya bilimsel bilgi işlem yapanlar, gelişmiş bir Python yorumlayıcısı olan IPython veya Jupyter Notebook'larını kullanır. %run komutunu kullandığınızda, IPython’da aynı işlemi %run komutu ile yaparız.

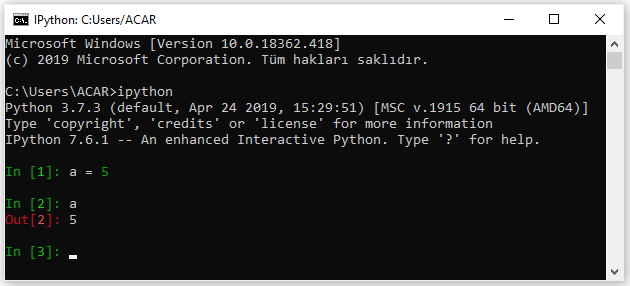


Varsayılan IPython istemi, standart “>>>” komut istemiyle karşılaştırıldığında [2]: stilindeki numaralandırmayı benimser.

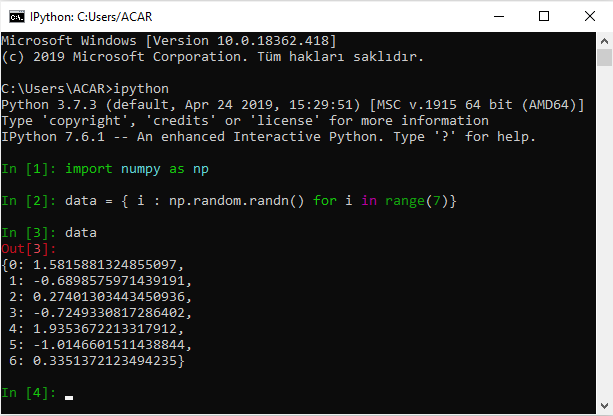
**2.2 IPython Temelleri**

**IPython Kabuğunu Çalıştırma**

IPython kabuğunu, Python yorumlayıcısının başlatılması gibi komut satırından başlatabiliriz:

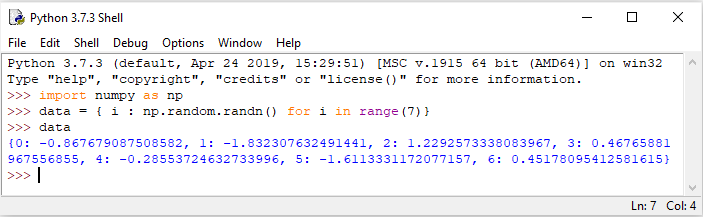


Birçok Python IDE türü, normal baskıdan farklı, daha okunaklı veya güzel yazdırılacak şekilde biçimlendirilmiştir.



İlk iki satır Python kodu ifadeleridir; ikinci ifade veri adında bir değişken oluşturur. Son satır veri değişkeninin değerini yazdırır.

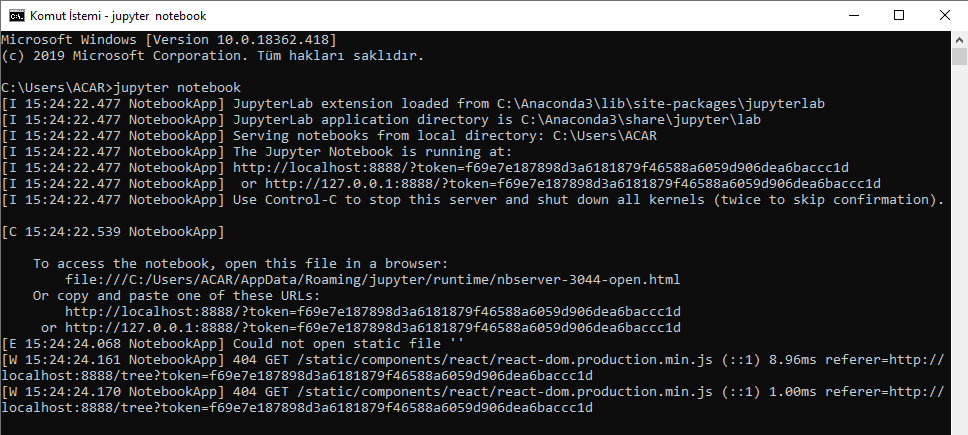
Yukarıdaki veri değişkenini standart Python yorumlayıcısında yazdırdıysanız, okunabilirliği daha az olurdu:



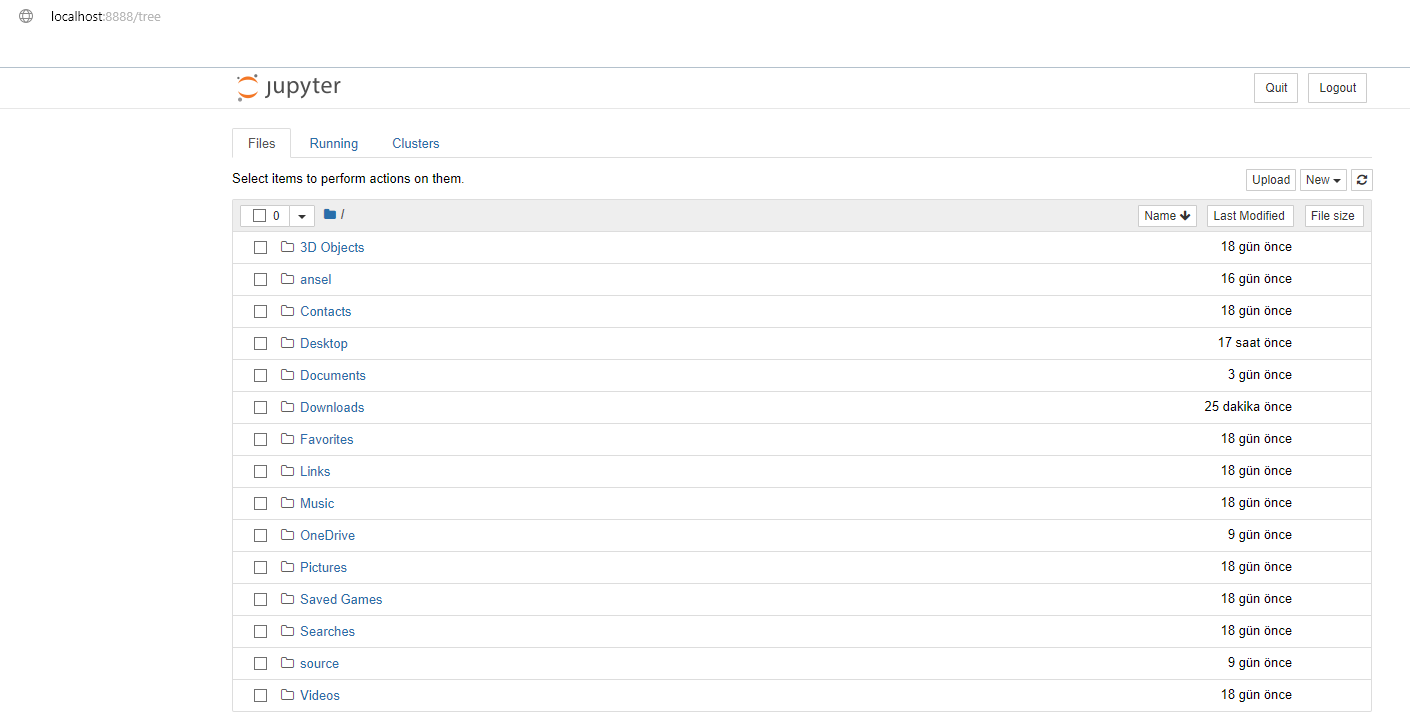
**Jupyter Notebook’u Çalıştırma**

Jupyter Notebook, çeşitli programlama dilleri için etkileşimli bir ortam sağlayan açık kaynak kodlu bir programdır. Bir web tarayıcısı üzerinden kodları düzenlemeyi ve çalıştırmayı sağlayan bir belge türüdür. Jupyter, IPython çekirdeğini kullanır.

Jupyter'i başlatmak için bir terminalde “jupyter notebook” komutu çalıştırılır:

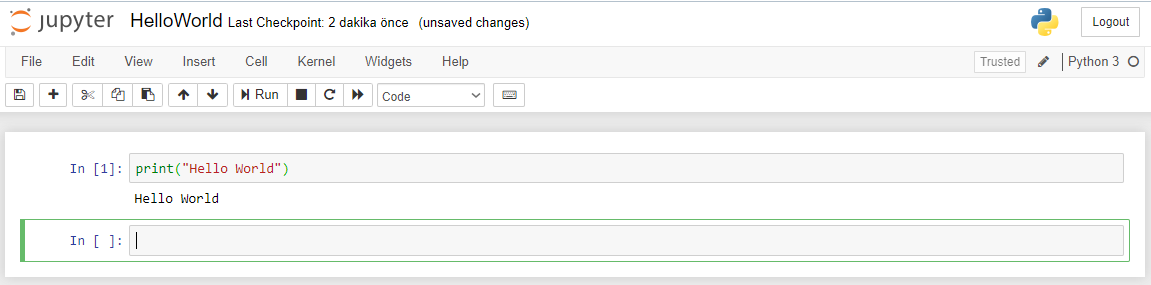


Birçok platformda, Jupyter otomatik olarak varsayılan web tarayıcınızda açılır.



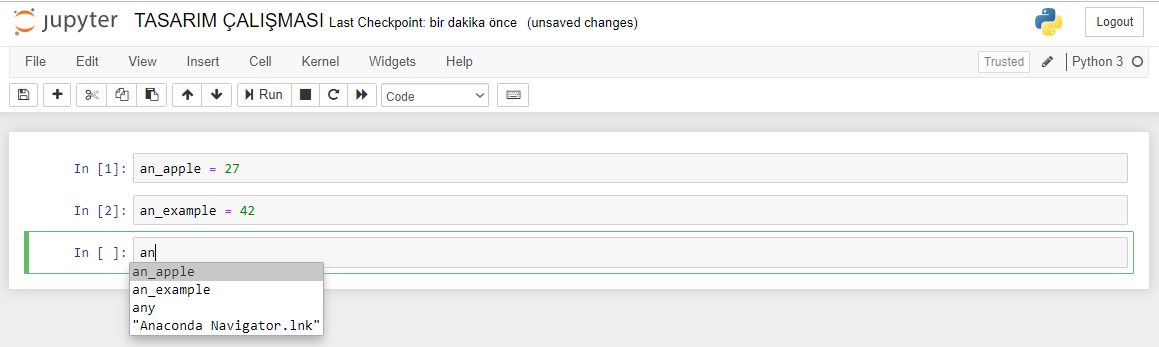
Şekil 2.1. Jupyter notebook açılış sayfası

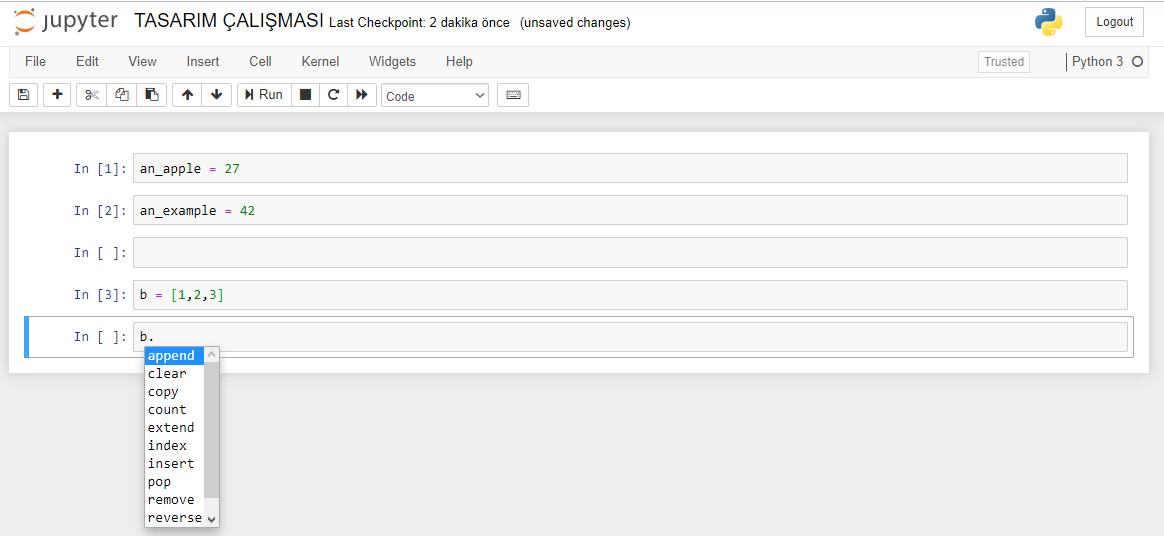
Yeni bir not defteri oluşturmak için “New” düğmesine tıklayıp, "Python 3" seçeneğini seçeriz. Boş hücreye Python kod satırını girip, çalıştırmak için Shift-Enter tuşuna basarız.

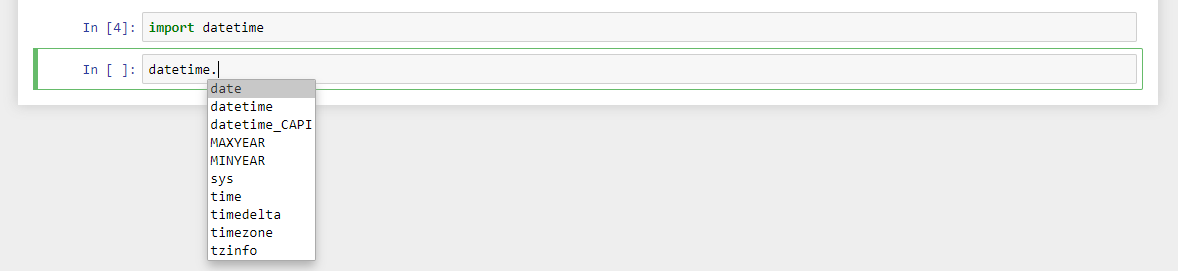


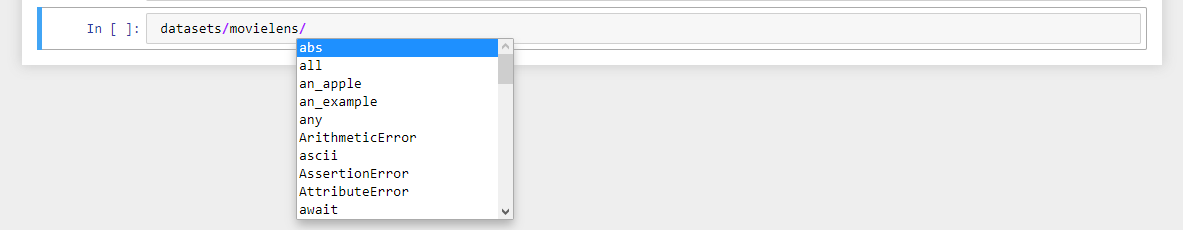
**Sekme Tamamlama**

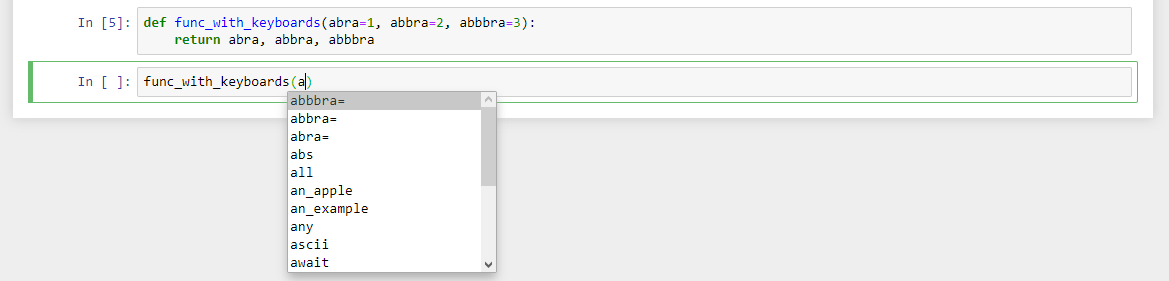
Kod bloğuna ifadeler girerken, “Tab” tuşuna basmak şu ana kadar yazdığımız karakterlerle eşleşen tüm değişkenleri (nesneler, fonksiyonlar vb.) arar:





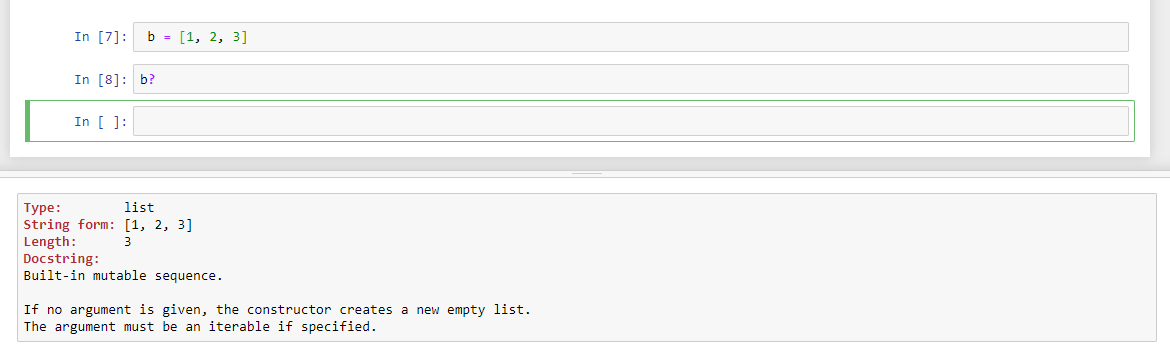


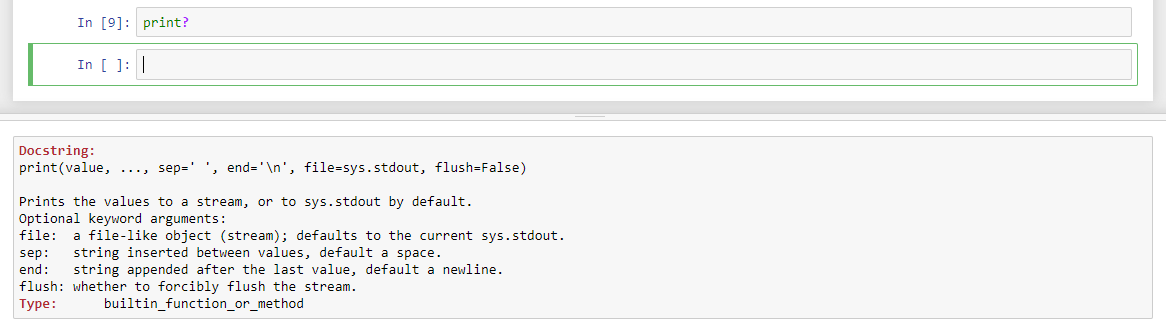




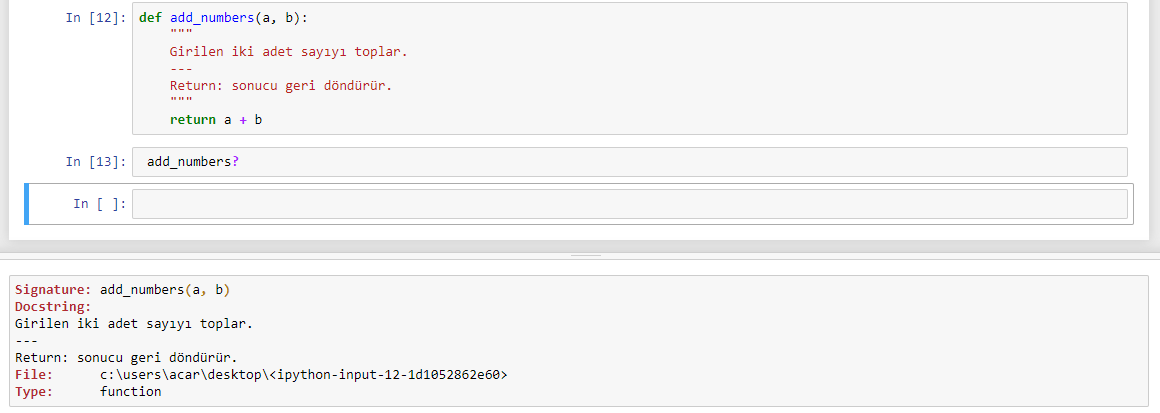
**İç Gözlem**

Bir değişkenden önce veya sonra bir soru işareti (?) kullanmak, nesne hakkında bazı genel bilgileri gösterir:





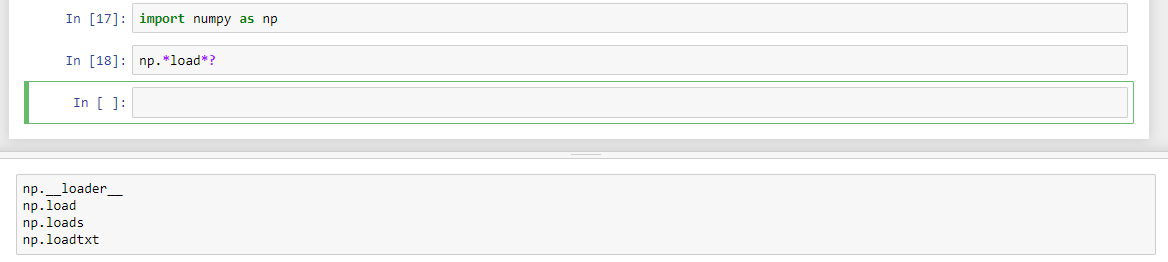
Eğer kod içerisinde tanımlanmış “docstring” varsa bu bilgilerde gösterilir:



Bir değişkenden önce veya sonra iki soru işareti (??) kullanmak, fonksiyonun kaynak kodunu da gösterir:

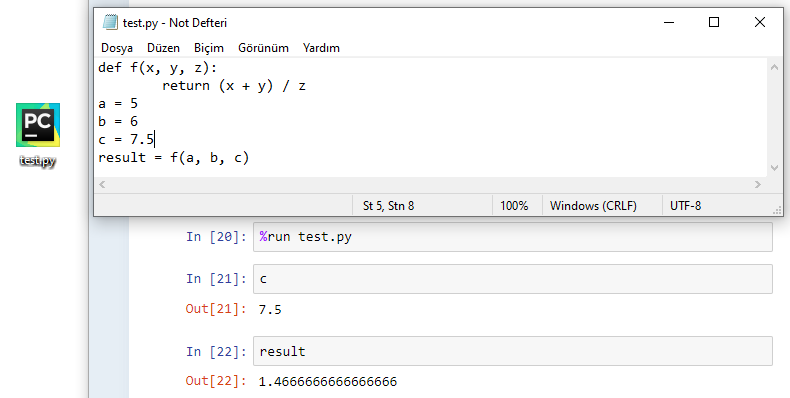


Soru işaretinin (?) son bir kullanımı vardır. Joker karakterle (\*) birlikte bir dizi karakter ifadesine uyan tüm adları gösterir. Örneğin, NumPy kütüphanesinde “load” alanındaki tüm işlevlerin bir listesini alabiliriz:

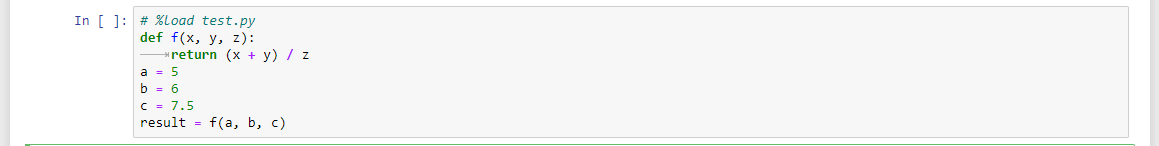


**%run Komutu**

%run komutunu kullanarak herhangi bir dosyayı Jupyter ile çalıştırabiliriz:



%load ile bir dosyayı bir kod hücresine aktarabiliriz:

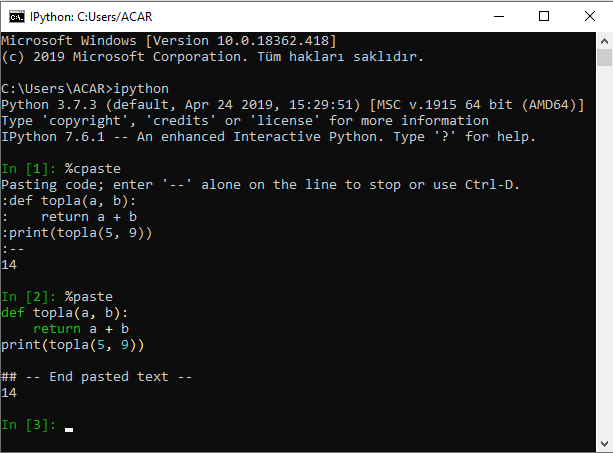


**Panodan Kod Yürütme**

Jupyter notebook üzerinde kodu herhangi bir kod hücresine kopyalayıp yapıştırabilir ve çalıştırabilirsiniz. Kodu IPython kabuğundaki panodan çalıştırmak da mümkündür. Başka bir uygulamada aşağıdaki kodu kopyaladığımızı varsayalım:

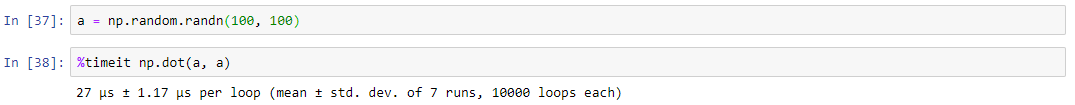


Kopyaladığımız kodu IPython kabuğunda “%paste” veya “%cpaste” komutları ile çalıştırabiliriz:

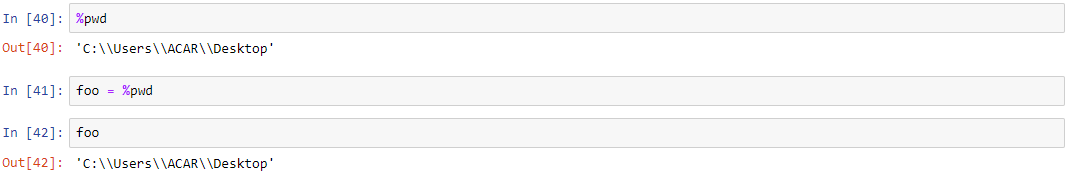


**Sihirli Komutlar**

IPython’un özel (Python’un içine yerleştirilmemiştir) “sihirli” komutları olarak bilinir. Bunlar ortak görevleri kolaylaştırmak ve IPython sisteminin davranışını kolayca kontrol etmenizi sağlamak için tasarlanmıştır. Bir sihirli komut, yüzde sembolü (%) ön eki eklenmiş herhangi bir komuttur. Örneğin, %timeit işlevini kullanarak, matris çarpımı gibi herhangi bir Python ifadesinin yürütme süresini kontrol edebiliriz:



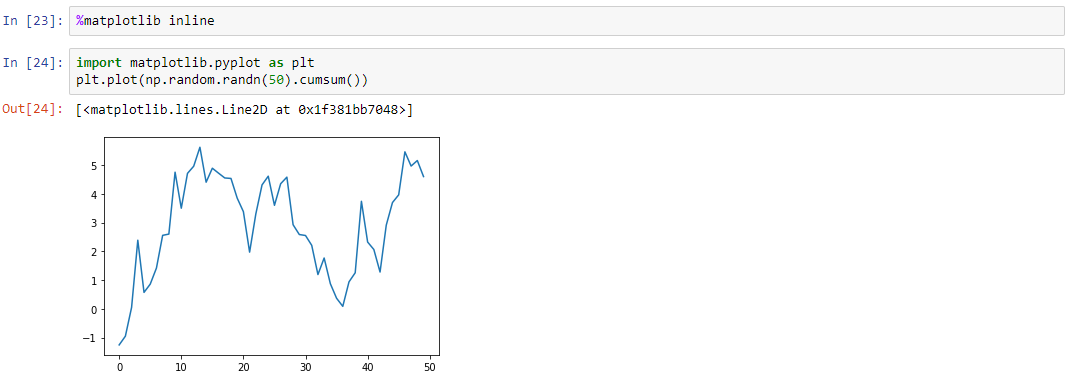
Bazı sihirli işlevlerin çıktıları bir değişkene atanabilir:



%magic komutu ile tüm sihirli işlevlere ve içeriklerine ulaşabiliriz.

**Matplotlib Entegrasyonu**

IPython’un analitik hesaplamadaki popülerliğinin bir nedeni, veri görselleştirmesi ve matplotlib gibi diğer kullanıcı ara yüzü kütüphaneleri ile iyi bütünleşmesidir.



**2.3 Python Dilinin Temelleri**

Python dili tasarımı okunabilirlik, basitlik ve açıklık vurgusuyla ayırt edilir. Bazı insanlar “çalıştırılabilir sözde kod” olarak nitelendirilebilecek kadar ileri gider.

Python, R, C ++, Java ve Perl gibi birçok dilde olduğu gibi parantez kullanmak yerine, kodu yapılandırmak için boşluk kullanır. İki nokta üst üste girintili bir kod bloğunun başlangıcını gösterir, bundan sonra tüm kodun bloğun sonuna kadar aynı miktarda girintili olması gerekir.

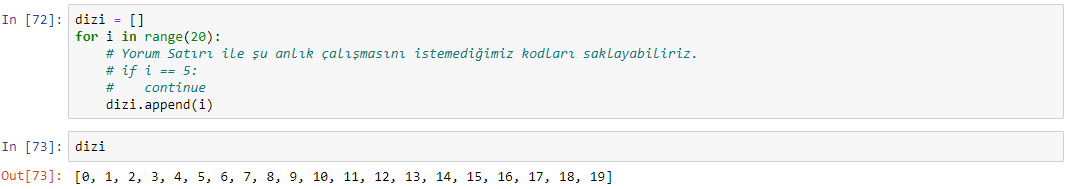


Python ifadelerinin de noktalı virgüllerle sonlandırılmasına gerek yoktur. Noktalı virgül, ancak tek bir satırda birden çok deyimleri ayırmak için kullanılabilir:



**Yorum Satırı**

Kod bloğumuza yorum satırı eklemek için # işaretini kullanırız. Python yorumlayıcı #işaretinden sonra yazılanları yok sayar. Kodumuzda bir kod satırının çalışmasını istemediğimizde de bu yöntemi kullanabiliriz.





**Fonksiyon ve Nesne Metotlarının Çağrımı**



**Değişkenler ve Argüman Geçişi**

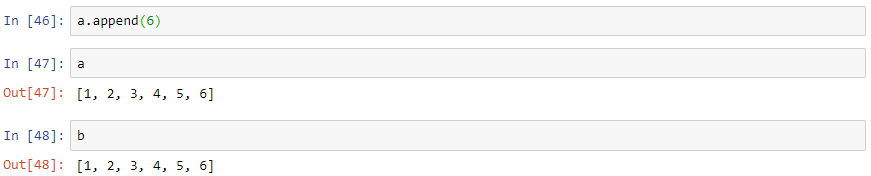
Python'da bir değişken tanımladığımızda, eşittir işaretinin sağ tarafındaki nesneye referans oluştururuz.



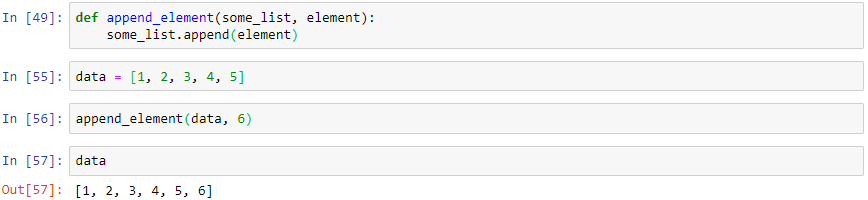
Yeni bir değişkene a’yı atayalım:



Bazı dillerde bu atama, verinin [1, 2, 3, 4, 5] kopyalanmasına neden olur. Python'da a ve b aslında aynı nesneyi temsil eder. Bunu a öğesine bir öğe ekleyerek ve sonra b'nin öğelerini inceleyerek bakabiliriz.



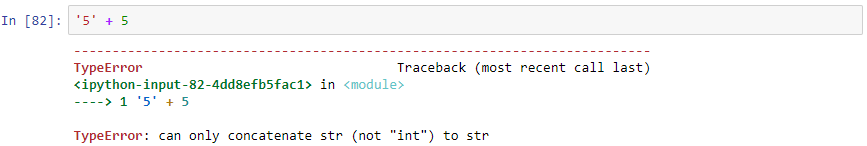
Nesneleri bir fonksiyona argüman olarak ilettiğimizde, orijinal nesnelere kopyalama yapmadan başvuruda bulunan yeni yerel değişkenler oluşturulur. Yeni bir nesneyi bir fonksiyon içindeki bir değişkene bağlarsanız, bu değişiklik ana kapsama yansıtılmaz. Bu nedenle değişken bir argümanın içindekileri değiştirmek mümkündür.



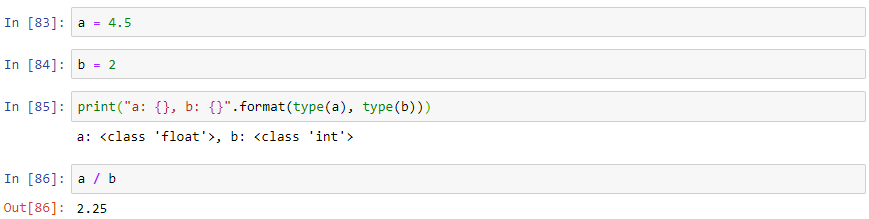
**Dinamik Referanslar, Güçlü Türler**

Python'daki nesne referanslarının kendisiyle ilişkilendirilmiş veri tipi yoktur. Örnek:

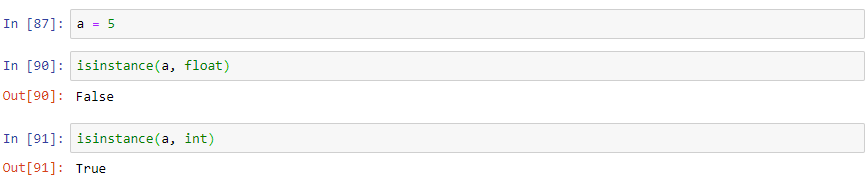




Visual Basic gibi bazı dillerde, ‘5’ dizesi bir tam sayıya dönüştürülebilir böylece 10 elde edilebilir. JavaScript gibi diğer dillerde, tamsayı ‘5’ bir dizeye dönüştürülür ve birleştirilmiş dize ‘55’ elde edilebilir. Bu bağlamda Python, güçlü bir dil olarak kabul edilir; bu her nesnenin belirli bir türe (veya sınıfa) sahip olduğu anlamına gelir ve örtülü dönüşümler yalnızca aşağıdaki gibi belirli durumlarda gerçekleşir:



Bir nesnenin türünü bilmek. Bir nesnenin “isinstance” işlevini kullanarak belirli bir türün bir örneği olup olmadığını kontrol edebiliriz:

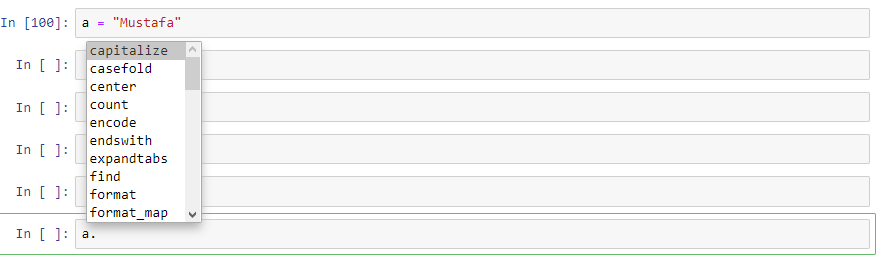


“isinstance” fonksiyonu, bir nesnenin türünün demet içerisinde bulunan veri tipleri arasında olup olmadığını kontrol etmek istiyorsak bu şekilde de kullanılabiliriz:



**Metotlar ve Öznitelikler**

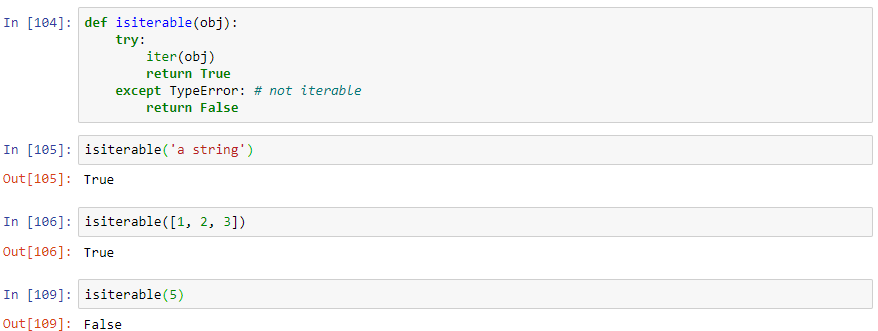
Python'daki nesneler tipik olarak hem niteliklere (nesnenin içinde depolanmış diğer Python nesneleri) hem de yöntemlere (nesnenin dahili verilerine erişebilecek bir nesneyle ilişkili fonksiyonlara) sahiptir. Her ikisine de, object.nitelik\_ismi sözdizimi üzerinden erişilir:



Öznitelikler ve metotlar, “getattr” fonksiyonu aracılığıyla ada göre de erişilebilir:

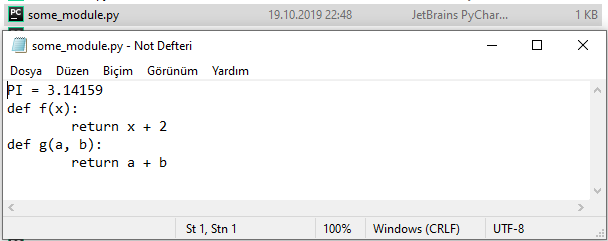


**Duck Typing**



**Modül Import Etme**

Python'da bir modül, içerisinde sadece Python kodu içeren ‘.py’ uzantısına sahip bir dosyadır. Aşağıdaki modülün olduğunu varsayalım:



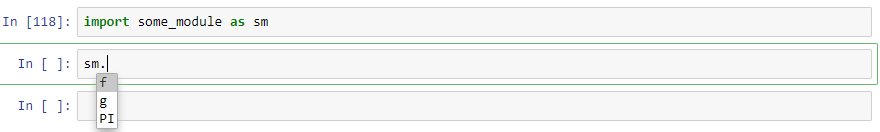
“some\_module.py” içinde tanımlanan değişkenlere ve fonksiyonlara erişmek istersek, kullandığımız yorumlayıcıya import etmemiz gerekir:

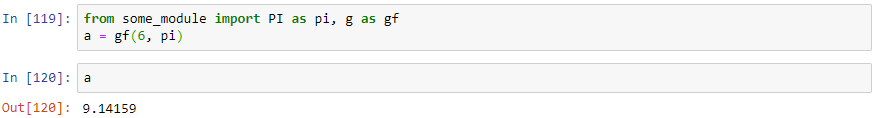


Veya modül içerisinde istediğimiz değişken ve fonksiyonları dahil edebiliriz:



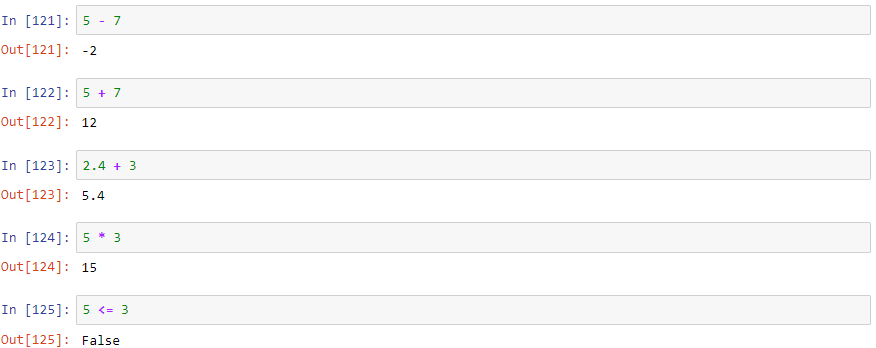
“as” anahtar sözcüğünü kullanarak farklı değişken adları içe aktarabilirsiniz:



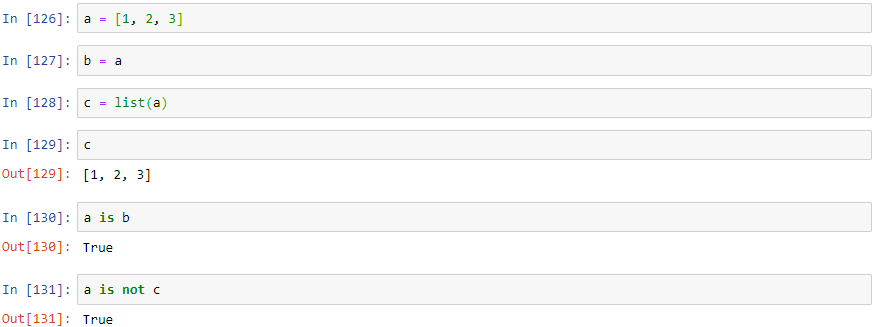


**Operatörler**

İkili matematik işlemleri ve karşılaştırmalarına örnek:



İki referansın aynı nesneyi gösterip göstermediğini kontrol etmek için “is” anahtar sözcüğü kullanılır. İki nesnenin aynı olmadığını kontrol etmek istiyorsak “is not” anahtar sözcüğünü kullanırız:



“list()” fonksiyonu her zaman yeni bir liste oluşturur. Yukarıdaki örnekte list() fonksiyonu ile a’nın bir kopyasını c’ye atamış olduk. Bu sebeple a ve c’nin aynı nesneyi göstermediğinden farklı olduğunu anlayabiliriz.

“is” ve “==” operatörü birbirinden farklıdır. “==” operatörü ile nesnelerin tipleri kontrol edilir:



“is” ve “is not” ile bir değişkenin None (içerisinde değer bulunmayan) olup olmadığını kolayca öğrenebiliriz:

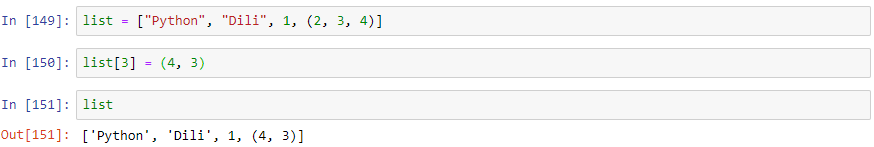


*Tablo: İkili Operatörler*

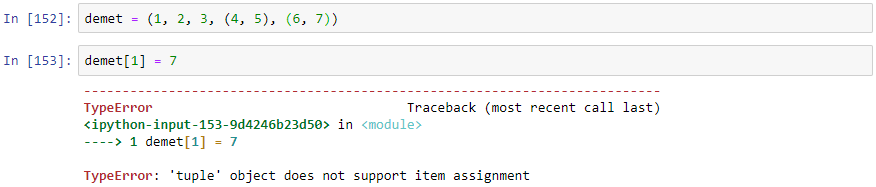
|  |  |
| --- | --- |
| İşlem | Açıklama |
| a + b | b’yi a’ya ekle |
| a – b | b’yi a’dan çıkar |
| a \* b | a’yı b ile çarp |
| a / b | a’yı b’ye böl |
| a // b | a’yı b’ye böl, kesirli alanlar atılır |
| a \*\* b | a üssü b olarak hesaplar |
| a & b | a ve b’nin true olmasıdır. Tamsayılarda “and” kullanılır |
| a | b | a veya b’nin true olmasıdır. Tamsayılarda “or” kullanılır |
| a ^ b | xor özel komutudur |
| a == b | a eşittir b’ye |
| a != b | a eşit değildir b’ye |
| a <= b, a < b | a küçük eşittir b’ye, a küçüktür b’den |
| a >= b, a > b | a büyük eşittir b’ye, a büyüktür b’den |
| a is b | a ve b aynı nesneye başvurursa sonuç true olur |
| a is not b | a ve b farklı nesneye başvurursa sonuç true olur |

**Değiştirilebilir ve Değiştirilemez Nesneler**

Python'da listeler, sözlükler, NumPy dizileri ve çoğu kullanıcı tanımlı tür (sınıf) gibi nesnelerin geneli değişkendir. Bu, içerdikleri nesnenin veya değerlerin değiştirilebileceği anlamına gelir:



Demetler içyapısını korur ve bu yüzden değiştirilemezdir.



**Skaler Veri Tipleri**

*Tablo: Standart Skaler Veri Tipleri*

|  |  |
| --- | --- |
| Tip | Açıklama |
| None | “null” değeri |
| str | Metinsel verileri tutar |
| bytes | Değeri 0 ile 255 arasında imzalanmamış 8-bit (1-bayt) tamsayıları tutar. |
| float | Ondalıklı sayıları tutar |
| bool | True veya False |
| int | Tam sayıları tutar |

**Sayısal Tipler**

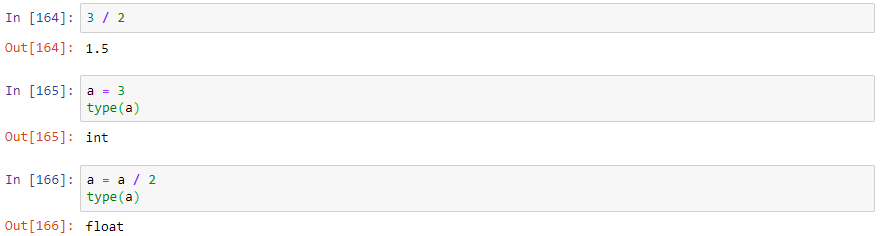
Sayılar için birincil Python tipleri int ve float’tır.



Float veri tipi bilimsel notasyon ile ifade edilebilir:



Tam sayı ile sonuçlanmayan bölme işlemlerinde veri tipi float olur:



Tam sayı ile sonuçlanmayan bölme işlemlerinde sonucu tam sayı istersek “//” operatörünü kullanırız:



**Dizeler (Strings)**

Python’da dizgeler tırnak işaretleri ile bitişik karakter kümesi olarak gösterilmektedir. Python’da string tanımlamak için tek ya da çift tırnak kullanılabilir.



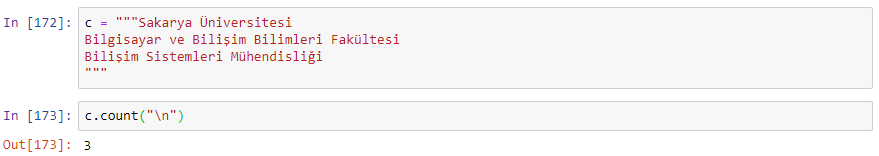
Çoklu satır girmek istiyorsak ''' veya """ kullanmamız gerekir:



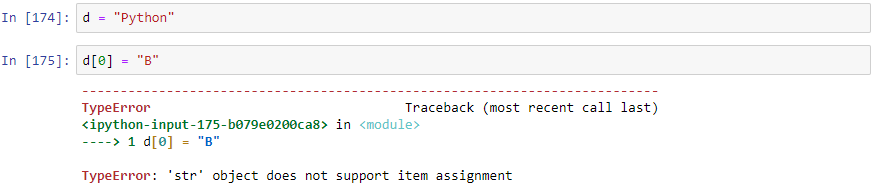
Bir dizedeki satır sayısını öğrenmek istersek count() fonksiyonunu kullanabiliriz:

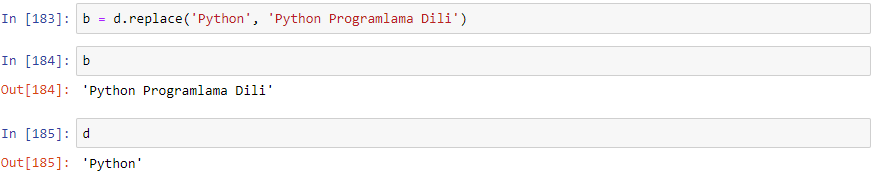


Değişkenin içerisine üç satır girmemize rağmen fonksiyon dört satır gösterdi. Bu durumun sebebi """ işaretinden hemen sonra dizenin başlamasıdır. Örnek:



Dizeler değiştirilemez veri tipidir:





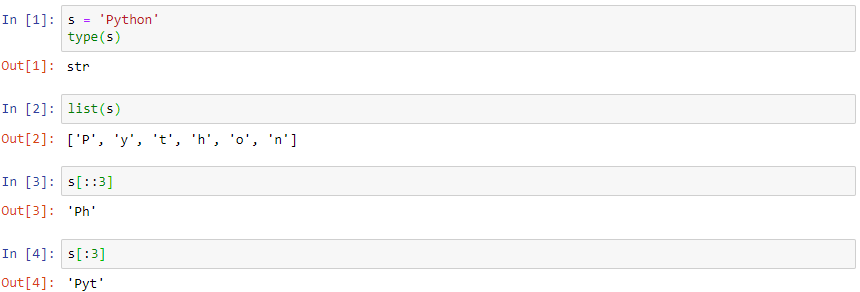
Bu işlem d değişkenini değiştirmez.



Birçok Python nesnesi, str() dönüşümü ile bir dize (string) ifadeye dönüştürülebilir:



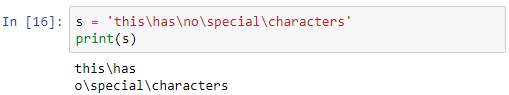
Dizeler bir Unicode karakter dizisidir ve bu nedenle listeler ve demetler gibi ele alınabilir:



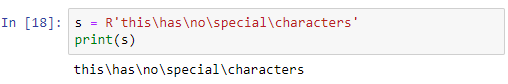
Ters eğik çizgi karakteri ‘\’ bir kaçış karakteridir, yani yeni satır ‘\n’ veya Unicode karakterleri gibi özel karakterleri belirtmek için kullanılır.



Çok fazla ters eğik çizgi bulunan ve özel karakter içermeyen bir dizginiz varsa sonuç çok farklı çıkabilir:

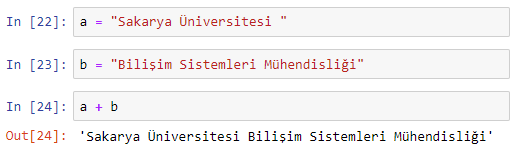


Bu durumu düzeltmek için dizenin başındaki alıntıyı r ile başlatabilirsiniz, yani karakterlerin şu şekilde yorumlanması gerekir:

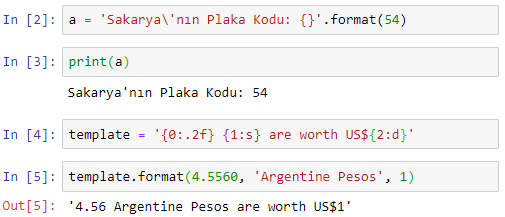


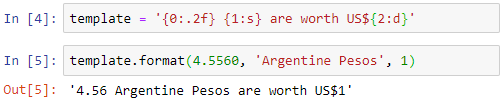
R, ham anlamına gelmektedir.

Birden fazla dizeyi ‘+’ operatörü ile birleştirebiliriz:



Dizeler biçimlendirme yöntemine sahiptir:





Bu dizgede,

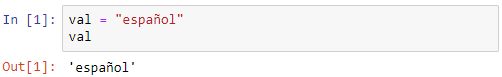
• {0: .2f}, ilk argümanı, iki ondalık basamaklı kayan nokta sayı olarak biçimlendirir.

• {1: s}, ikinci argümanı dize olarak biçimlendirir.

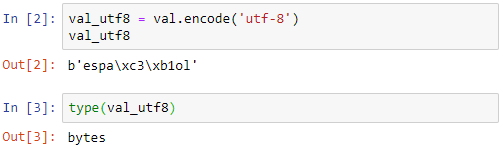
• {2: d}, üçüncü argümanı bir tamsayı olarak biçimlendirir.

**Bayt ve Unicode**

Modern Python'da (yani Python 3.0 ve üstü) Unicode, ASCII ve ASCII olmayan metinlerin daha tutarlı işlemesini sağlayan birinci sınıf dize türü haline gelmiştir.

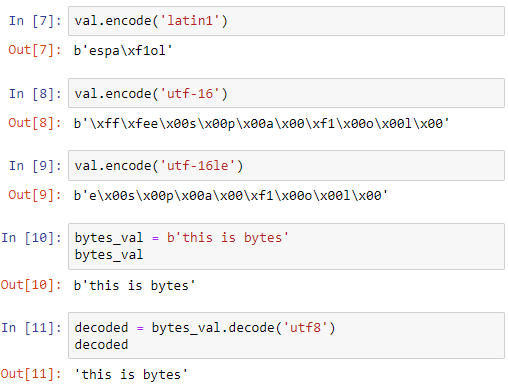


Bu Unicode dizesini, encode yöntemini kullanarak UTF-8 bayt gösterimine dönüştürebiliriz:



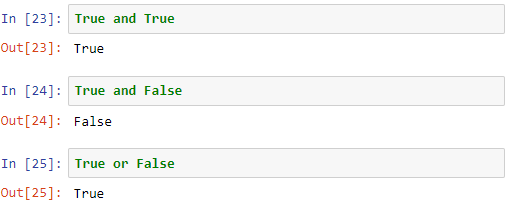
Bir byte nesnesini, decode yöntemini kullanarak geri döndürebiliriz:



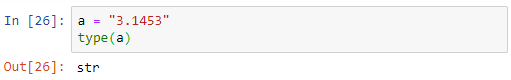


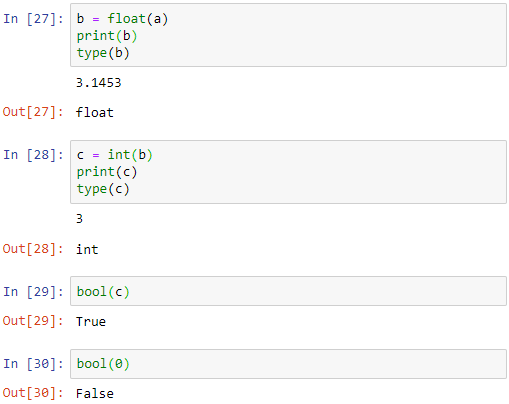
**Booleans**

Boolean, 1 (True) ve 0 (False) olmak üzeri iki değerden oluşmaktadır. Yani Doğru veya Yanlış olarak değerlendirilir. Boolean değerleri and ve or anahtar kelimelerle kullanılır:



**Tip Dönüşümü (Type Casting)**



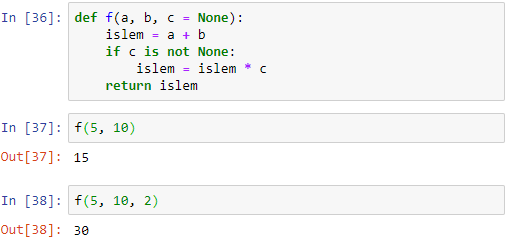


**None**

Nne, Python null değer türüdür. Herhangi bir değeri ifade etmez ve bu sebeple veri tipi belli değildir.

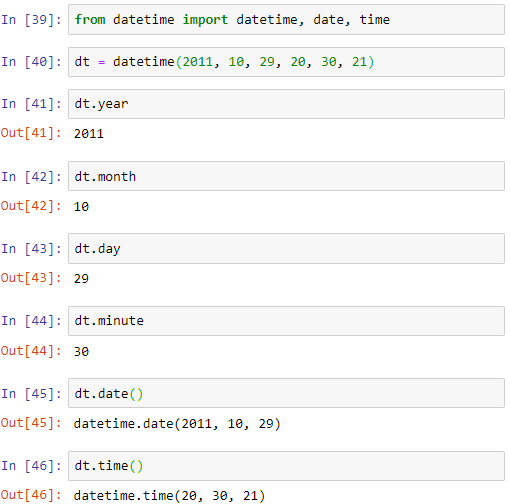


Bir fonksiyona ait parametrelere veya parametreye başlangıç değerini None tipi yapabiliriz. Fonksiyon çağrımında, başlangıç değeri verilmiş bir parametreye değer girmezsek hata almayız:



**Tarihler ve Saatler (Dates and Times)**

Python’da tarih ve saat bilgilerini içeren datetime kütüphanesi vardır. Bu kütüphaneyi kullanabilmemiz için Python çalışma sayfamıza import etmeliyiz.



“strftime” yöntemi, bir datetime nesnesini bir dize olarak biçimlendirir:



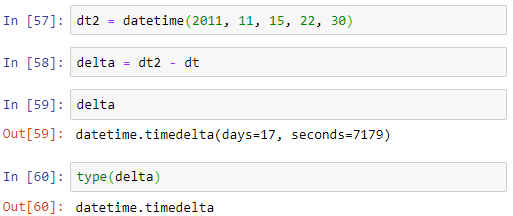
Dizeler “strptime” işleviyle datetime nesnelerine dönüştürülebilir (ayrıştırılabilir):



“replace” fonksiyonu ile bir datetime nesnesi içerisinde değişiklik yapabiliriz:



İki datetime nesnesinin farkı bir “datetime.timedelta” türü oluşturur:



Timedelta çıkışı 17, 7179’dur yani 17 gün ve 7.177 saniyelik bir fark olduğunu gösterir.

Bir datetime nesnesine bir zaman çizelgesi ekleyebiliriz:



**Kontrol Akışı (Control Flow)**

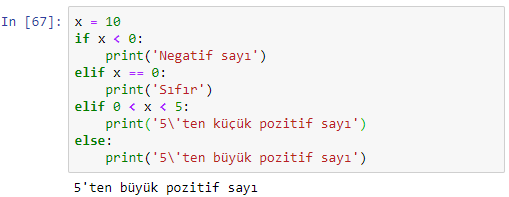
Python, koşullu durumlar, döngüler ve diğer programlama dillerinde bulunan diğer standart kontrol akışı kavramları için birçok yerleşik anahtar kelimeye sahiptir.

**if, elif, and else**

if deyimi, belirtilen koşulun doğruluğunu kontrol eder ve eğer koşul sağlanırsa blok içerisindeki kodlar çalışır.



Bir if deyimini isteğe bağlı sayıda elif deyimi takip edebilir. if koşulunun false olma durumunda elif deyimi ile başka bir koşul kontrol edilir.



Koşullardan herhangi biri True ise ve sonrasında takip eden elif veya else koşulu varsa bunlar çalışmaz.

and veya or kullanılan bir bileşik koşulda, koşullar soldan sağa değerlendirilir.



Eğer or koşulu kullandıysak ve ilk koşul sağlandıysa devamındaki koşul kontrol edilmez.

Zincir karşılaştırmalar yapmak da mümkündür:



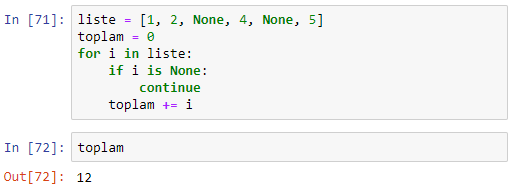
**For Döngüsü (for loops)**

For döngüsü, bir koleksiyon (dizi, demet vb.) veya bir aralık içerisinde gezinmek içindir.

Standart bir for döngüsü tanımlanması:



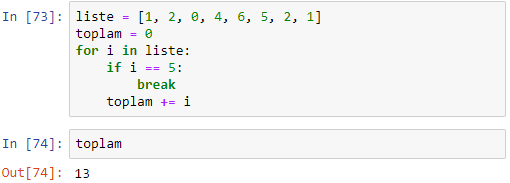
İçerisinde tam sayılar ve null değerler olan bir liste tanımlayarak. Bir for döngüsü ile bu liste üzerinde gezinerek tam sayıları toplayan bir kod bloğu:



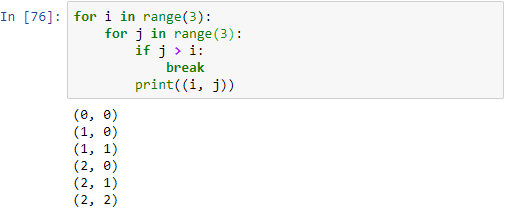
**Continue:** Bulunduğu komut satırından aşağıdaki kodları atlayarak bir sonraki döngüye girmeyi sağlar.

**Break:** Bulunduğu döngüyü sonlandırır.

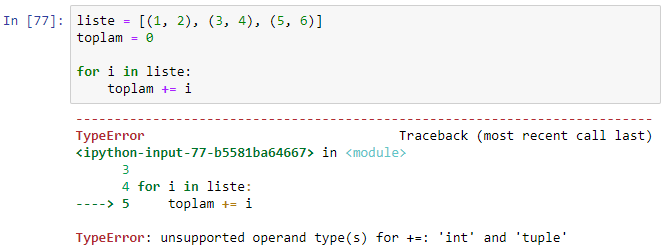
Buradaki örneğimizde if koşulu ile i değerinin 5’e eşit olduğu durumda döngüyü sonlandırıyoruz.



Break anahtar sözcüğü yalnızca en içteki döngü için sonlandırır, dıştaki döngüler çalışmaya devam eder.



Eğer bizi dizi içerisinde demetler halinde veri saklıyorsa bazı noktalarda hata alabiliriz:

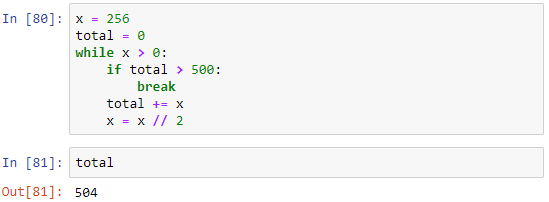


Bu hatanın sebebi; dizimizin indisleri içerisinde bulunan veriler (1, 2) gibi saklanmaktadır ve bu değerleri toplamaya ( (1, 2) + (3, 4) gibi) çalışmak bize hata döndürecektir. Çözümü ise for döngüsü içerisinde demetler içerisindeki veri sayısı kadar değer (value) tanımlamalıyız:



**While Döngüsü (While Loops)**

While döngüsünde tanımlanan koşul False değerine ulaşıncaya kadar veya bir break anahtar sözcüğünü görene kadar kod bloğunu yürütür.



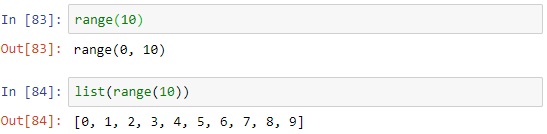
**Pass Anahtar Sözcüğü**

pass, Python'da "no-op" ifadesidir. Hiçbir işlem yapılmayan bloklarda (veya henüz uygulanmayan bir kod için yer tutucu görevi görür) kullanılabilir:

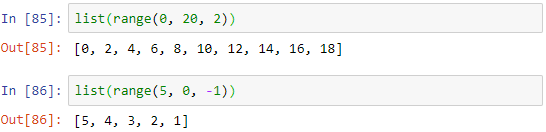


**Range**

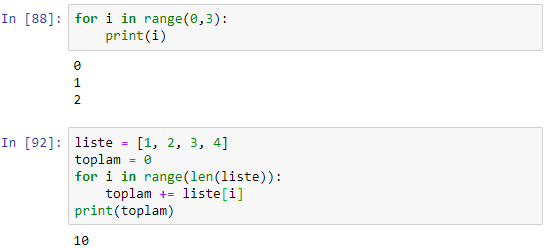
Range işlevi, eşit aralıklı tamsayılar dizisi veren bir yineleyici döndürür:



Bir başlangıç, son ve adım (adım negatif olabilir) verilebilir:



Yaygın olarak döngüler içerisinde kullanılır. Örneğin bir döngünün yineleme aralığı belirtilebilir:



Bu kod, 3 veya 5'in katları olan, 0 ile 99,999 arasındaki tüm sayıları toplar:



**Ternary Expressions**

Python'daki bir üçlü ifade, bir değer üreten bir if-else bloğunu tek bir satır veya ifadede birleştirmenize izin verir. Python'da bunun sözdizimi şöyledir:

*değisken = doğru\_ifade if koşul else yanlış\_ifade*

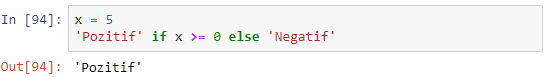
Bu sözdizimi bildiğimiz (ayrıntlı) şekilde tanımlamak istersek:

*if koşul:*

*değisken = doğru\_ifade*

*else:*

*değisken = yanlış\_ifade*



**BÖLÜM 3**

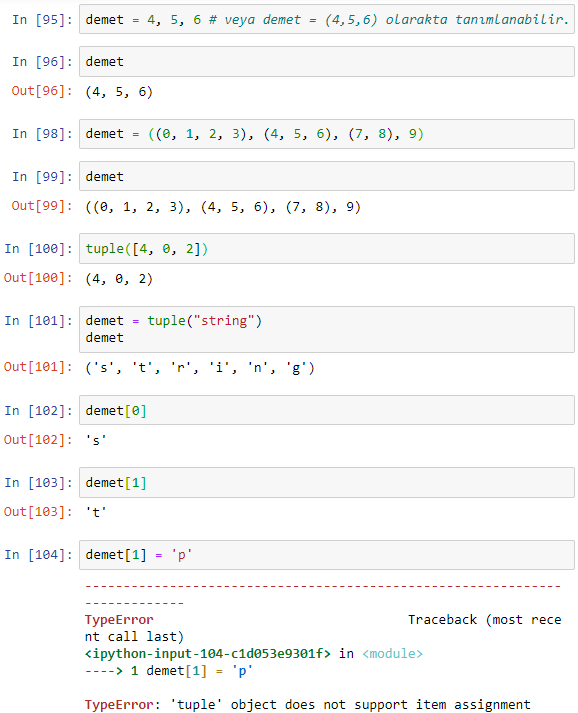
**Yerleşik Veri Yapıları, İşlevleri ve Dosyaları**

**3.1 Veri Yapıları ve Diziler**

Python’un veri yapıları basit ama güçlüdür. Kullanımına hâkim olmak, yetkin bir Python programcısı olmanın kritik bir parçasıdır.

**Demetler (Tuple)**

Demetler oluşturulduktan sonra değiştirilemez bir Python nesne türüdür.



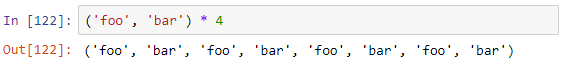
Demetler içerisindeki indislere ulaşabiliriz ancak bu indis değerlerinde bir değişiklik yapamayız çünkü demet nesnesinin mevcut verileri sonradan değiştirilemezdir.



Daha uzun demet üretmek için + operatörünü kullanarak birleştirebiliriz:

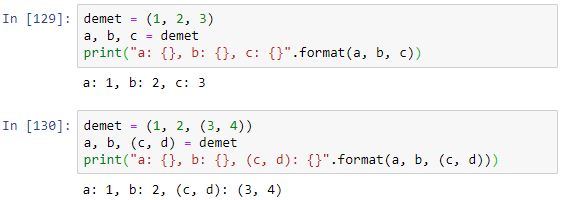


Bir demeti bir tamsayı ile çarpmak, listelerde olduğu gibi demetin birçok kopyasını bir araya getirme etkisine sahiptir:

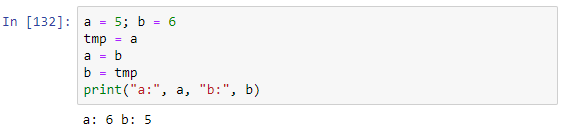


**Demet Açma (Unpacking Tuples)**

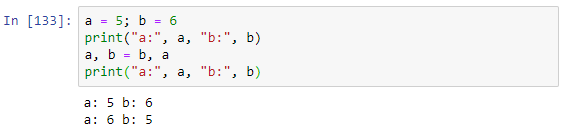
Demet içerisindeki verileri değişkenlere aktarabiliriz.

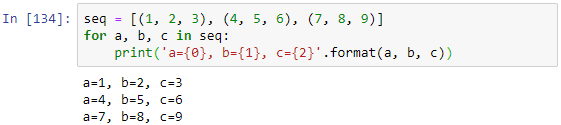


Bu işlevselliği kullanarak, birçok dilde değişkenlere ait verilerin yerlerini değiştirebiliriz:

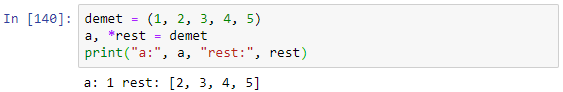


Ancak Python’da bunun kolay yolu vardır:





Python, birden fazla elemanı “almak” isteyebileceğimiz durumlara yardımcı olmak için biraz daha gelişmiş bir demet açma işlemine sahiptir. \*rest anahtar sözcüğünü kullanırız.

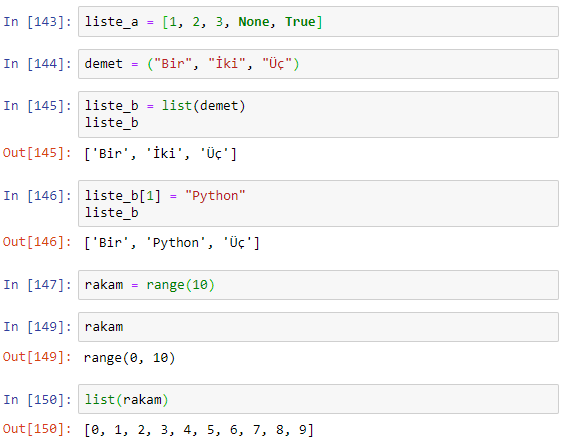


Bir verinin demet içerisinde kaç kez geçtiğini count fonksiyonu ile öğrenebiliriz:



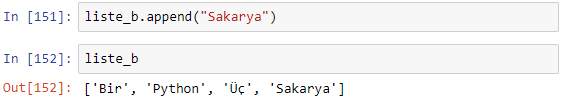
**Listeler (List)**

Demetlerin aksine, listeler değişken uzunluktadır ve içerikleri veriler değiştirilebilir. Köşeli parantez [] veya list() işlevini kullanarak tanımlayabiliriz:

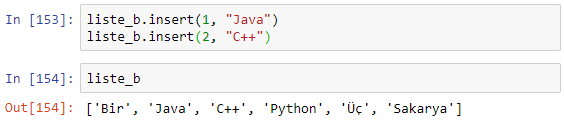


**Eleman Ekleme ve Çıkarma**

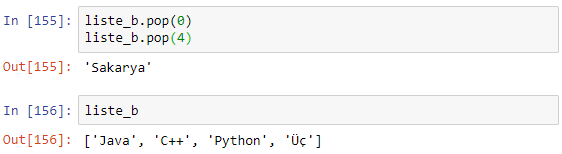
Listenin sonuna eleman eklemek için append metodu kullanılır.



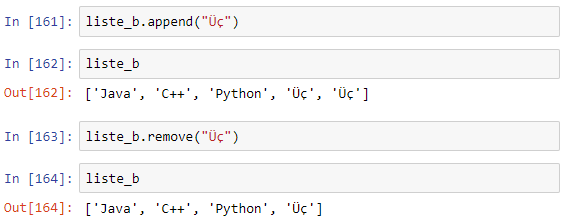
Listenin belirlediğimiz bir indise eleman eklemek için insert metodu kullanılır.



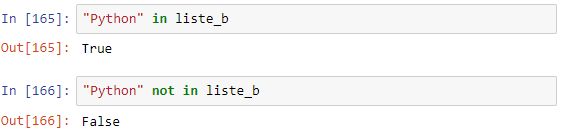
İstediğimiz indisteki elemanı listeden çıkarmak için pop metodu kullanılır.



remove metodu silinmesini istediğimiz elemanın ismini verdiğimizde listenin ilk indisinden başlayarak ileri doğru gider ve ilk eşleştirdiği indisi siler.

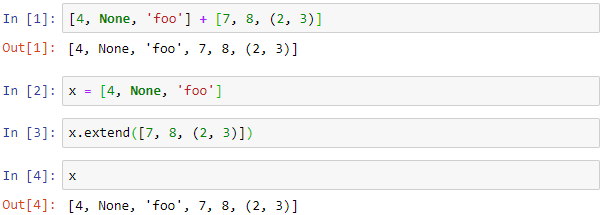


Bir listenin in anahtar sözcüğünü kullanarak bir elemanı içerip içermediğini kontrol edebiliriz:



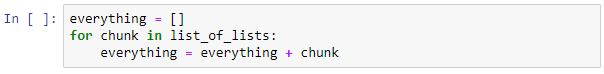
**Listeleri Birleştirme**

Demetlerde olduğu gibi + operatörü ile listeleri birleştirebiliriz.



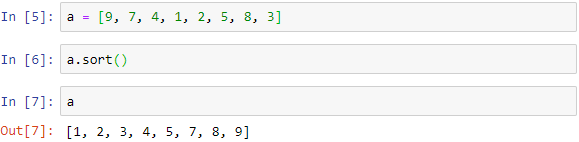
Liste birleştirme işleminde nesneler kopyalanır ve yeni bir listede birleştirilir. Bu işlem zahmetli bir iştir. Bunun yerine extend metodu ile var olan bir listeye öğeleri eklemek daha yararlıdır.



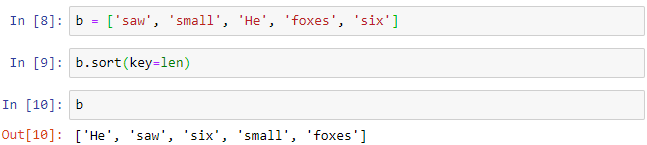


**Sıralama (Sorting)**

Bu fonksiyon ile bir listeyi (yeni bir liste oluşturmadan) sıralayabiliriz.



İkincil bir sıralama anahtarı tanımlayabiliriz. Örneğin liste elemanlarını uzunluklarına göre sıralayabiliriz:

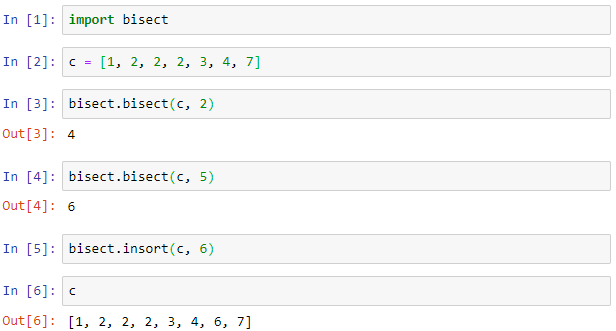


**İkili Arama (Binary search and maintaining a sorted list)**

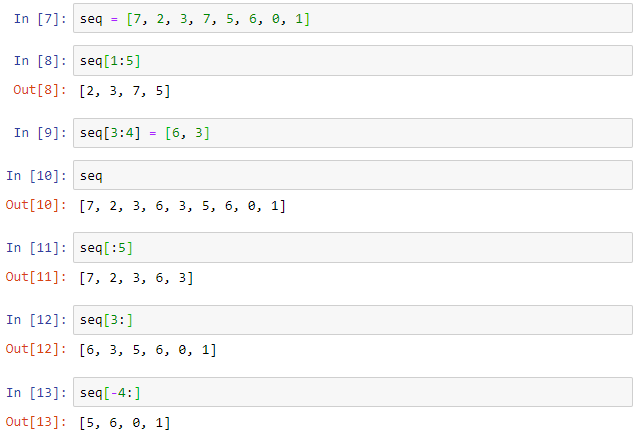
Yerleşik bisect modülü, ikili arama ve sıralı listeye eleman eklemeyi sağlar.

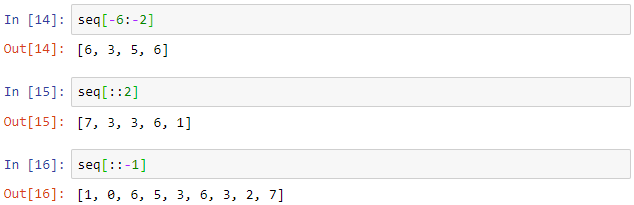
bisect.bisect: Bir elemanın, sıralanmış listede ekleneceği yeri bulur.

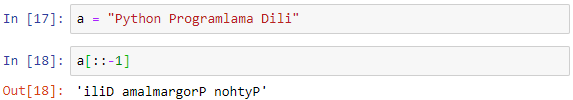
bisect.insort: Elemanı gerçekte o konuma yerleştirir.



**Dilimleme (Slicing)**







**Yerleşik Liste Fonksiyonları**

**Enumerate**

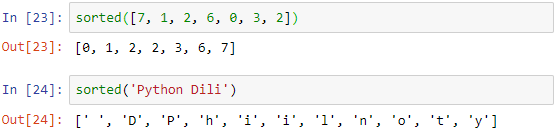
Bir listenin indeksini takip etmek için liste tekrarlarken genelde şöyle yaparız:



Python, (i, değer) demet dizisini döndüren enumerate yerleşik bir fonksiyona sahiptir:

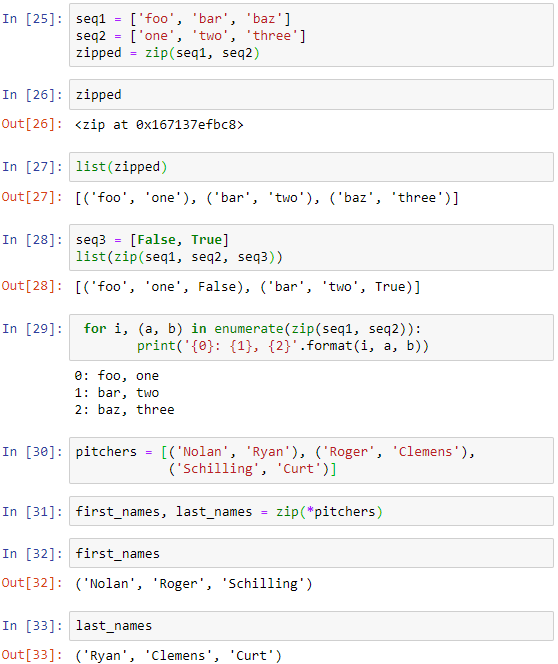


**Sorted**

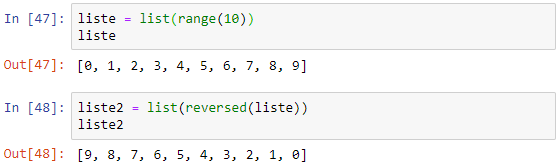


**Zip**

Listelerin aynı indislerinde bulunan elemanları alarak bir demetler listesi oluşturur.



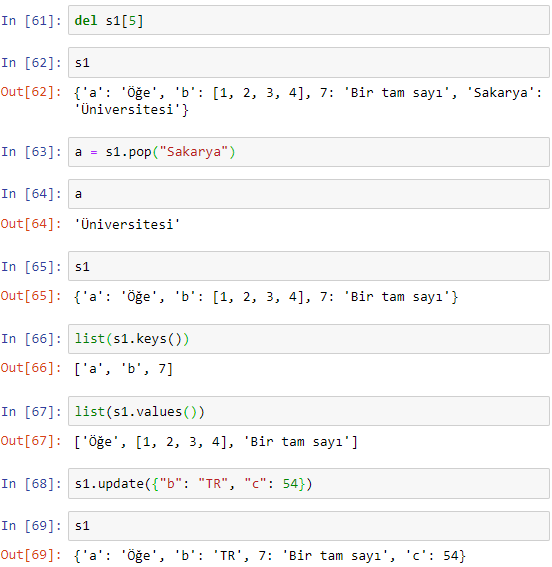
**Reversed**



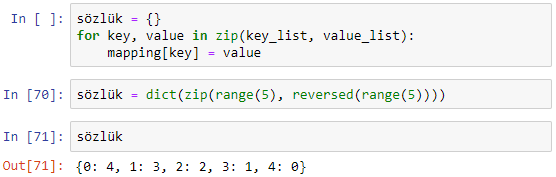
**Sözlükler (Dictionary)**

Python Sözlükler (Dictionary), sırasız bir öğe koleksiyonudur. Key, value yani anahtar ve değer yapısında veri kümeleri oluştururlar. Her bir anahtar 'key' değerinden iki nokta üst üste işareti ile ayrılır ve bunlara öğe denir. Bu öğeler süslü parantez içerisinde virgülle ayrılarak yer alırlar. Herhangi bir öğeye sahip olmayan boş sözlükler içi boş süslü parantezlerle ifade edilirler. {}

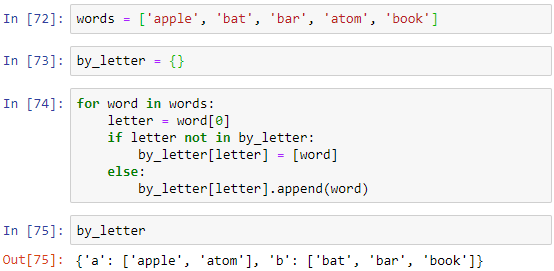


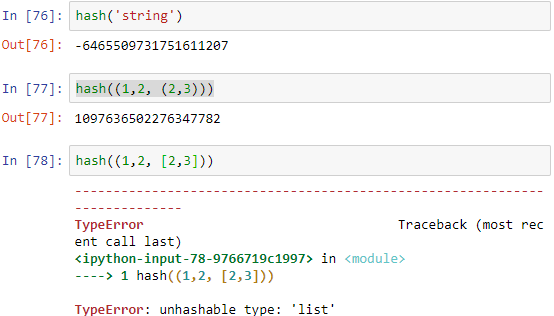


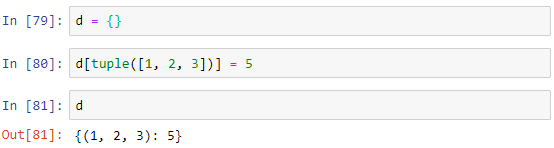
**Listelerden Sözlük Oluşturma (Creating dicts from sequences)**



**Varsayılan Değerler (Default Values)**

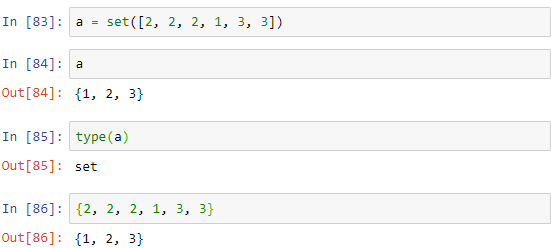
****

****

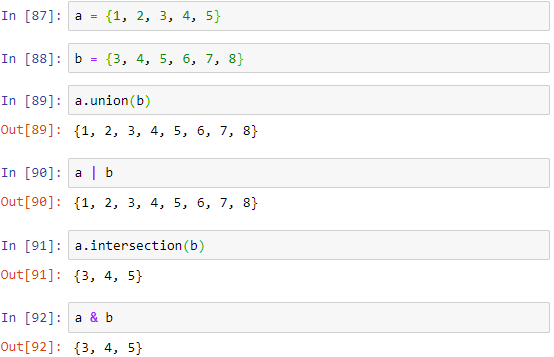
****

**Küme (Set)**

Bir küme, sıra dışı bir benzersiz elemanlar topluluğudur. Kümeleri bir sözlük gibi düşünebilirsiniz, ancak yalnızca anahtar (key) değer yok. Bir küme iki şekilde oluşturulabilir: set() aracılığıyla veya süslü parantez {}

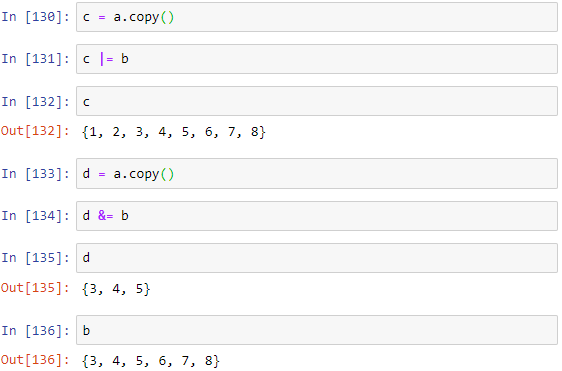


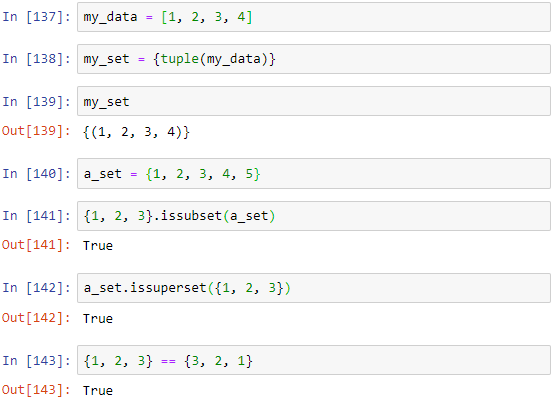
Kümeler, birleşme, kesişme, fark ve simetrik fark gibi matematiksel küme işlemlerini destekler. Bu iki örnek seti üzerinde bu örnekleri inceleyebiliriz:



*Tablo: Python Küme İşlemleri*

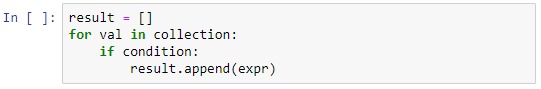
|  |  |  |
| --- | --- | --- |
| Fonksiyon | Alternatif Syntax | Açıklama |
| a.add(x) |  | x öğesini a kümesine ekle |
| a.clear() |  | a kümesinin elemanlarını temizler ve boş küme haline getirir. |
| a.remove(x) |  | x öğesini a kümesinden çıkar |
| a.pop() |  | a kümesinden rasgele bir öğe çıkar |
| a.union(b) | a | b | a ve b kümesindeki birbirinden farklı öğeleri alır |
| a.update(b) | a |= b | a kümesini, a ve b’nin birbirinden farklı öğelerinin birleşimine eşitler |
| a.intersection(b) | a & b | a ve b kümlerinin ortak öğelerini alır |
| a.intersection\_update(b) | a &= b | a kümesini, a ve b kümlerinin ortak öğelerine eşitler |
| a.difference(b) | a - b | a’nın içinde olan ama b’de olmayan öğeleri alır |
| a.difference\_update(b) | a -= b | a kümesini, a’nın içinde olan ama b’de olmayan öğelerine eşitler |
| a.symmetric\_difference(b) | a ^ b | a ve b kümelerinde aynı elemanlar haricindekileri alır |
| a.symmetric\_difference\_update(b) | a ^= b | a ve b kümelerinde aynı elemanlar haricindekileri alır ve a kümesine eşitler |
| a.issubset(b) |  | a öğelerinin tümü b'de bulunuyorsa True |
| a.issuperset(b) |  | b öğelerinin tümü a içinde bulunuyorsa True |
| a.isdisjoint(b) |  | a ve b'nin ortak bir öğesi yoksa True |





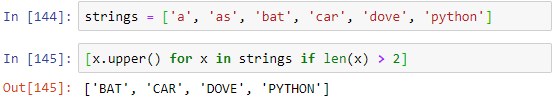
**List, Set, and Dict Comprehensions**

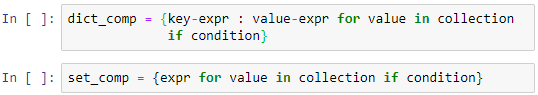
Koleksiyonun öğelerini filtreleyerek, filtreden geçen öğeleri tek bir ifadeye dönüştürerek yeni bir liste oluşturmanıza olanak tanır.

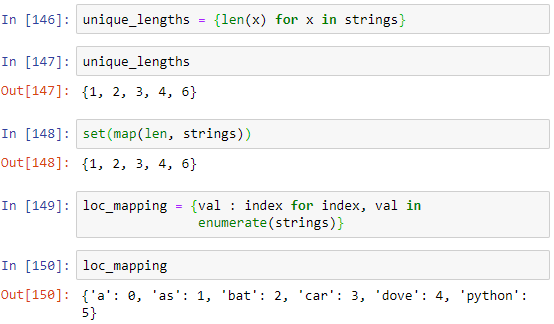


Bu yöntem yukarıdaki klasik kod parçası ile aynı görevi görür.

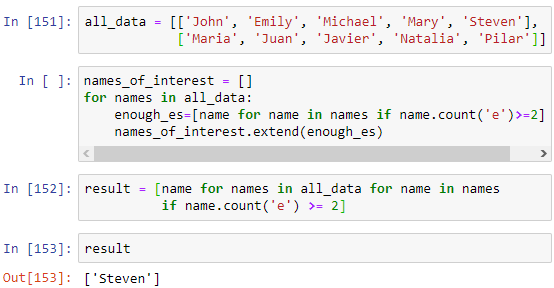


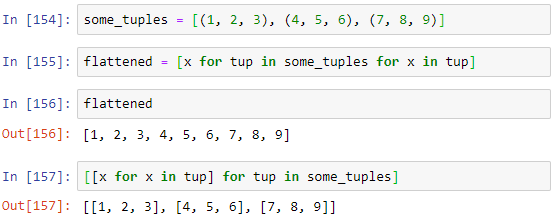






**Nested List Comprehensions (İç içe liste)**

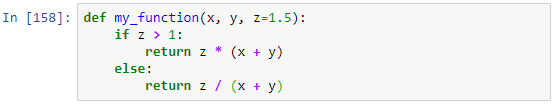




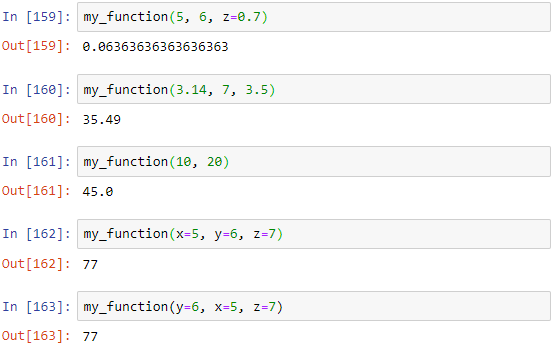
**3.2. Fonksiyonlar (Functions)**

Fonksiyonlar, belli işlevleri olan ve tekrar tekrar kullandığımız yapılardır. Kural olarak, aynı veya çok benzer kodu bir kereden fazla tekrarlayan kod varsa bunu tek bir fonksiyon içerisinde yapabiliriz. Fonksiyonlar, kodunuzu daha okunabilir hale getirmenize de yardımcı olabilir.

Fonksiyonlar def anahtar sözcüğü ile bildirilir ve return anahtar sözcüğü ile sonlandırılır:



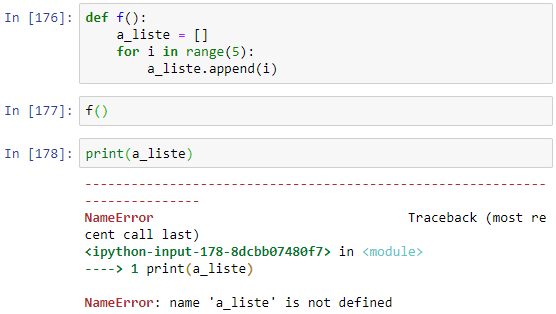
Her fonksiyonun konumsal argümanları ve anahtar kelime argümanları olabilir. Anahtar sözcüğü argümanları varsayılan değerleri veya isteğe bağlı argümanları belirtmek için kullanılır. Yukarıdaki fonksiyonda, x ve y konumsal argümanlardır, z ise bir anahtar sözcüğü argümanıdır. Bu, fonksiyonun şu yollardan biriyle çağrılabileceği anlamına gelir:



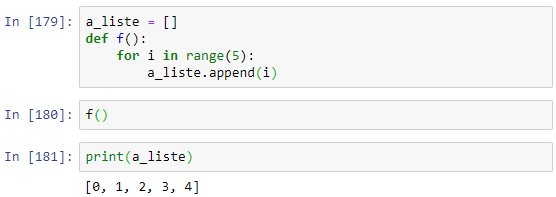
**İsim Alanları, Kapsam ve Yerel Fonksiyonlar**

Fonksiyonlar değişkenlere iki farklı kapsamda erişebilir: global ve yerel. Bir fonksiyon içerisinde tanımlanan değişkenler yerel alanda oluşturulmuş olur. Fonksiyon çalıştırıldığın oluşturulur ve fonksiyon bitiminde yok olur. Bu değişkenlere bazı istisnalar hariç fonksiyonun dışarısından erişilemezdir.

Örnek:



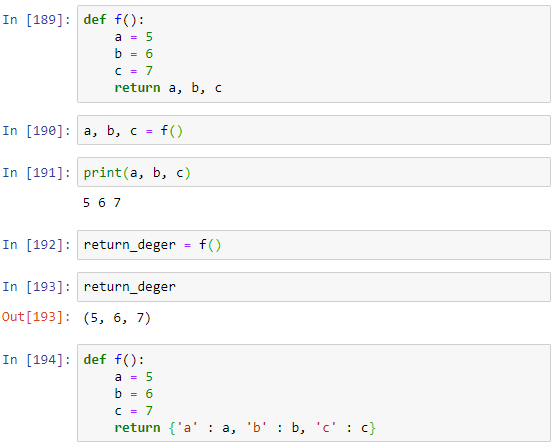
Fonksiyonun dışında tanımlarsak:



global anahtar sözcüğü ile bir fonksiyonun dışındaki değişkenleri tanımlayabiliriz:

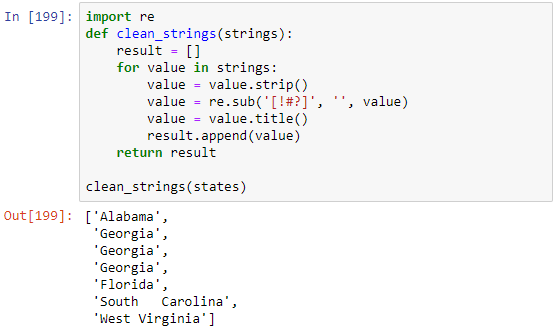


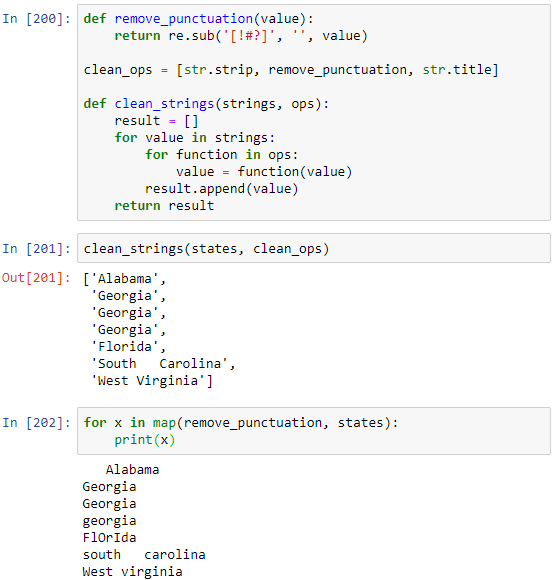
**Birden Fazla Değer Döndürme**



**Fonksiyonlar Nesnedir**

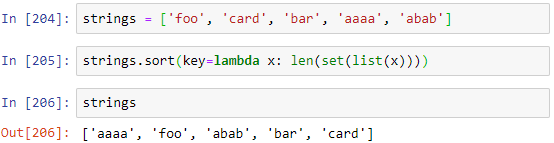
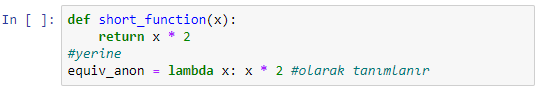
Python fonksiyonları nesne olduğundan, diğer dillerde yapılması zor olan birçok yapı kolayca ifade edilebilir. Bazı veri temizleme işlemlerini yaptığımızı ve aşağıdaki listeye bir dizi dönüşüm uygulamamız gerektiğini varsayalım:





**Anonymous (Lambda) Fonksiyonlar**

Python, tek bir ifadeden oluşan fonksiyonların yazılmasının bir yolu olan ve bunun sonucu olarak dönüş değeri olan anonim veya lambda fonksiyonuna sahiptir. lambda anahtar sözcüğü ile tanımlanır:

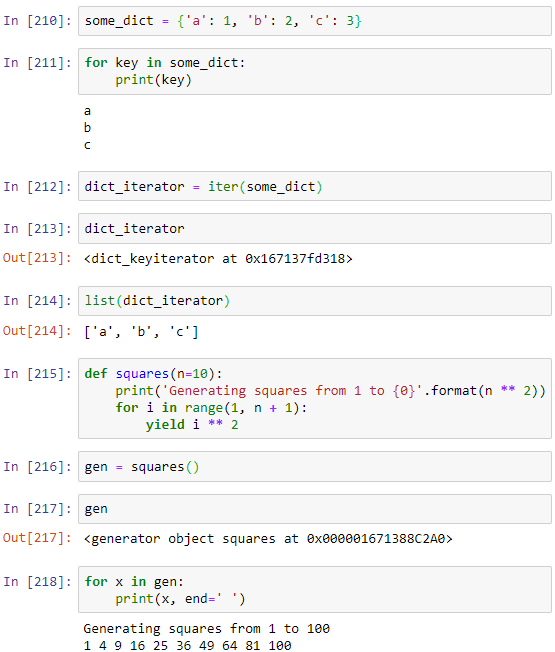


**Currying: Kısmi Argüman Uygulaması**

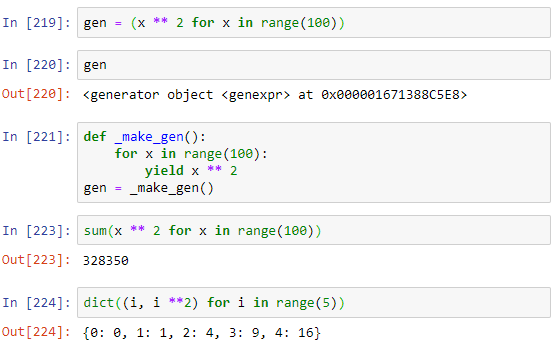
Kısmi argüman uygulaması ile mevcut fonksiyonlardan yeni fonksiyonlar türetme anlamına gelir. Örneğin, iki sayı ekleyen önemsiz bir fonksiyonumuz olsun:



**Generators**

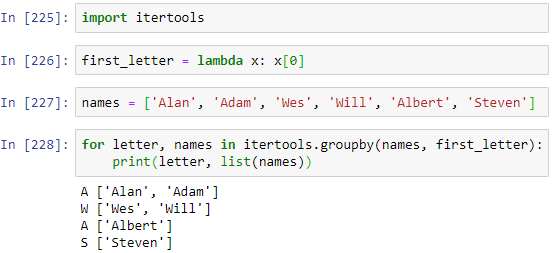


**Generator Expresssions**



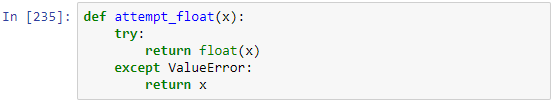
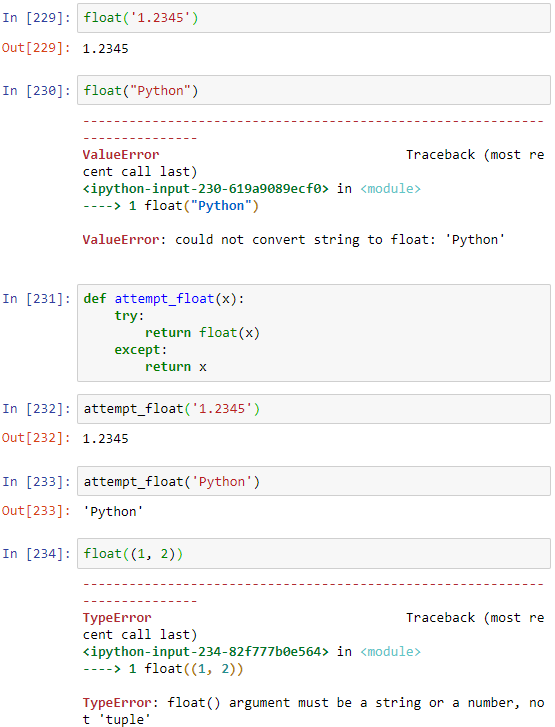
**itertools Modül**

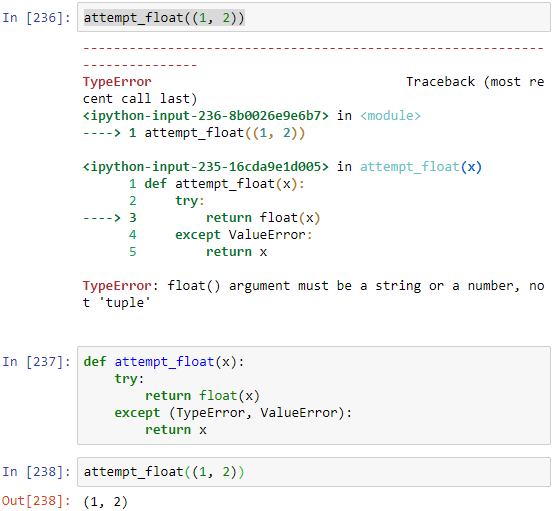
Standart kütüphane itertools modülü, birçok yaygın veri algoritması için bir generatorkoleksiyonuna sahiptir. Örneğin, groupby herhangi bir diziyi ve fonksiyonu alır, listedeki ardışık elemanları fonksiyonun dönüş değeri ile gruplandırır:



**Hatalar ve İstisna İşleme (Errors and Exception Handling)**

Python hatalarını veya istisnalarını incelikle kullanmak, sağlam programlar oluşturmanın önemli bir parçasıdır. Veri analizi uygulamalarında, birçok fonksiyon yalnızca belirli girdi türleri üzerinde çalışır.





try bloğundaki kodun başarılı olup olmamasına bakılmaksızın bazı kodların yürütülmesini isteyebiliriz. Bunu yapmak için, finally kullanılır:



Burada f dosya bağlantısı, kodun başarılı olup olmamasına bakmadan en son kapanır.

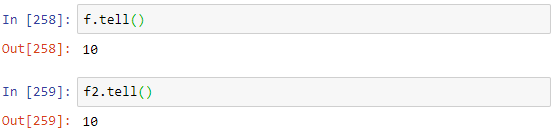
**IPython'da İstisnalar (Exceptions in IPython)**

Bir deyimi çalıştırırken veya herhangi bir dosyayı çalıştırırken istisna ortaya çıkarsa, IPython varsayılan olarak o istisna ile ilgili bilgi ekranı basar:

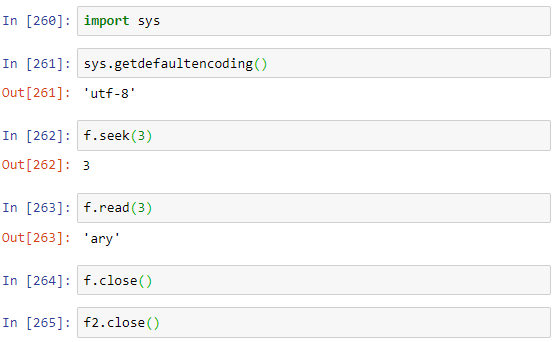
**3.3 Dosyalar (Files and the Operating System)**



Eğer o anda dosyanın hangi bayt konumunda bulunduğumuzu öğrenmek istersek tell() adlı bir metottan yararlanabilirsiniz. Bu metodu parametresiz olarak kullanıyoruz:

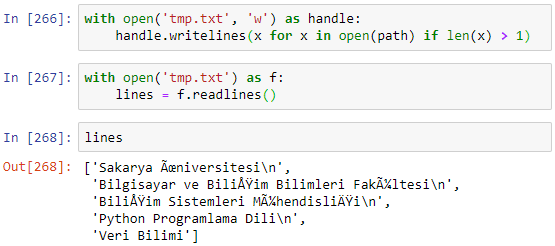


Hangi konumdan başlamak istediğimizi seek metodu ile belirtebiliriz. İşlemlerimiz bittiğinde dosyayı mutlaka kapatmalıyız.



*Tablo: Dosya Kipleri*

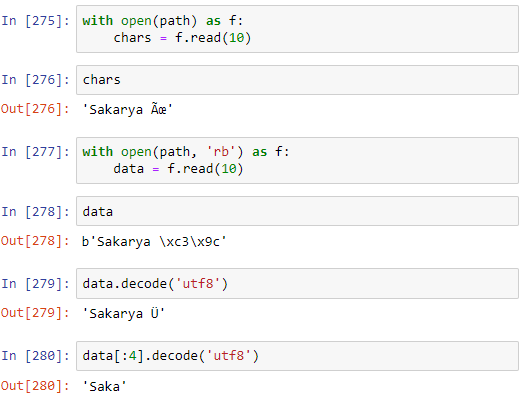
|  |  |
| --- | --- |
| Kip | Açıklama |
| r | Eğer bu kipte açılmak istenen dosya mevcut değilse Python bize bir hata mesajı gösterecektir. Dediğimiz gibi, bu öntanımlı kiptir. Dolayısıyla dosyayı açarken herhangi bir kip belirtmezsek Python dosyayı bu kipte açmak istediğimizi varsayacaktır. |
| w | Bu kip dosyayı yazma yetkisiyle açar. Eğer belirttiğiniz adda bir dosya zaten disk üzerinde varsa, Python hiçbir şey sormadan dosya içeriğini silecektir. Eğer belirttiğiniz adda bir dosya diskte yoksa, Python o adda bir dosyayı otomatik olarak oluşturur. |
| x | Bu kip dosyayı yazma yetkisiyle açar. Eğer belirttiğiniz adda bir dosya zaten disk üzerinde varsa, Python varolan dosyayı silmek yerine size bir hata mesajı gösterir. Zaten bu kipin “w” kipinden farkı, varolan dosyaları silmemesidir. Eğer belirttiğiniz adda bir dosya diskte yoksa, bu kip yardımıyla o ada sahip bir dosya oluşturabilirsiniz. |
| a | Bu kip dosyayı yazma yetkisiyle açar. Eğer dosya zaten disk üzerinde mevcutsa içeriğinde herhangi bir değişiklik yapılmaz. Bu kipte açtığınız bir dosyaya eklediğiniz veriler varolan verilere ilave edilir. Eğer belirttiğiniz adda bir dosya yoksa Python otomatik olarak o adda bir dosyayı sizin için oluşturacaktır. |
| r+ | Bu kip, bir dosyayı hem yazma hem de okuma yetkisiyle açar. Bu kipi kullanabilmeniz için, belirttiğiniz dosyanın disk üzerinde mevcut olması gerekir. |
| rb | Bu kip, metin dosyaları ile ikili (binary) dosyaları ayırt eden sistemlerde ikili dosyaları okuma yetkisiyle açmak için kullanılır. “r” kipi için söylenenler bu kip için de geçerlidir. |
| wb | Bu kip, metin dosyaları ile ikili dosyaları ayırt eden sistemlerde ikili dosyaları yazma yetkisiyle açmak için kullanılır. “w” kipi için söylenenler bu kip için de geçerlidir. |



*Tablo: Dosya Metotları*

|  |  |
| --- | --- |
| Metot | Açıklama |
| read([size]) | Boyut argümanı ile isteğe bağlı okunacak bayt sayısını dosyadan bir dize olarak veri döndürür |
| readlines([size]) | Boyut argümanı ile isteğe bağlı sayıda dosyadaki satırları okuyarak liste halinde döndürür. |
| write(str) | Karakter dizesini dosyaya yazar |
| writelines(strings) | Dosyaya liste tipinde verileri yazma imkanı verir |
| close() | Bağlantıyı kapatır |
| flush() | Dosyada yapılan değişikleri disk üzerine kaydeder |
| seek(pos) | Belirtilen dosya konumuna getir (tam sayı) |
| tell() | Hangi bayt konumunda bulunduğunuzu gösterir |

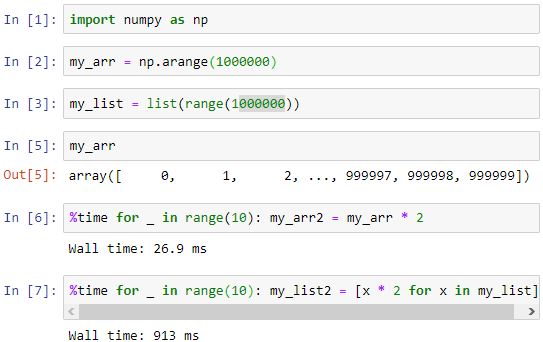
**Bytes and Unicode with Files**



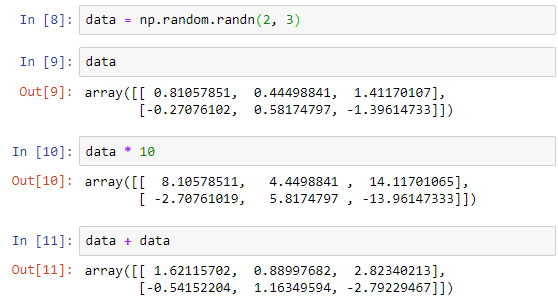
**BÖLÜM 4**

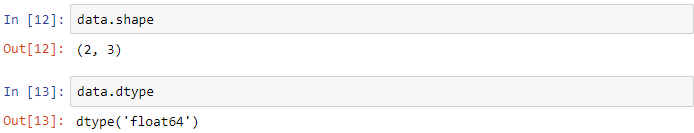
**NumPy Temelleri: Diziler ve Vektörize Hesaplama**

NumPy (Numerical Python) bilimsel hesaplamaları hızlı bir şekilde yapmamızı sağlayan bir matematik kütüphanesidir. Numpy’ın temelini numpy dizileri oluşturur. Numpy dizileri python listelerine benzer fakat hız ve işlevsellik açısından python listelerinden daha kullanışlıdır. Ayrıca python listelerinden farklı olarak Numpy dizileri homojen yapıda olmalıdır yani dizi içindeki tüm elemanlar aynı veri tipinden olmalıdır.

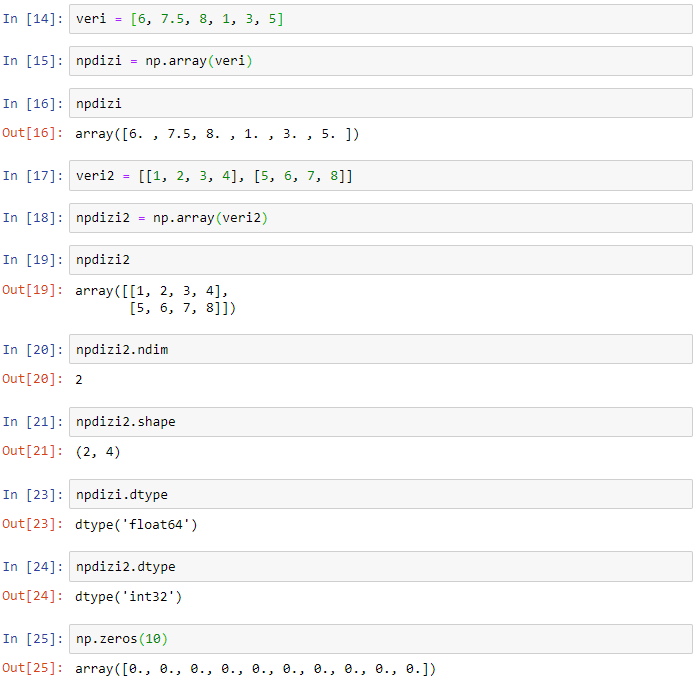


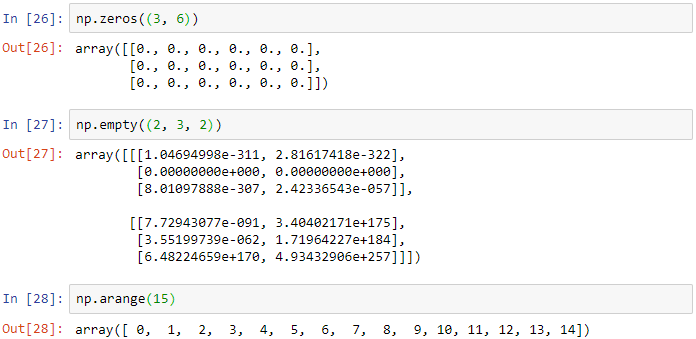
**4.1 NumPy ndarray: Çok Boyutlu Bir Dizi Nesnesi**





**Dizi Oluşturma**



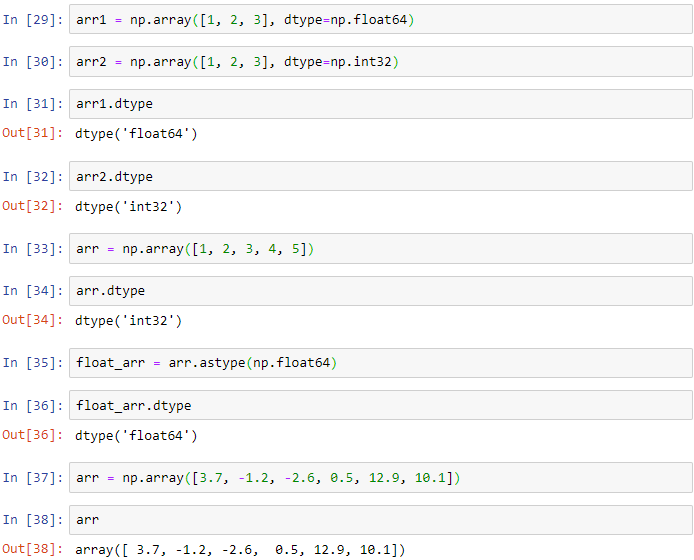


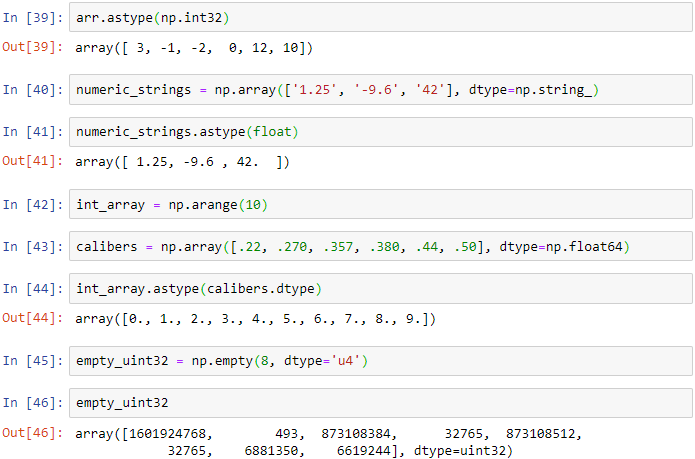


*Tablo: Dizi oluşturma İşlemleri*

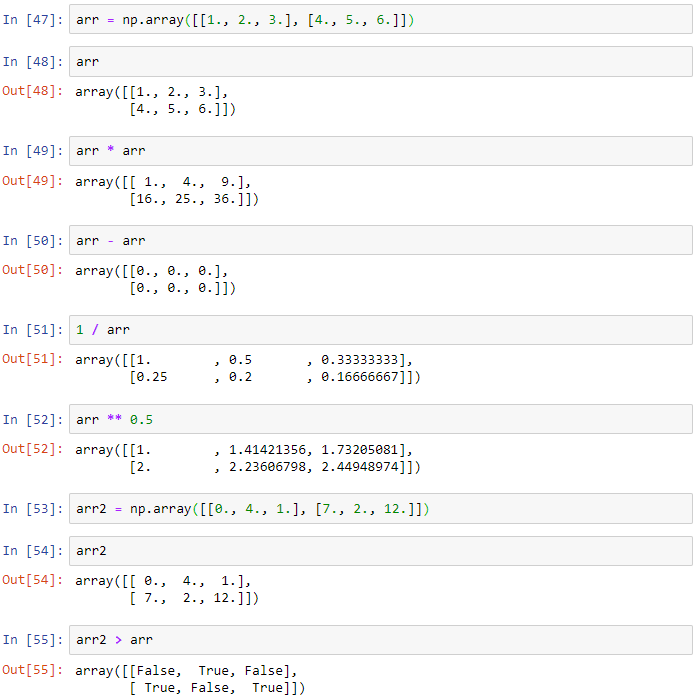
|  |  |
| --- | --- |
| İşlemler | Açıklama |
| array | Girdi verilerini (liste, demet, dizi vs.) bir bir ndarray'e dönüştürür |
| asarray | Girdi verilerini ndarray'a dönüştürür ancak giriş zaten bir ndarray ise kopyalamaz |
| arange | range metodu gibidir ancak bir liste yerine ndarray döndürür |
| ones | Elemanlarının hepsi 1 olan dizi yaratır. |
| zeros | Elemanlarının hepsi 0 olan dizi oluşturmamızı sağlar. |
| empty | Hafızanın durumuna göre rastgele belirlenmiş elemanlardan oluşan dizi yaratır. |

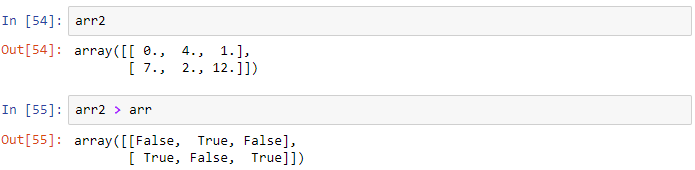
**ndarray Veri Tipleri**



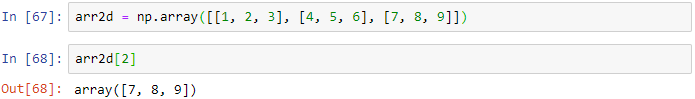


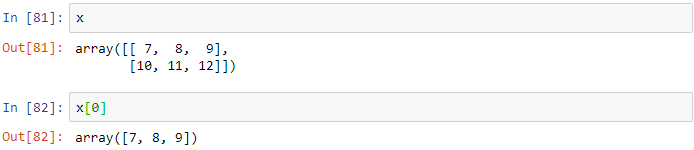
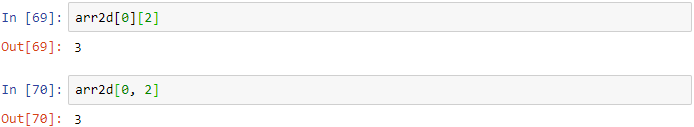
**NumPy Dizileri ile Aritmetik İşlemler**



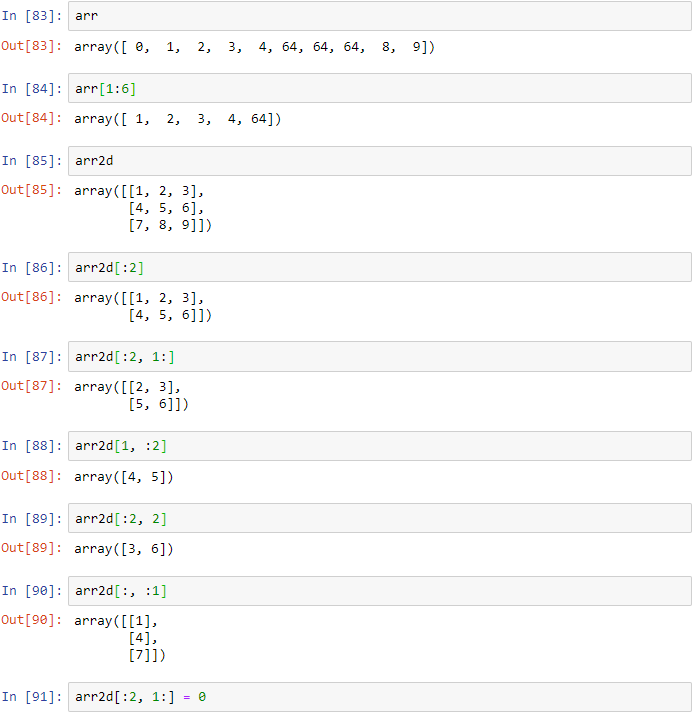


**Temel İndeksleme ve Dilimleme**

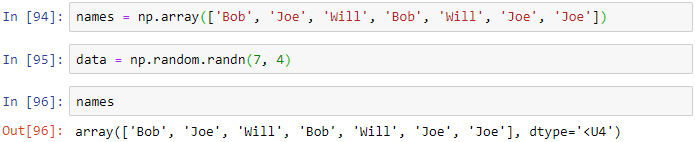


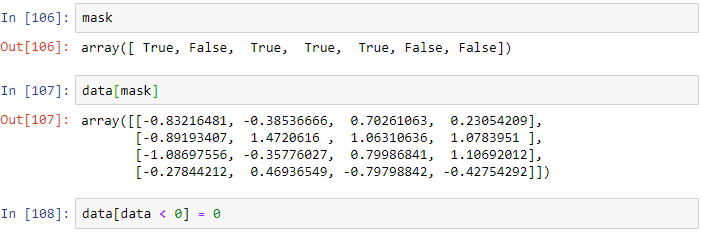
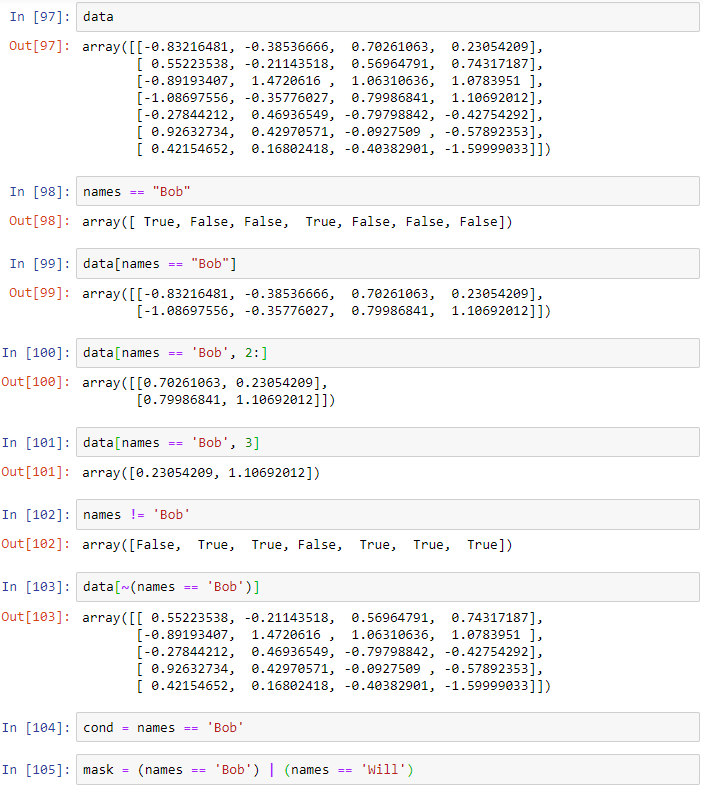


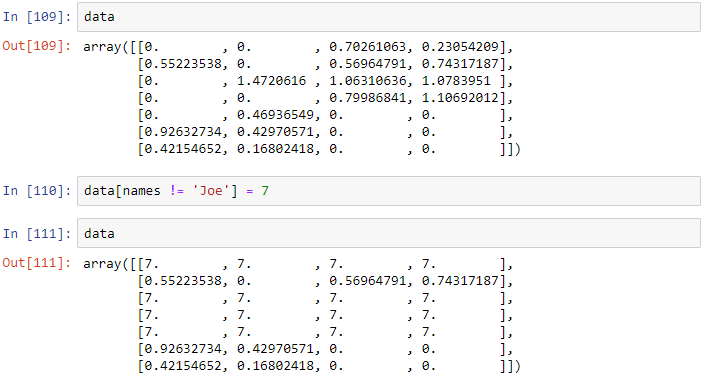
**Dilimlerle İndeksleme**



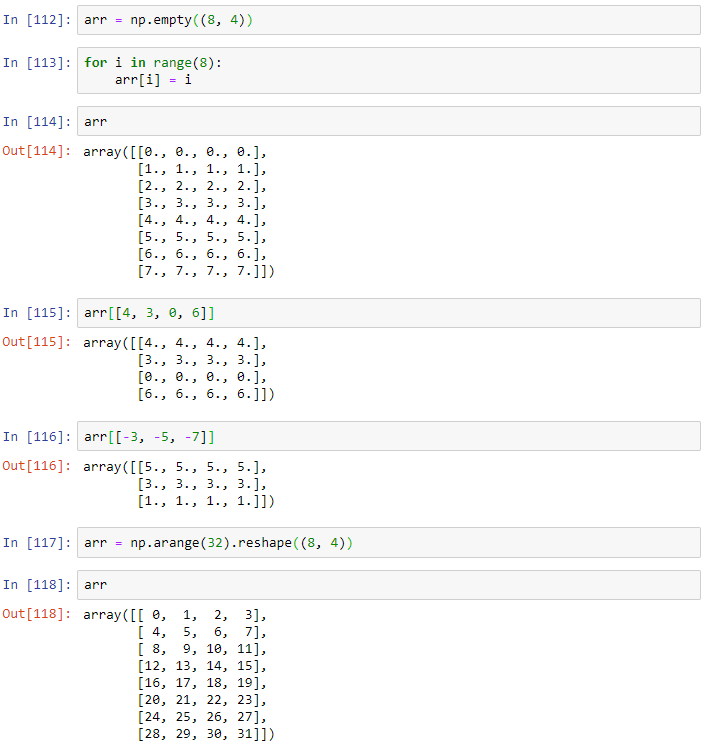
**Boolean Indexing**

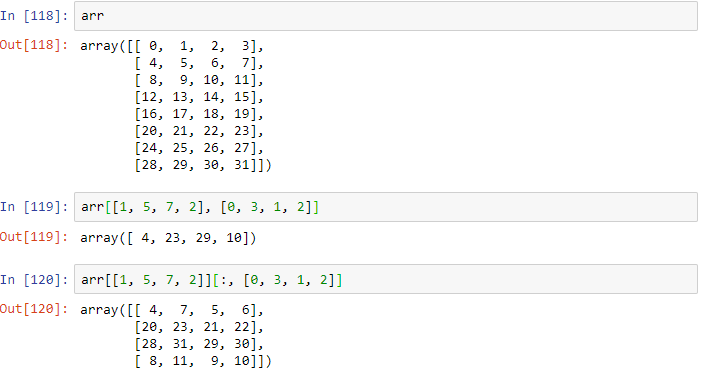




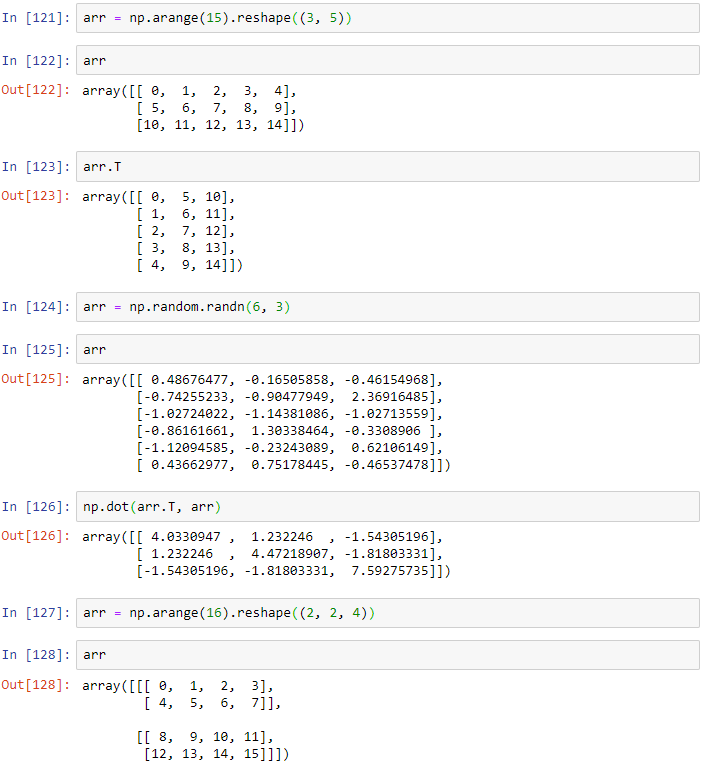


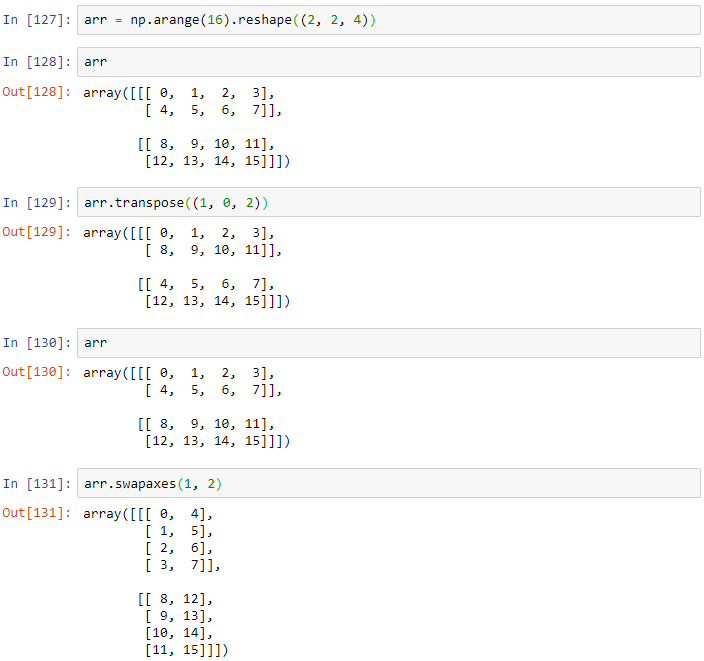
**Fancy Indexing**



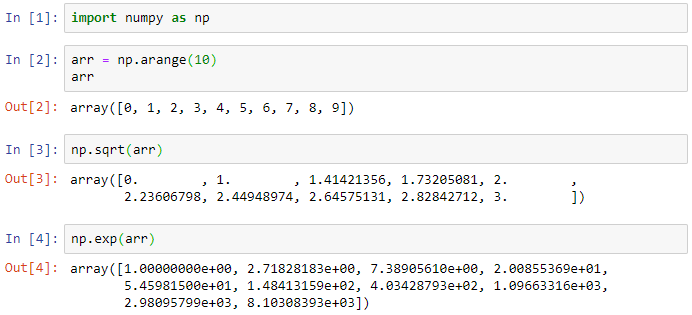


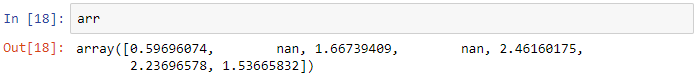
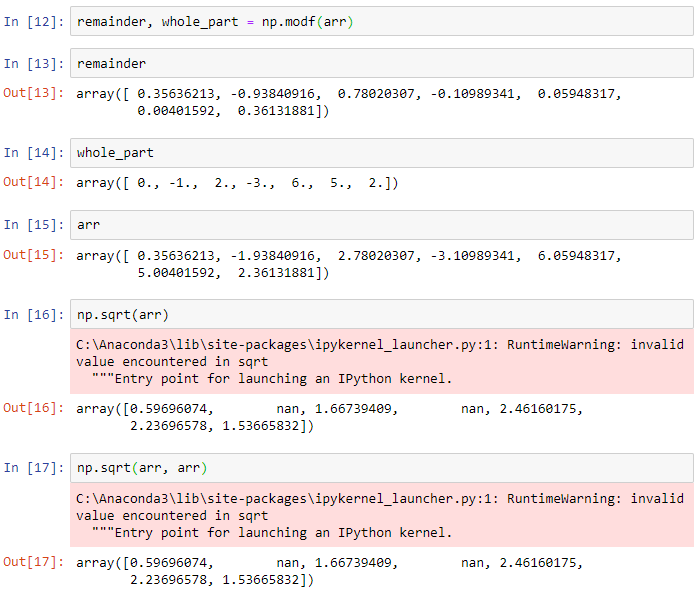
Dizilerde Transpoz İşlemi





**4.2 Evrensel Fonksiyonlar**





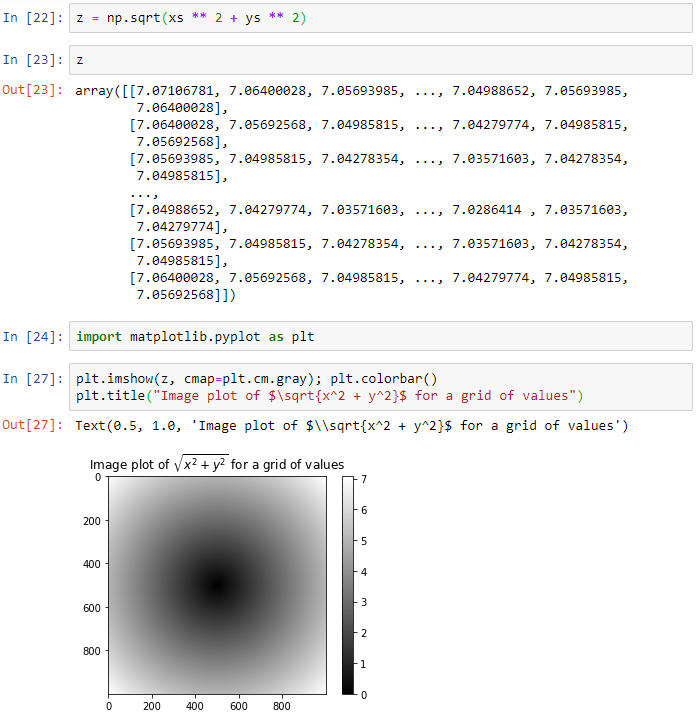
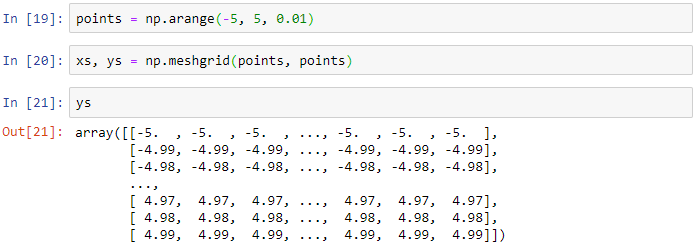
*Tablo: Evrensel Fonksiyonlar*

|  |  |
| --- | --- |
| Fonksiyon | Açıklama |
| abs, fabs | Tamsayı, kayan nokta veya karmaşık değerler için mutlak değer hesaplar |
| sqrt | Her öğenin karekökünü hesaplar (array \*\* 0.5'e eşdeğerdir) |
| square | Her öğenin karesini hesaplar (array \*\* 2'ye eşdeğerdir) |
| exp | Her elemanın e \*\* x değerini hesaplar |
| log, log10, log2, log1p | Sırasıyla, doğal logaritma(taban e), log tabanı 10, log tabanı 2 ve log (1 + x) |
| sign | Her öğenin işaretini hesaplar: 1 (pozitif), 0 (sıfır) veya -1 (negatif) |
| ceil | Paremetre olarak girilen bir ondalıklı sayıyı bir üst tamsayıya yuvarlar. Negatif sayılar içinde kullanılabilir. |
| floor | Parametre olarak girilen ondalıklı sayının virgülden sonraki kısmın büyüklüğünü dikkate almadan bir alt tam sayıya yuvarlar. |
| rint | Öğeleri en yakın tamsayıya yuvarlar |
| modf | Dizinin kesirli ve integral parçalarını ayrı bir dizi olarak döndürür |
| isnan | Her değerin NaN olup olmadığını belirten boolean dizisini döndür |
| isfinite, isinf | Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively |
| cos, cosh, sin, sinh, tan, tanh | Düzenli ve hiperbolik trigonometrik fonksiyonlar |
| arccos, arccosh, arcsin, arcsinh, arctan, arctanh | Ters trigonometrik fonksiyonlar |

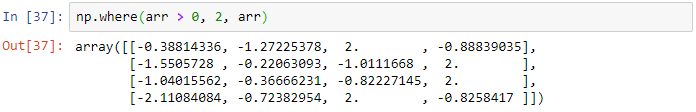
*Tablo: Binary Evrensel Fonksiyonlar*

|  |  |
| --- | --- |
| Fonksiyon | Açıklama |
| add | Dizilere karşılık gelen öğeleri ekler |
| subtract | İkinci dizideki elemanları ilk diziden çıkarır |
| multiply | Dizi elemanlarını çarpar |
| divide, floor\_divide | Böl veya kat böl (kalanı keserek) |
| power | İlk dizideki elemanları, ikinci dizideki güçlere yükseltir |
| maximum, fmax | Max elemanı verir. fmax, NaN'yi dikkate almaz |
| Minimum, fmin | Min elemanı verir. fmax, NaN'yi dikkate almaz |
| mod | Bölümün kalanını verir |
| copysign | İkinci argümandaki değerlerin işaretini ilk argümandaki değerlere kopyalar |
| greater, greater\_equal, less, less\_equal, equal, not\_equal | Sırasıyla; >, >=, <, <=, ==, != operatörlerine eşit |

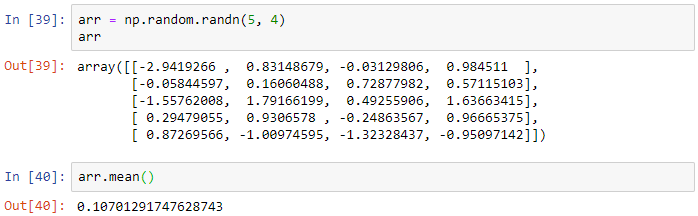
**4.3. Dizi Yöntemli Programlama**

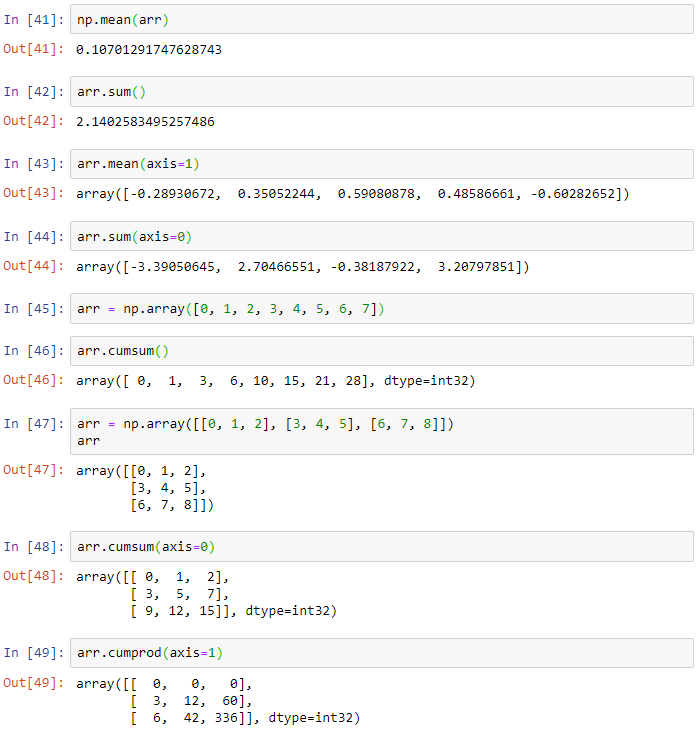


**Koşullu Mantığı Dizi İşlemleri Olarak İfade Etme**



**Matematiksel ve İstatistiksel Yöntemler**

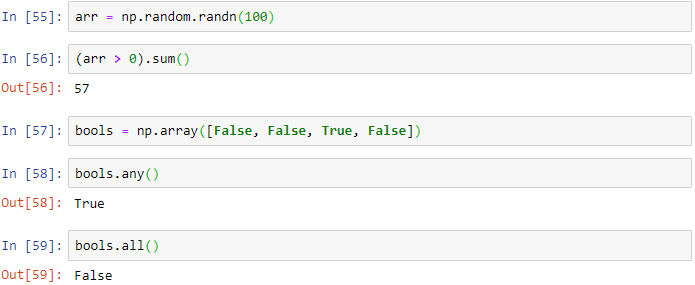




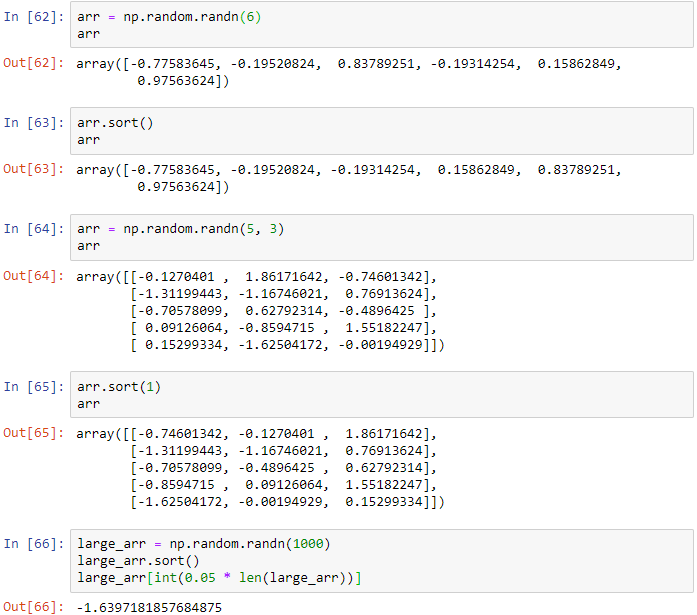
*Tablo: Temel İstatiksel Dizi Yöntemleri*

|  |  |
| --- | --- |
| Metot | Açıklama |
| sum | Dizideki veya bir eksen boyunca tüm öğelerin toplamını döndürür |
| mean | Aritmetik ortalama döndürür. Sıfır uzunluklu diziler NaN ortalamalarına sahiptir |
| std, var | İsteğe bağlı serbestlik dereceleriyle birlikte sırasıyla standart sapma ve varyans (varsayılan payda n) |
| min, max | Minimum, Maximum |
| argmin, argmax | Sırasıyla minimum ve maksimum elemanların indisleri |
| cumsum | 0'dan başlayan öğelerin kümülatif toplamı |
| cumprod | 1'den başlayan elementlerin kümülatif sonucu |

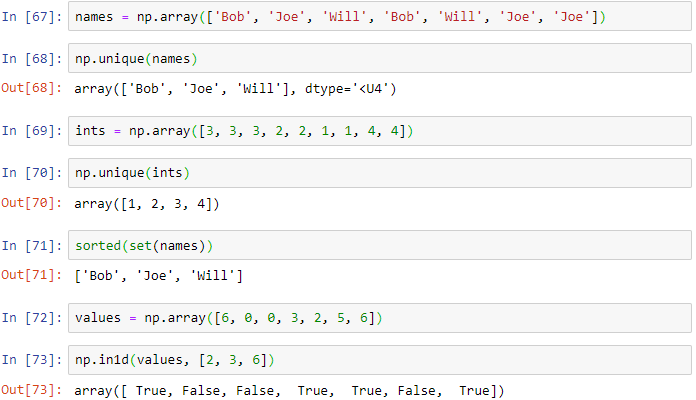
**Boolean Diziler İçin Metotlar**



**Sıralama (Sorting)**



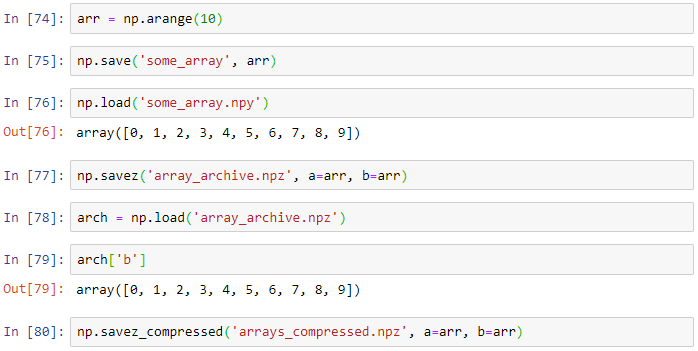
**Unique ve Diğer Küme Mantıkları**



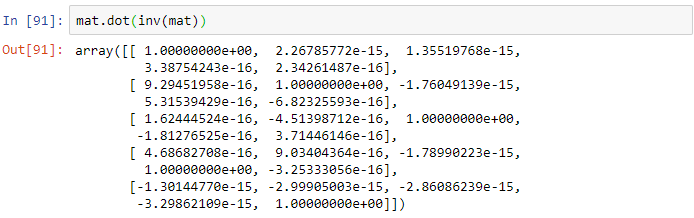
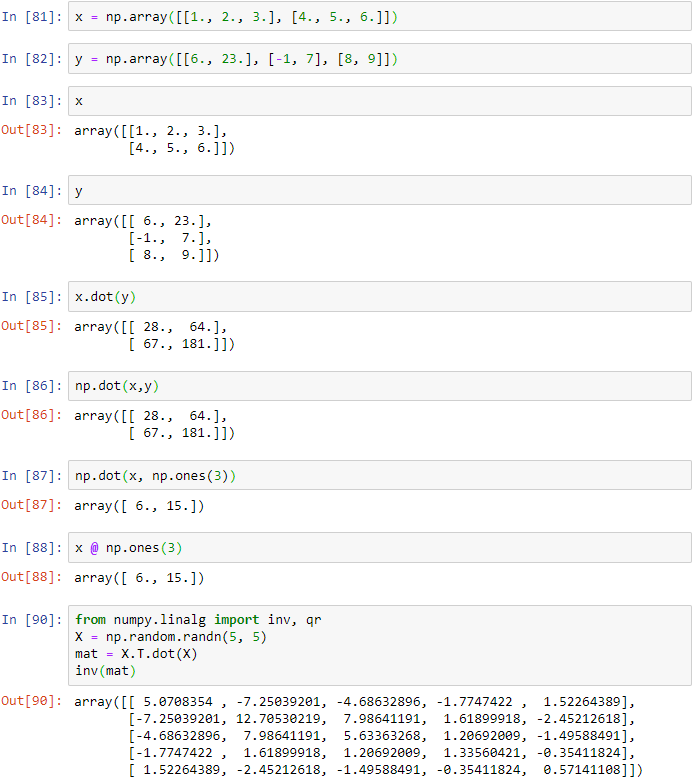
*Tablo: Dizi Kümesi İşlemleri*

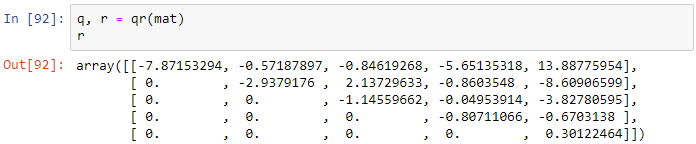
|  |  |
| --- | --- |
| Metot | Açıklama |
| unique(x) | x'deki sıralanmış, benzersiz öğeleri hesaplar |
| intersect1d(x, y) | x ve y'deki sıralanmış, ortak öğeleri hesaplar |
| union1d(x, y) | Öğelerin sıralanmış birleşimini hesaplayın |
| in1d(x, y) | Her x öğesinin y'de olup olmadığını gösteren bir boolean dizisi hesaplar |
| setdiff1d(x, y) | x'deki öğelerin y'de olmayanlarını alır |
| setxor1d(x, y) | Simetrik farklar; dizilerden herhangi birinde bulunan, ancak her ikisinde olmayan öğeleri alır |

**4.4. Dizilerle Dosya Girişi ve Çıkışı**



**4.5. Doğrusal Cebir**

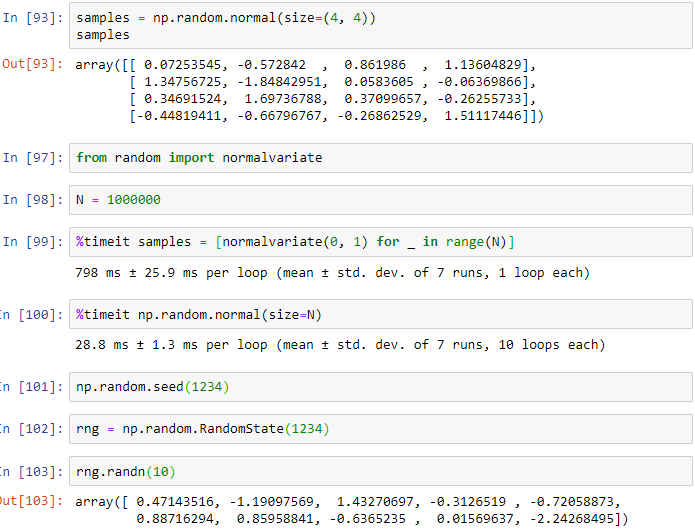




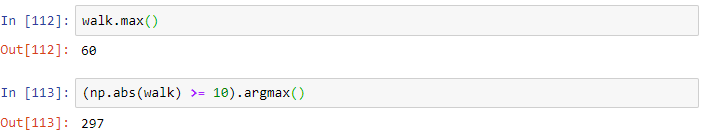
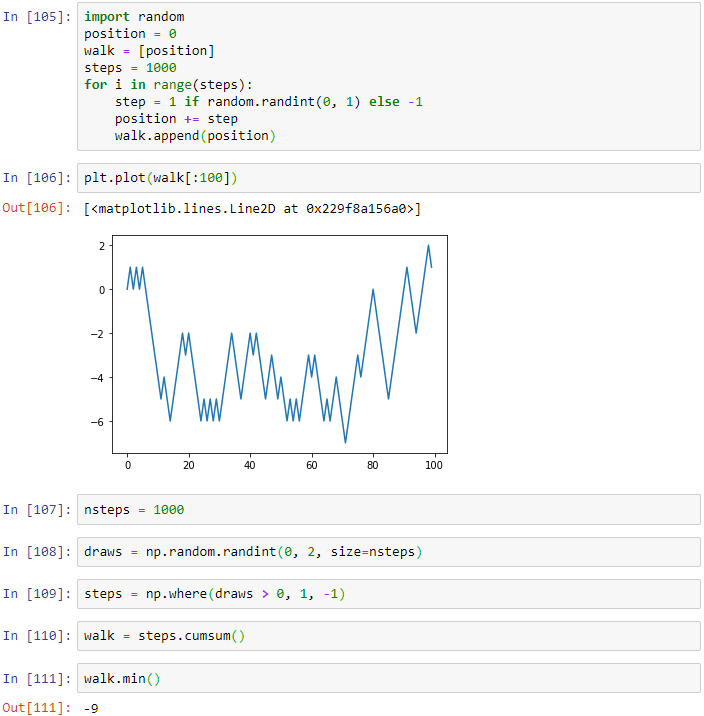
*Tablo: Sık kullanılan numpy.linalg fonksiyonları*

|  |  |
| --- | --- |
| Fonksiyon | Açıklama |
| diag | Bir kare matrisin köşegen (veya çapraz-dışı) öğelerini 1D dizisi olarak döndürür veya 1D dizisini, köşegen dışında sıfırlara sahip bir kare matrise dönüştürür |
| dot | Matris çarpımı |
| trace | Çapraz elemanların toplamını hesaplar |
| det | Matris determinantını hesaplar |
| eig | Bir kare matrisin özdeğerlerini ve özvektörlerini hesaplar |
| inv | Bir kare matrisin tersini hesaplar |
| pinv | Moore-Penrose yalancı bir matrisin tersini hesaplar |
| qr | QR ayrışmasının hesaplanması |
| svd | Tekil değer ayrışımını (SVD) hesaplar |
| solve | Ax = b doğrusal sistemini çözer, burada A: kare matris |
| listsq | En küçük kareler çözümünü Ax = b olarak hesaplar |

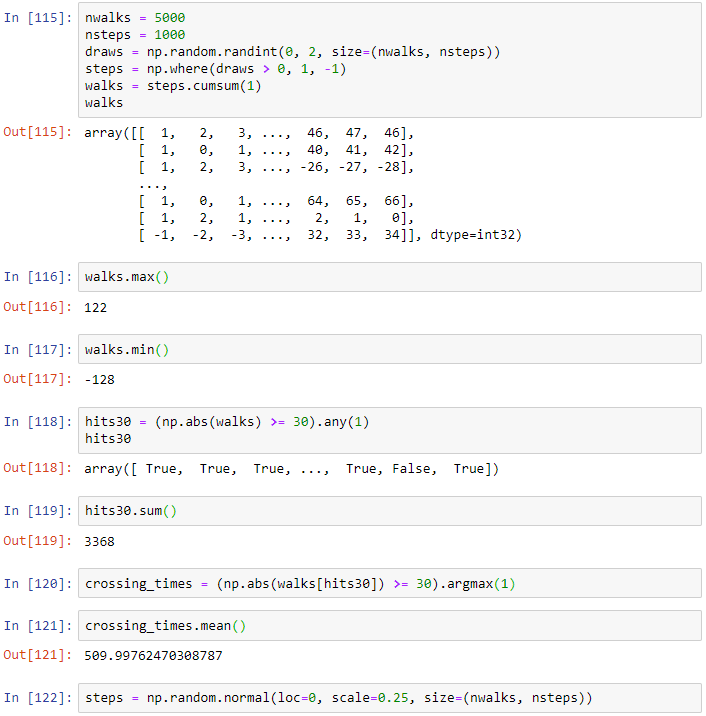
**4.6. Sözde Sayı Üretimi (Pseudorandom Number Generation)**



**4.7. Örnek: Rastgele Yürüyüşler**



**Aynı Anda Birçok Rastgele Yürüyüşü Simüle Etme**



**BÖLÜM 5**

**PANDAS**

Veri bilimi projeleri, verinin keşfedilmesi ve temizlenmesi ile başlar ve bu işlemler projelerin en çok zaman alan kısımlarıdır. Dolayısıyla verinin keşfi ve temizlenmesi sırasında işleri kolaylaştıracak bir takım kütüphanelere ihtiyaç duyulur. Numpy’ın eksik kaldığı kısımlardaki eksiği Pandas tamamlıyor. Ancak Pandas Numpy’ın bir alternatifi olarak değil, uzantısı olarak düşünülmelidir. Pandas, Numpy’ın sütun adları ve homojen olmayan verilerle çalışamama gibi eksik kaldığı kısımlara ve daha fazlasına çözümler üretir. Pandas ile veri analizi yaparken kullanılan temel veri yapıları Seriler ve DataFrame’lerdir.

Pandas kütüphanesinin projemize import edilmesi:

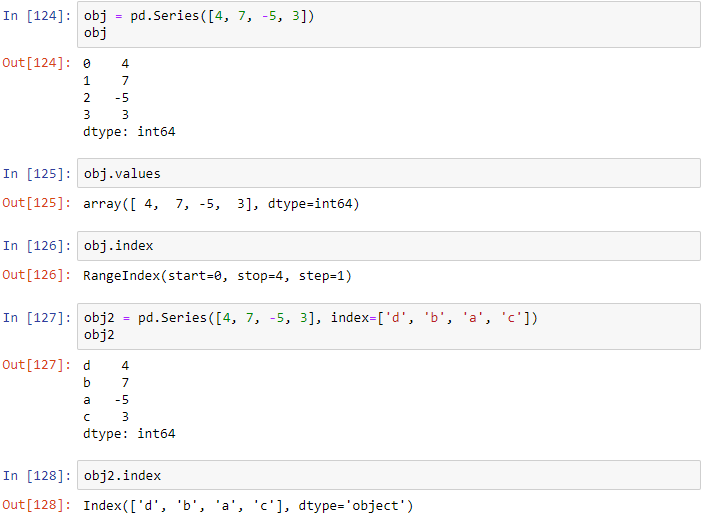


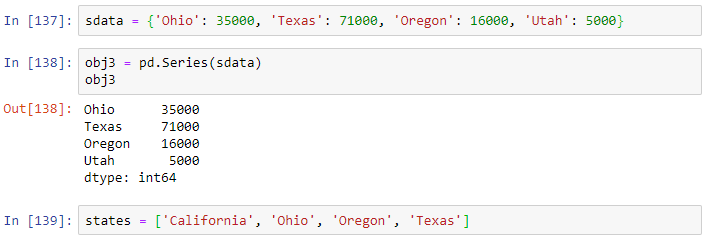
**5.1 Pandas Veri Yapıları**

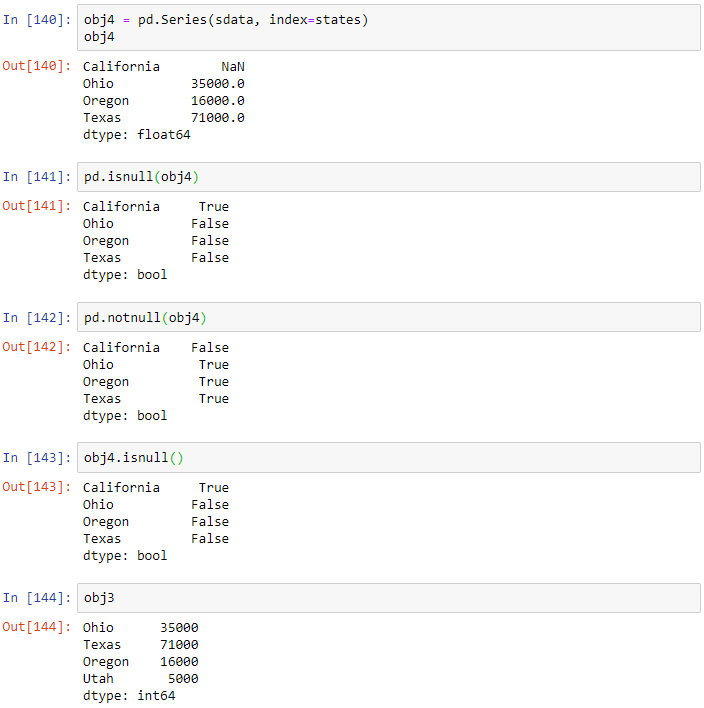
Pandas’ın iki güçlü veri yapısı vardır: Series ve DataFrame. Her sorun için evrensel bir çözüm olmasalar da çoğu uygulama için sağlam, kullanımı kolay bir temel sağlarlar.

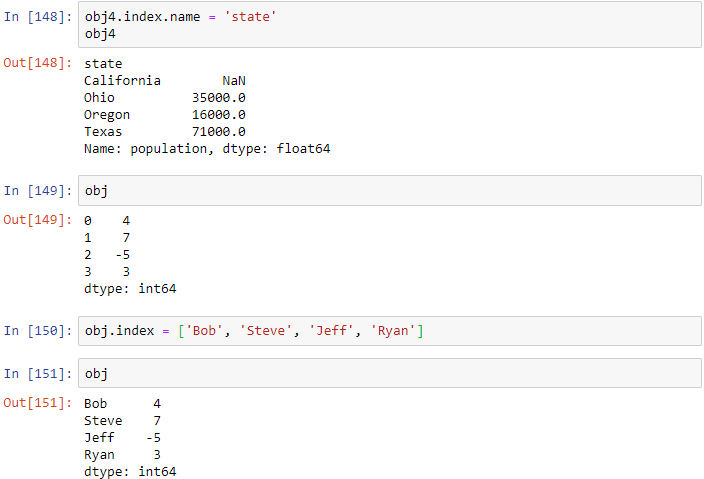
**Seriler (Series)**

Seriler Numpy dizileri baz alınarak oluşturulmuştur. Dolayısıyla tek boyutlu Numpy dizilerine çok benzerler.



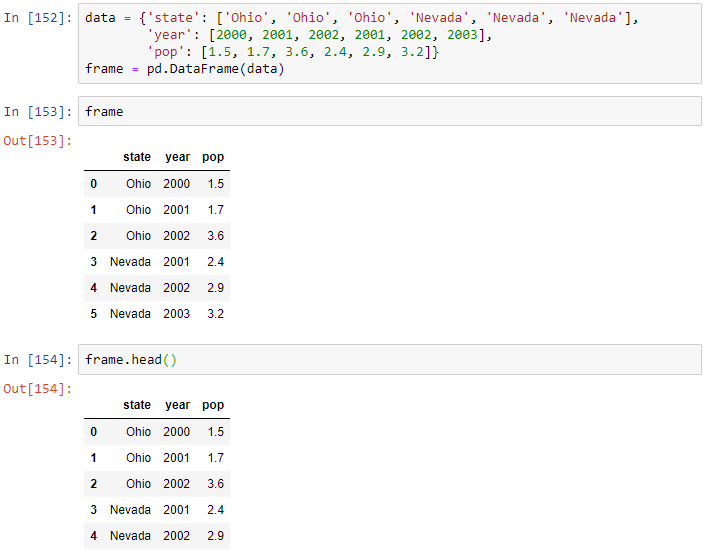


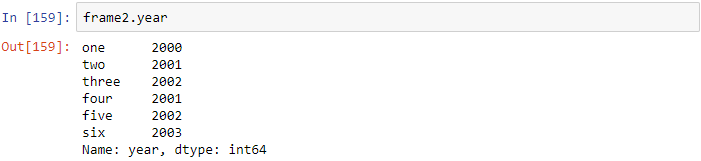
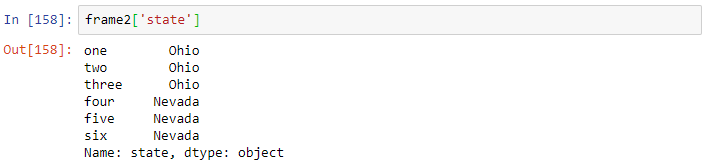
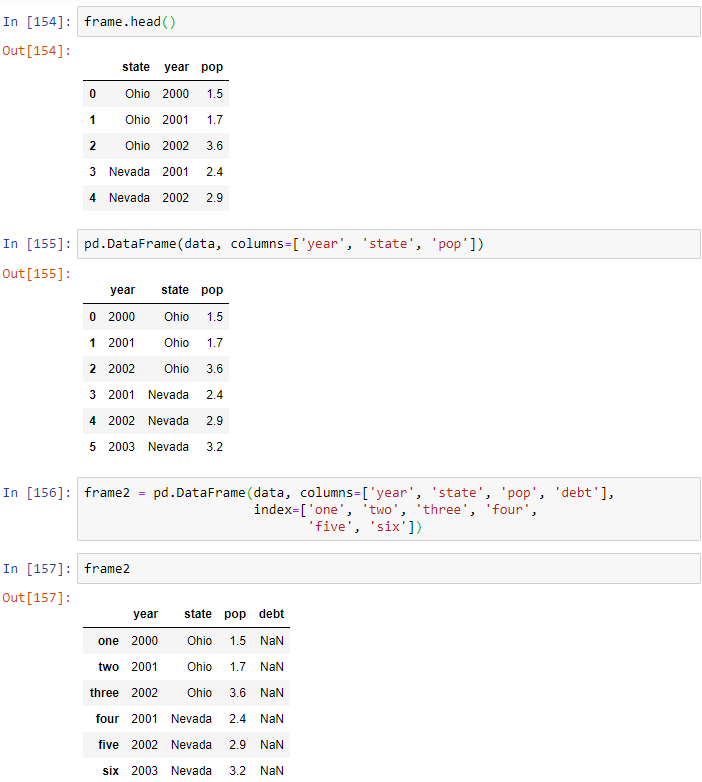


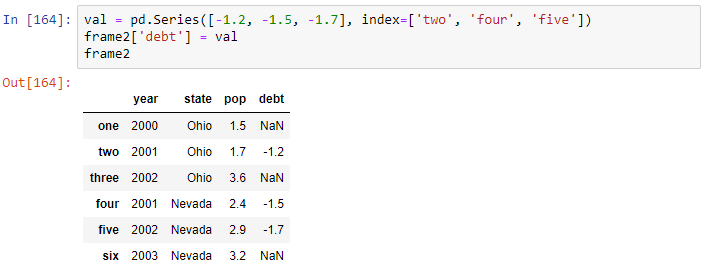
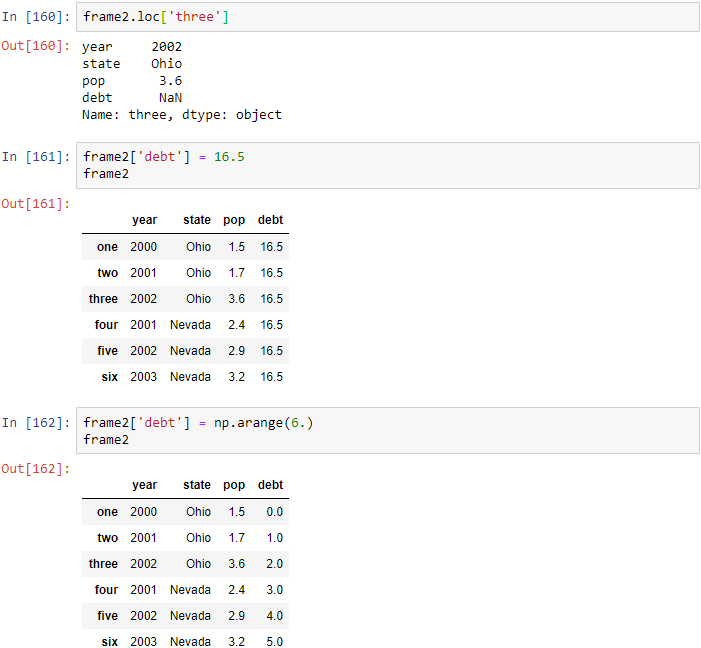


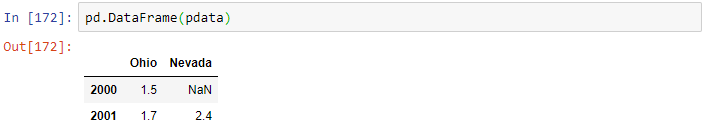
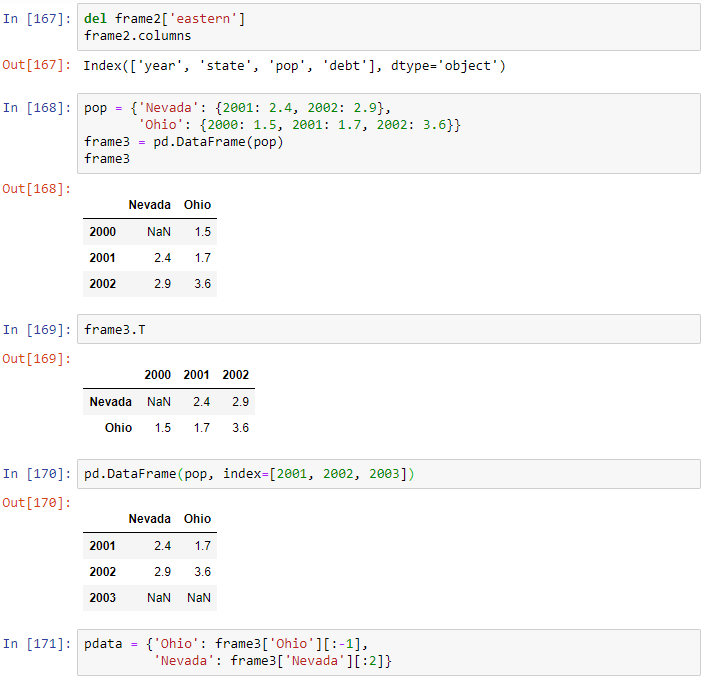
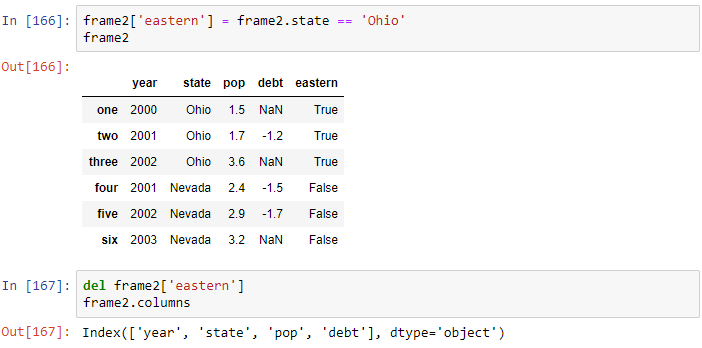
**DataFrame**

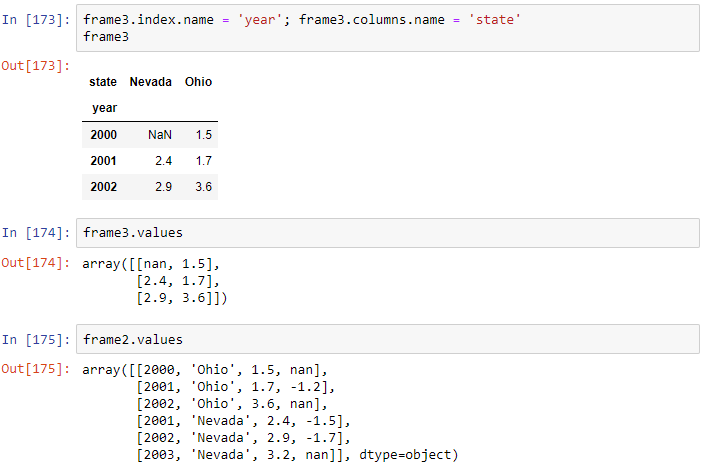
DataFrame dikdörtgen bir veri tablosunu temsil eder ve her biri farklı bir değer türü (sayısal, string, boolean, vb.) olabilen düzenli bir sütun koleksiyonu içerir. DataFrame'de hem satır hem de sütun dizini bulunur. Hepsi aynı endeksi paylaşan bir seri dizisi olarak düşünülebilir.





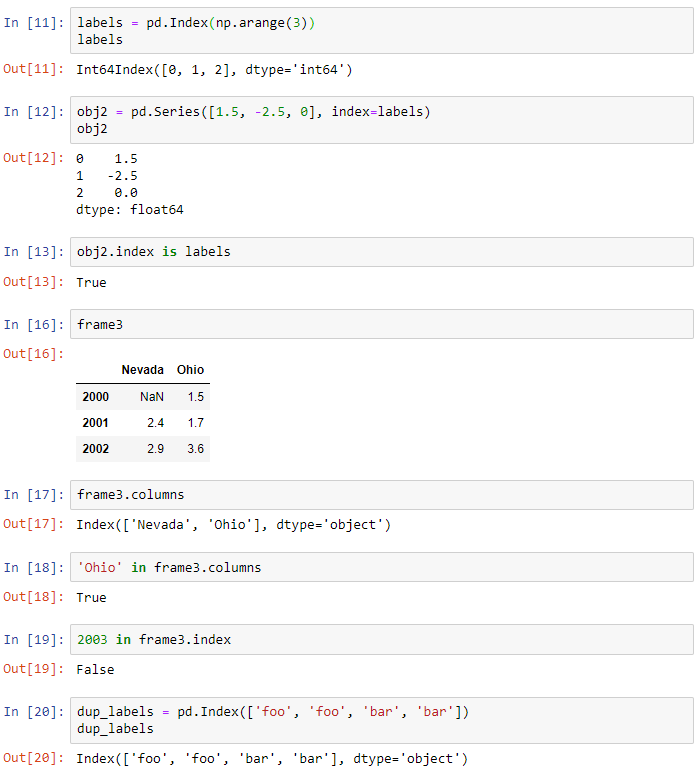






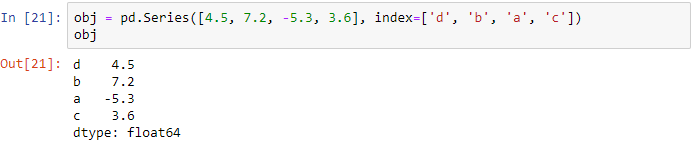
**Index Objects**

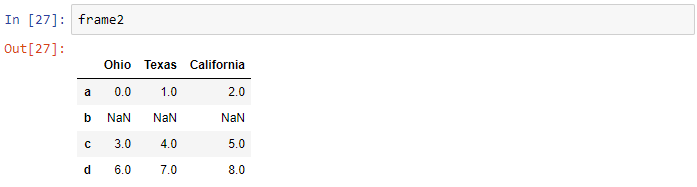
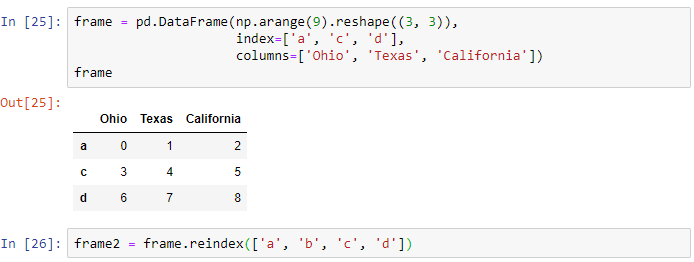
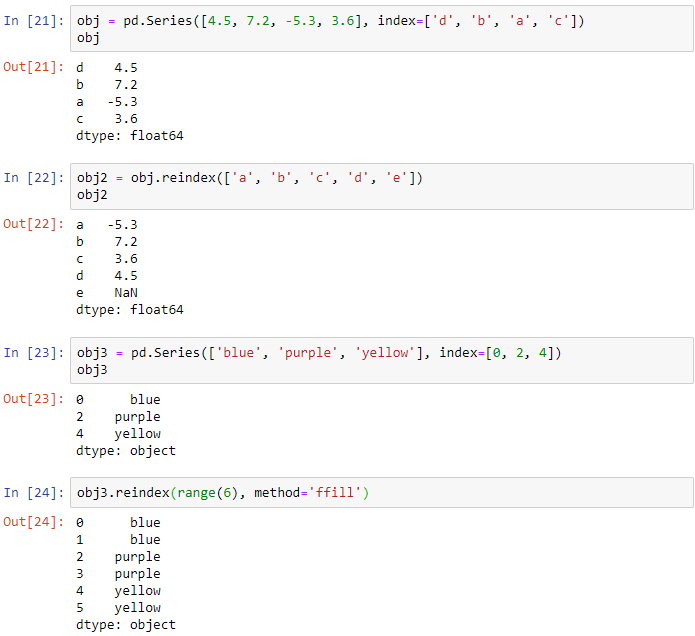


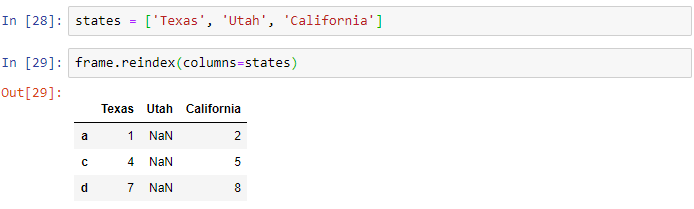


**5.2. Temel İşlevsellik**

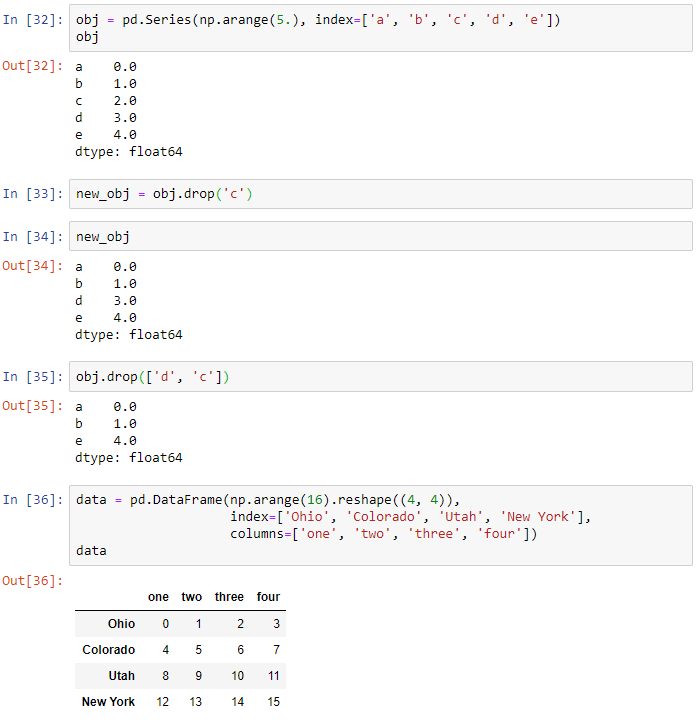
**Reindexing (Yeniden endeksleme)**





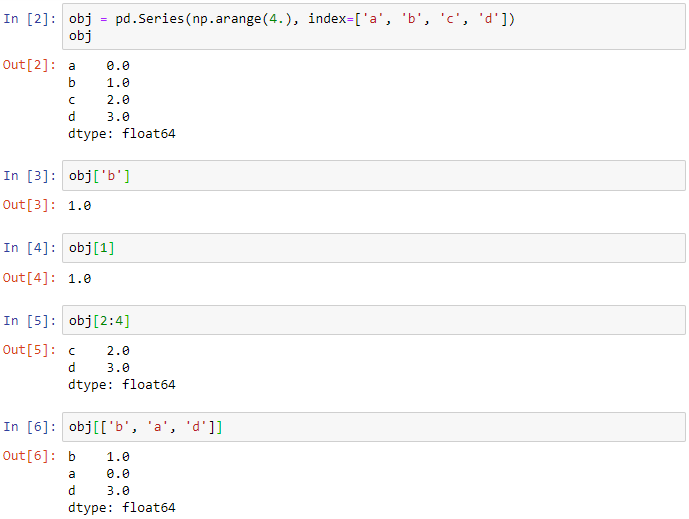


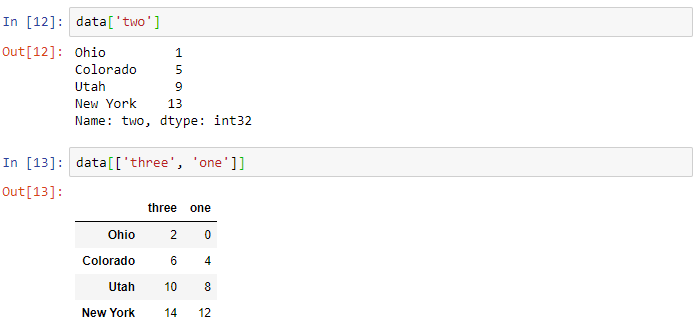
**Bir Eksenden Girişleri Bırakma**

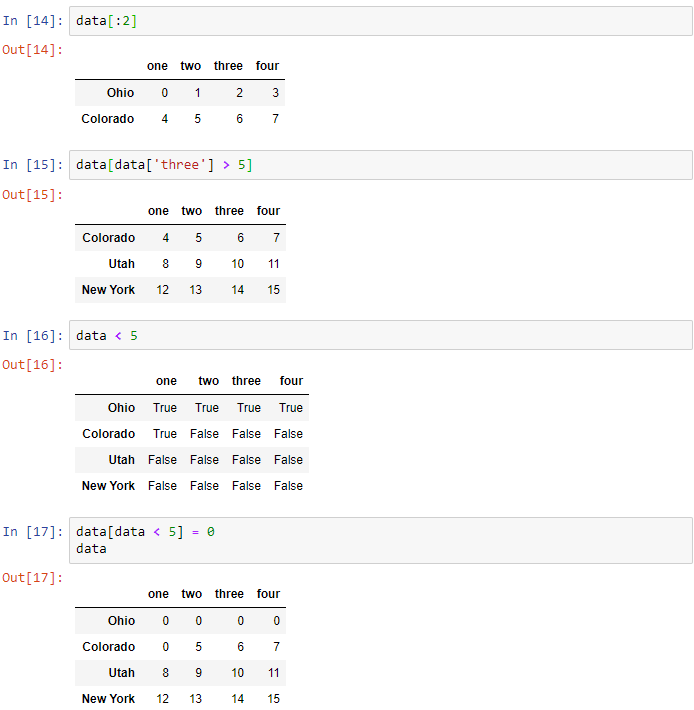




**Dizin Oluşturma, Seçim ve Filtreleme**

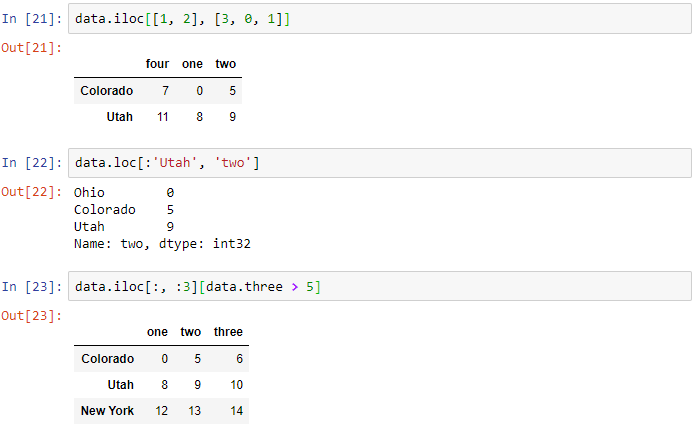




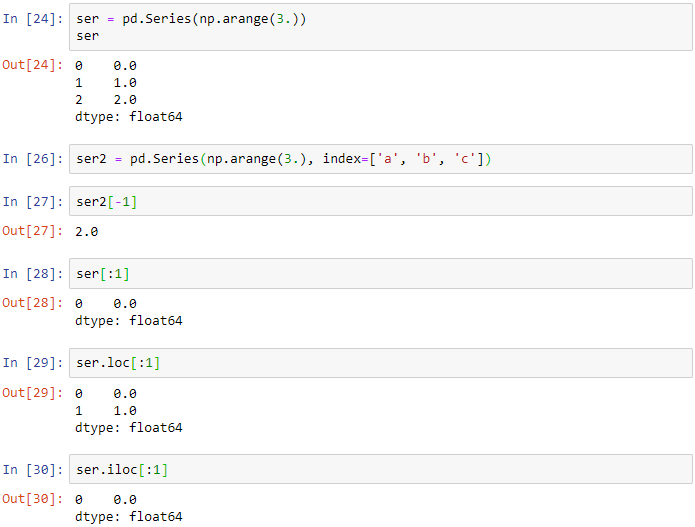


**loc ve iloc ile seçim**

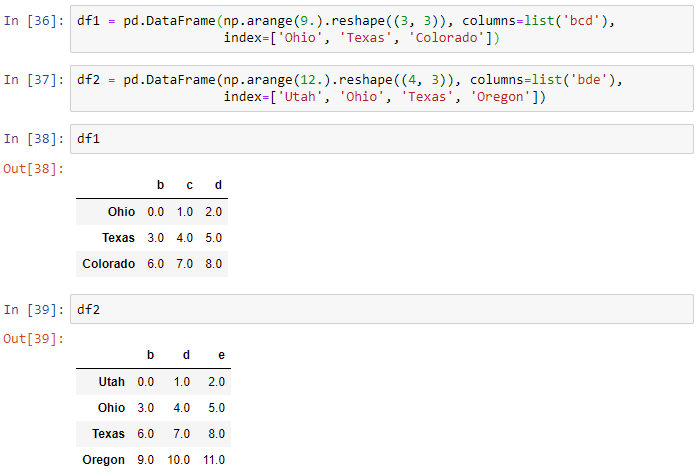
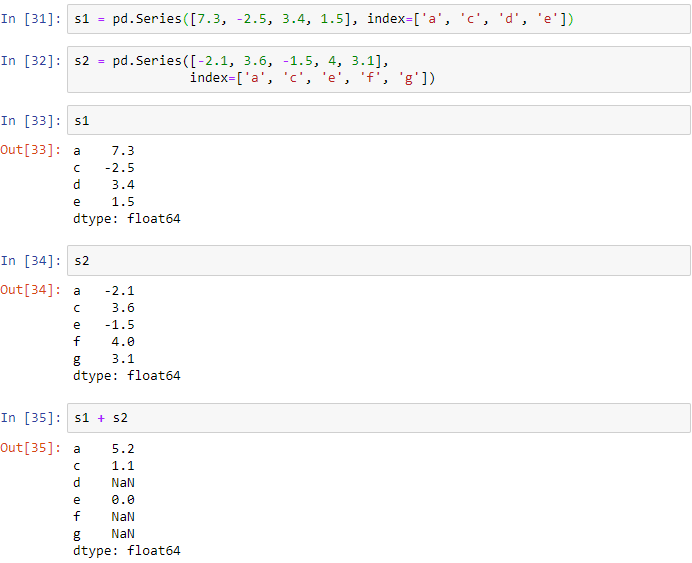


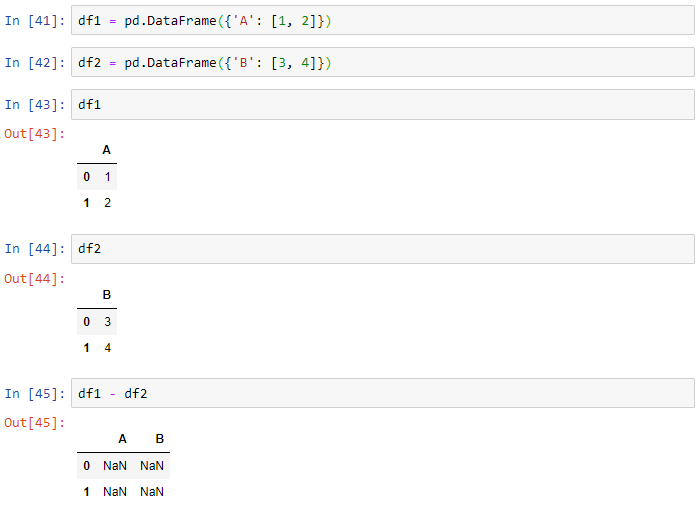
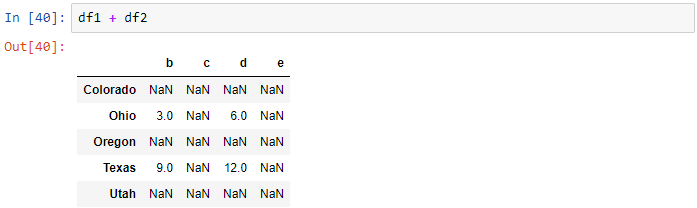


**Tam Sayılı Dizinler**

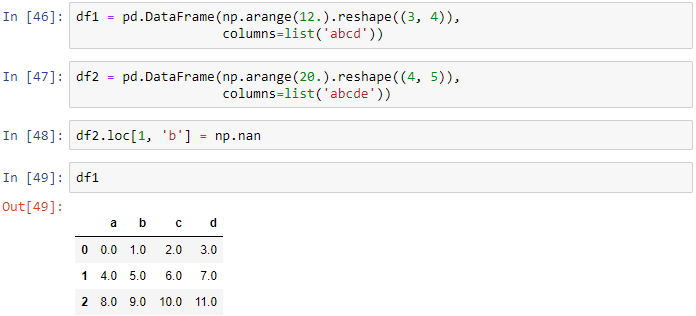


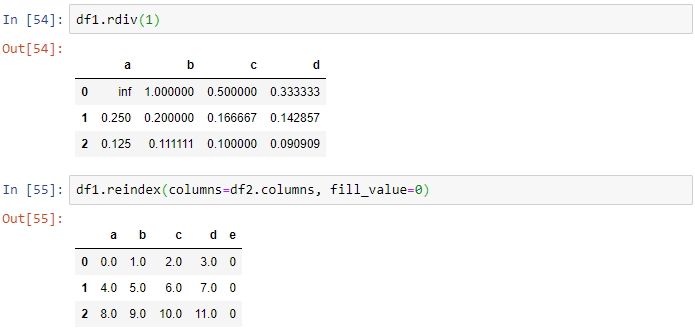
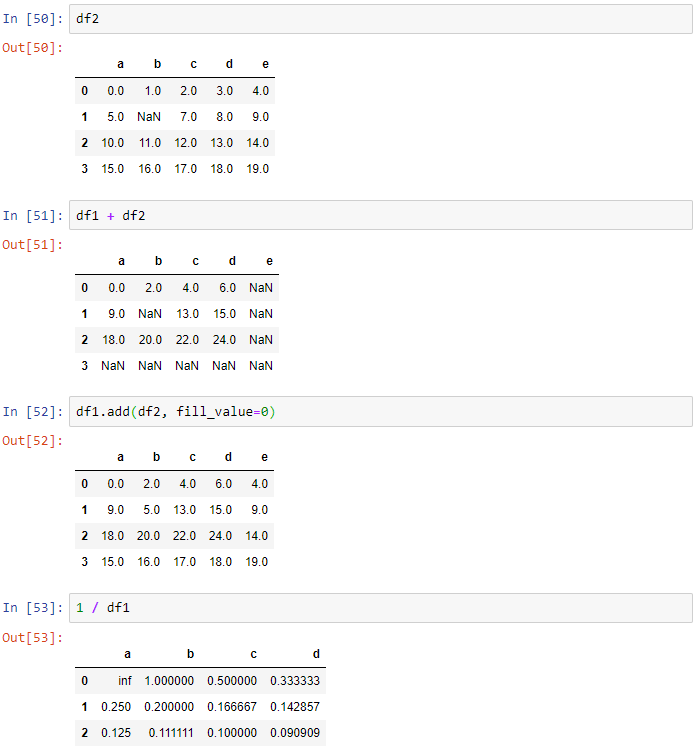
**Aritmetik ve Veri Hizalama**



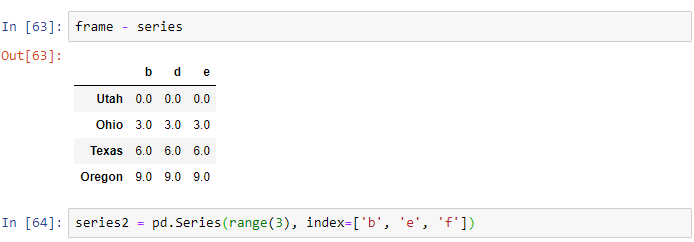
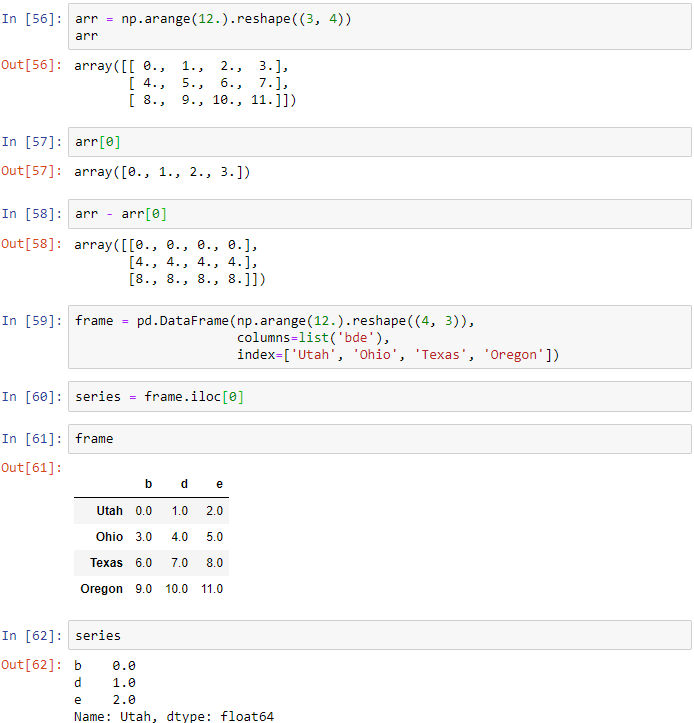


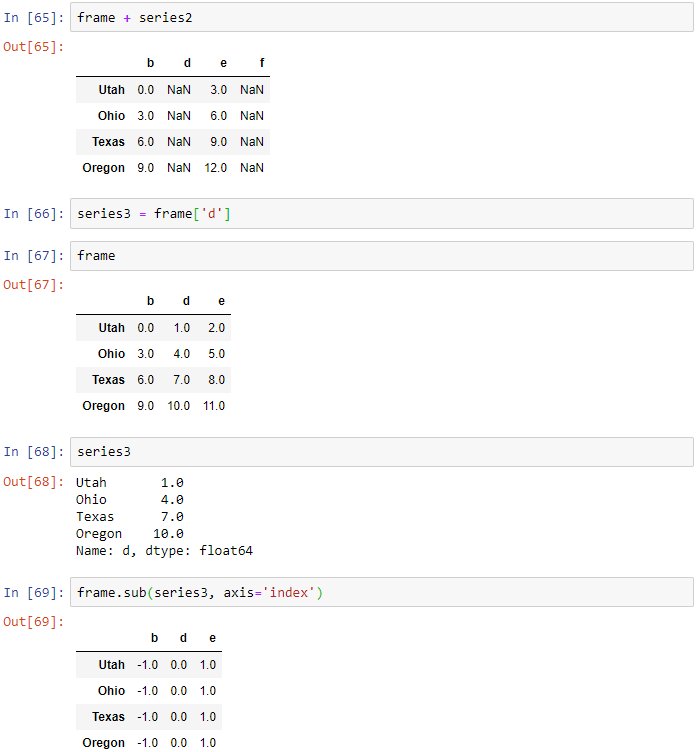
**Doldurma Değerler ile Aritmetik Yöntemler**



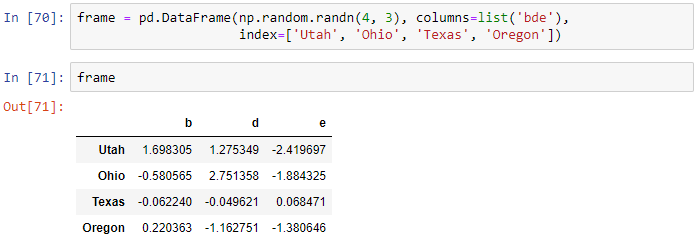


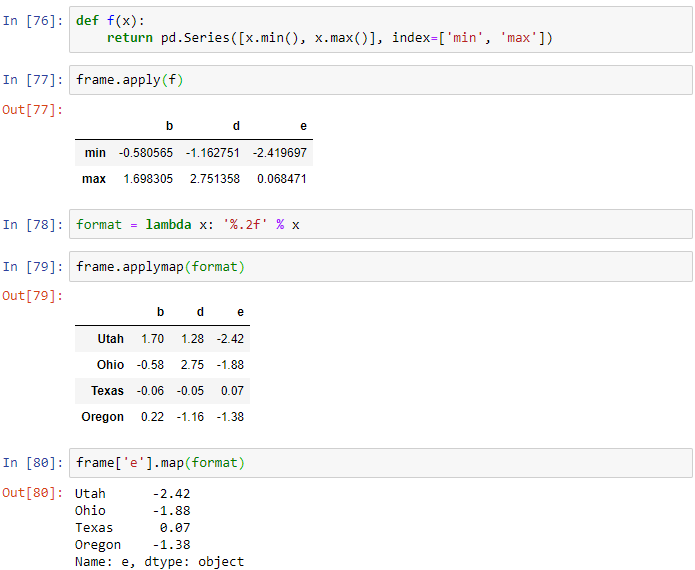
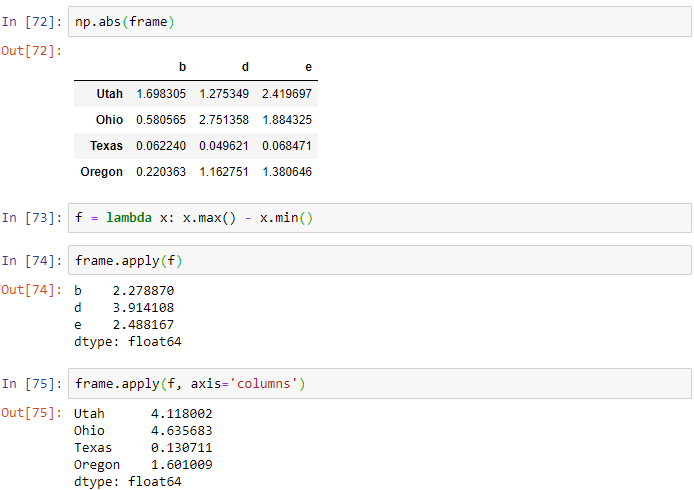
**DataFrame ve Series Arasındaki İşlemler**



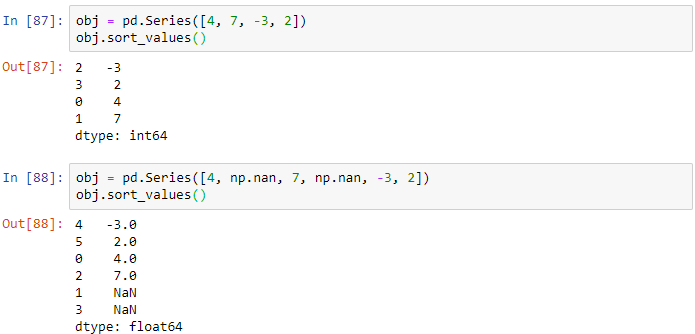
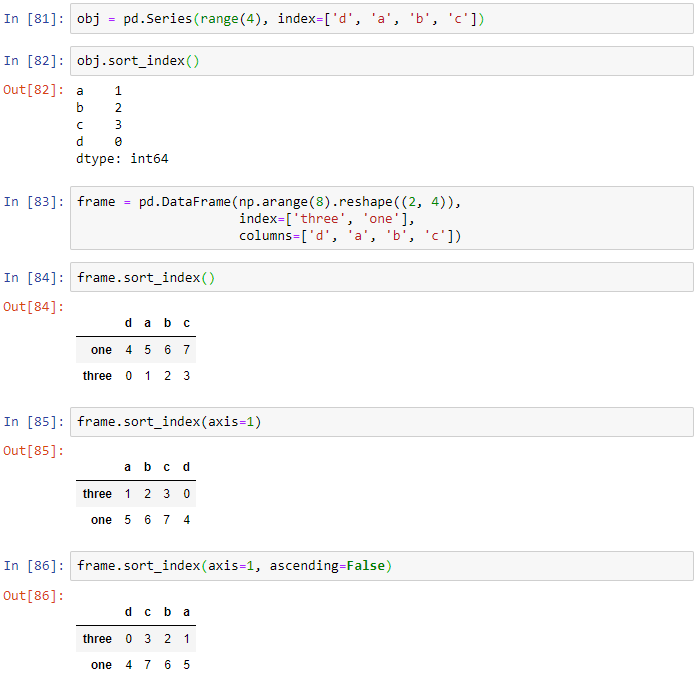


**Fonksiyon Uygulaması ve Haritalama**

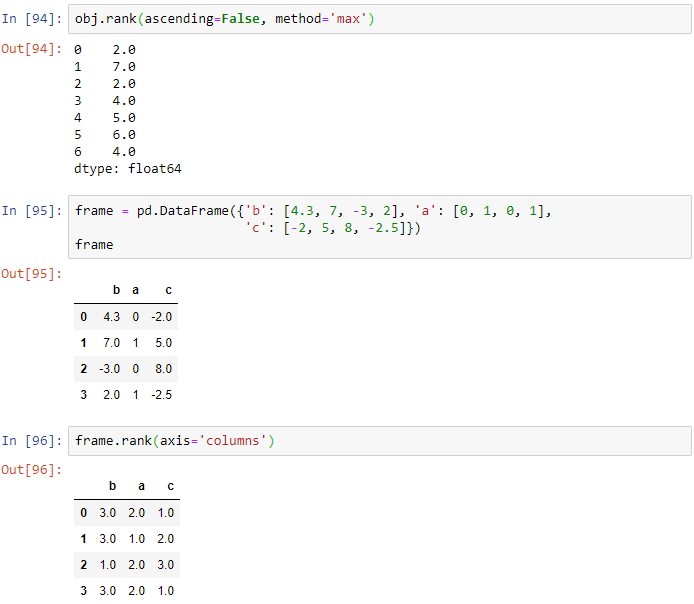




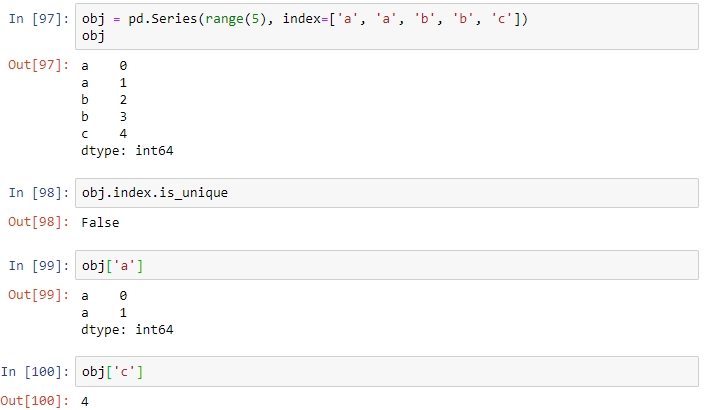
**Sorting and Ranking**

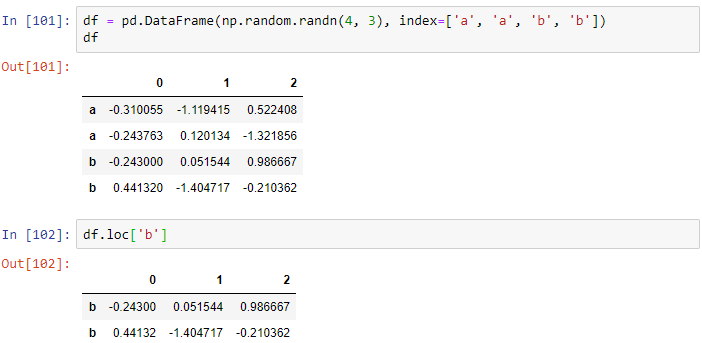




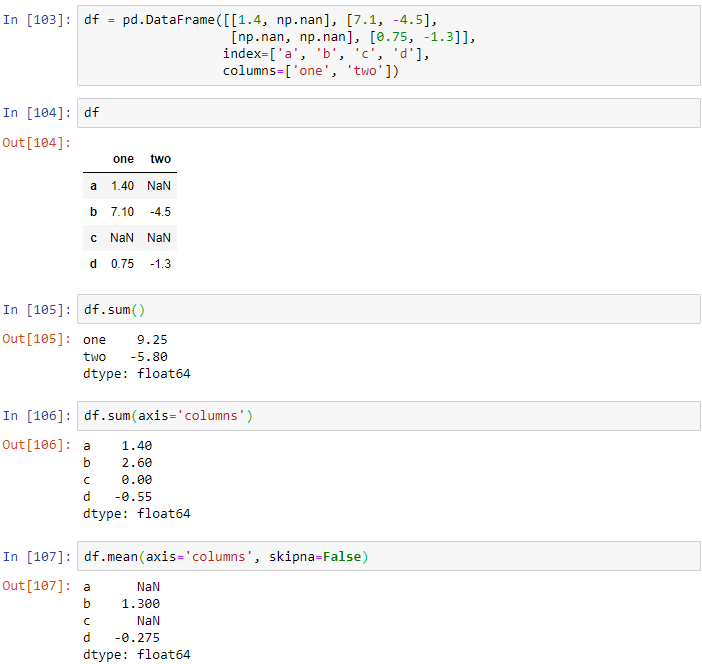


**Çift Etiketli Eksen Dizinleri**



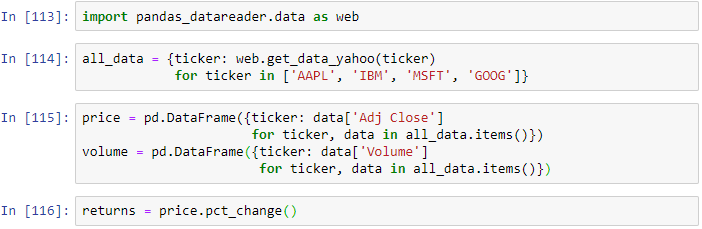


**5.3. Tanımlayıcı İstatistiğin Özetlenmesi ve Hesaplanması**





**Korelasyon ve Kovaryans**





**Benzersiz Değerler, Değer Sayıları ve Üyelik**

