Bu eğitimde, Python'da içe aktarma işlemlerini öğrenecek, bilmediğiniz kütüphanelerle (ve döndürdükleri nesnelerle) çalışmak için bazı ipuçları alacak ve operatör aşırı yüklemesini derinlemesine inceleyeceksiniz.

Imports

Şimdiye kadar dile yerleşik olan tipler ve fonksiyonlardan bahsettik.

Ancak Python'ın en iyi yanlarından biri (özellikle bir veri bilimcisiyseniz), Python için yazılmış çok sayıda yüksek kaliteli özel kütüphane olmasıdır.

Bu kütüphanelerin bazıları "standart kütüphane" içindedir, yani Python'ı çalıştırdığınız her yerde bulabilirsiniz. Diğer kütüphaneler, her zaman Python ile birlikte gelmeseler bile kolayca eklenebilir.

Her iki durumda da, bu koda içe aktarma işlemleriyle erişeceğiz.

Örneğimize, standart kütüphaneden math'i içe aktararak başlayacağız.

import math

print("It's math! It has type **{}**".format(type(math)))

It's math! It has type <class 'module'>

Math bir modüldür. Bir modül, başkası tarafından tanımlanmış bir değişkenler koleksiyonudur (isterseniz bir ad alanı da diyebilirsiniz). Math'deki tüm isimleri yerleşik dir() fonksiyonunu kullanarak görebiliriz.

print(dir(math))

['\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

Bu değişkenlere nokta sözdizimini kullanarak erişebiliriz. Bazıları, math.pi gibi basit değerlere atıfta bulunur:

print("pi to 4 significant digits = **{:.4}**".format(math.pi))

pi to 4 significant digits = 3.142

Ancak modülde bulacağımız şeylerin çoğu math.log gibi fonksiyonlardır:

math.log(32, 2)

5.0

Elbette, math.log'un ne işe yaradığını bilmiyorsak, üzerinde help() çağırabiliriz:

help(math.log)

Help on built-in function log in module math:

log(...)

log(x, [base=math.e])

Return the logarithm of x to the given base.

If the base not specified, returns the natural logarithm (base e) of x.

Modülün kendisinde help() fonksiyonunu da çağırabiliriz. Bu, modüldeki tüm fonksiyonlar ve değerler için birleşik dokümantasyonu (ve modülün üst düzey bir açıklamasını) sağlayacaktır. Matematik yardım sayfasının tamamını görmek için "Çıktı" düğmesine tıklayın.

help(math)

### **Other import syntax[¶](https://www.kaggle.com/code/colinmorris/working-with-external-libraries" \l "Other-import-syntax" \t "_self)**

Matematikte fonksiyonları sıklıkla kullanacağımızı biliyorsak, biraz yazma zahmetinden kurtulmak için daha kısa bir takma ad altında içe aktarabiliriz (bu durumda "matematik" zaten oldukça kısadır).

import math as mt

mt.pi

3.141592653589793

Pandas, Numpy, Tensorflow veya Matplotlib gibi bazı popüler kütüphanelerde bunu yapan kodlar görmüş olabilirsiniz. Örneğin, numpy'yi np olarak, pandas'ı ise pd olarak içe aktarmak yaygın bir kuraldır.

as, içe aktarılan modülün adını değiştirir. Aşağıdakine benzer bir işlemdir:

import math

mt = math

Matematik modülündeki tüm değişkenlere tek başlarına başvurabilseydik harika olmaz mıydı? Yani math.pi veya mt.pi yerine sadece pi değişkenine başvurabilseydik? İyi haber: Bunu yapabiliriz.

from math import \*

print(pi, log(32, 2))

3.141592653589793 5.0

import \* komutu, modülün tüm değişkenlerine doğrudan (noktalı önek olmadan) erişmenizi sağlar.

Kötü haber: Bazı puristler bunu yaptığınız için size kızabilir.

Daha kötüsü: Bir bakıma haklılar.

from math import \*

from numpy import \*

print(pi, log(32, 2))

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/3018510453.py in <module>

**1** from math import \*

**2** from numpy import \*

----> 3 print(pi, log(32, 2))

TypeError: return arrays must be of ArrayType

Ne oldu? Daha önce işe yaramıştı!

Bu tür "yıldız içe aktarmaları" bazen tuhaf, hata ayıklaması zor durumlara yol açabilir.

Bu durumdaki sorun, hem math hem de numpy modüllerinin log adlı fonksiyonlara sahip olması, ancak farklı anlamlara sahip olmalarıdır. Numpy Second'dan içe aktardığımız için, log değişkeni math'tan içe aktardığımız log değişkeninin üzerine yazar (veya "gölgeler").

İyi bir uzlaşma, her modülden yalnızca ihtiyaç duyacağımız belirli şeyleri içe aktarmaktır:

from math import log, pi

from numpy import asarray

### **Submodules**

Modüllerin, fonksiyonlara veya değerlere atıfta bulunabilen değişkenler içerdiğini gördük. Dikkat edilmesi gereken bir nokta da, bunların diğer modüllere atıfta bulunan değişkenlere de sahip olabileceğidir.

import numpy

print("numpy.random is a", type(numpy.random))

print("it contains names such as...",

dir(numpy.random)[-15:]

)

numpy.random is a <class 'module'>

it contains names such as... ['seed', 'set\_state', 'shuffle', 'standard\_cauchy', 'standard\_exponential', 'standard\_gamma', 'standard\_normal', 'standard\_t', 'test', 'triangular', 'uniform', 'vonmises', 'wald', 'weibull', 'zipf']

Yani numpy'ı yukarıdaki gibi içe aktarırsak, rastgele "alt modül"deki bir fonksiyonu çağırmak için iki noktaya ihtiyaç duyulacaktır.

*# Roll 10 dice*

rolls = numpy.random.randint(low=1, high=6, size=10)

rolls

array([3, 4, 3, 4, 5, 5, 2, 1, 3, 3])

## **Oh the places you'll go, oh the objects you'll see[¶](https://www.kaggle.com/code/colinmorris/working-with-external-libraries" \l "Oh-the-places-you'll-go,-oh-the-objects-you'll-see" \t "_self)**

Yani 6 dersten sonra, tam sayılar, kayan noktalı sayılar, bool'lar, listeler, dizeler ve sözlükler konusunda uzman olacaksınız (değil mi?).

Bu doğru olsa bile, iş burada bitmiyor. Özel görevler için çeşitli kütüphanelerle çalışırken, bunların kendi tiplerini tanımladıklarını ve bunlarla nasıl çalışacağınızı öğrenmeniz gerektiğini göreceksiniz. Örneğin, matplotlib grafik kütüphanesiyle çalışırsanız, Alt Grafikler, Şekiller, İşaretler ve Açıklamaları temsil eden, kütüphanenin tanımladığı nesnelerle karşılaşacaksınız. Pandas fonksiyonları size Veri Çerçeveleri ve Seriler sunacaktır.

Bu bölümde, garip tiplerle çalışmak için hızlı bir hayatta kalma rehberi paylaşmak istiyorum.

## **Three tools for understanding strange objects[¶](https://www.kaggle.com/code/colinmorris/working-with-external-libraries" \l "Three-tools-for-understanding-strange-objects" \t "_self)**

Yukarıdaki hücrede, bir numpy fonksiyonunu çağırdığımızda bir "dizi" elde ettiğimizi gördük. Daha önce hiç böyle bir şey görmemiştik (en azından bu kursta). Ama panik yapmayın: Burada bize yardımcı olacak üç tanıdık yerleşik fonksiyonumuz var.

**1: type()** (what is this thing?)

type(rolls)

numpy.ndarray

**2: dir()** (what can I do with it?)

print(dir(rolls))

,

['T', '\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_array\_\_', '\_\_array\_finalize\_\_', '\_\_array\_function\_\_', '\_\_array\_interface\_\_', '\_\_array\_prepare\_\_', '\_\_array\_priority\_\_', '\_\_array\_struct\_\_', '\_\_array\_ufunc\_\_', '\_\_array\_wrap\_\_', '\_\_bool\_\_', '\_\_class\_\_', '\_\_complex\_\_', '\_\_contains\_\_', '\_\_copy\_\_', '\_\_deepcopy\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_float\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_iand\_\_', '\_\_ifloordiv\_\_', '\_\_ilshift\_\_', '\_\_imatmul\_\_', '\_\_imod\_\_', '\_\_imul\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_int\_\_', '\_\_invert\_\_', '\_\_ior\_\_', '\_\_ipow\_\_', '\_\_irshift\_\_', '\_\_isub\_\_', '\_\_iter\_\_', '\_\_itruediv\_\_', '\_\_ixor\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lshift\_\_', '\_\_lt\_\_', '\_\_matmul\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_neg\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmatmul\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_ror\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_setstate\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', '\_\_xor\_\_', 'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']

*# If I want the average roll, the "mean" method looks promising...*

rolls.mean()

3.3

*#Or maybe I just want to turn the array into a list, in which case I can use "tolist"*

rolls.tolist()

[3, 4, 3, 4, 5, 5, 2, 1, 3, 3]

**3: help()** (tell me more)

*# That "ravel" attribute sounds interesting. I'm a big classical music fan.*

help(rolls.ravel)

Help on built-in function ravel:

ravel(...) method of numpy.ndarray instance

a.ravel([order])

Return a flattened array.

Refer to `numpy.ravel` for full documentation.

See Also

--------

numpy.ravel : equivalent function

ndarray.flat : a flat iterator on the array.

*# Okay, just tell me everything there is to know about numpy.ndarray*

*# (Click the "output" button to see the novel-length output)*

help(rolls)

### **Operator overloading[¶](https://www.kaggle.com/code/colinmorris/working-with-external-libraries" \l "Operator-overloading" \t "_self)**

Aşağıdaki ifadenin değeri nedir?

[3, 4, 1, 2, 2, 1] + 10

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/2144087748.py in <module>

----> 1 [3, 4, 1, 2, 2, 1] + 10

TypeError: can only concatenate list (not "int") to list

Ne kadar da saçma bir soru. Elbette bir hata.

Peki ya...

rolls + 10

array([13, 14, 13, 14, 15, 15, 12, 11, 13, 13])

Python'ın, +, <, in, == veya indeksleme ve dilimleme için köşeli parantezler gibi temel sözdiziminin parçalarının nasıl davranacağını sıkı bir şekilde denetlediğini düşünebiliriz. Ancak aslında, müdahaleci olmayan bir yaklaşım benimser. Yeni bir tür tanımladığınızda, toplama işleminin onun için nasıl işleyeceğini veya o türdeki bir nesnenin başka bir şeye eşit olmasının ne anlama geldiğini seçebilirsiniz.

Liste tasarımcıları, listelerin sayılara eklenmesine izin verilmediğine karar verdi. Numpy dizi tasarımcıları ise farklı bir yol izledi (sayıyı dizinin her bir öğesine ekledi).

Numpy dizilerinin Python operatörleriyle beklenmedik şekilde (veya en azından listelerden farklı olarak) nasıl etkileşime girdiğine dair birkaç örnek daha:

*# At which indices are the dice less than or equal to 3?*

rolls <= 3

array([ True, False, True, False, False, False, True, True, True,

True])

xlist = [[1,2,3],[2,4,6],]

*# Create a 2-dimensional array*

x = numpy.asarray(xlist)

print("xlist = **{}\n**x =**\n{}**".format(xlist, x))

xlist = [[1, 2, 3], [2, 4, 6]]

x =

[[1 2 3]

[2 4 6]]

*# Get the last element of the second row of our numpy array*

x[1,-1]

6

*# Get the last element of the second sublist of our nested list?*

xlist[1,-1]

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/3020169379.py in <module>

**1** # Get the last element of the second sublist of our nested list?

----> 2 xlist[1,-1]

TypeError: list indices must be integers or slices, not tuple

numpy'nin ndarray tipi çok boyutlu verilerle çalışmak için özelleştirilmiştir, bu yüzden indeksleme için kendi mantığını tanımlar ve her boyuttaki indeksi belirtmek için bir tuple ile indeksleme yapmamıza olanak tanır.

### **When does 1 + 1 not equal 2?**[**¶**](https://www.kaggle.com/code/colinmorris/working-with-external-libraries#When-does-1-+-1-not-equal-2?)

İşler bundan daha da tuhaflaşabilir. Derin öğrenme için popüler olarak kullanılan bir Python kütüphanesi olan Tensorflow'u duymuş (hatta kullanmış) olabilirsiniz. Operatör aşırı yüklemesini yoğun bir şekilde kullanır.

import tensorflow as tf

*# Create two constants, each with value 1*

a = tf.constant(1)

b = tf.constant(1)

*# Add them together to get...*

a + b

<tf.Tensor: shape=(), dtype=int32, numpy=2>

a + b 2 değil, (TensorFlow dokümanlarından alıntı yapacak olursak)...

Bir İşlemin çıktılarından birinin sembolik tanıtıcısıdır. Bu işlemin çıktısının değerlerini tutmaz, bunun yerine bu değerleri bir TensorFlow tf.Session'da hesaplamanın bir yolunu sağlar.

Bu tür şeylerin mümkün olduğunu ve kütüphanelerin genellikle operatör aşırı yüklemesini bariz olmayan veya sihirli görünen şekillerde kullanacağını bilmek önemlidir.

Python operatörlerinin tam sayılara, dizelere ve listelere uygulandığında nasıl çalıştığını anlamak, bir TensorFlow Tensörüne, bir Numpy Ndarray'e veya bir Pandas DataFrame'e uygulandığında ne yaptıklarını hemen anlayabileceğiniz anlamına gelmez.

Örneğin, Veri Çerçeveleri'ni biraz tanıdıktan sonra, aşağıdaki gibi bir ifade ilgi çekici ve sezgisel görünmeye başlar:

*# Get the rows with population over 1m in South America*

df[(df['population'] > 10\*\*6) & (df['continent'] == 'South America')]

Peki neden işe yarıyor? Yukarıdaki örnekte yaklaşık 5 farklı aşırı yüklenmiş operatör var. Bu işlemlerin her biri ne işe yarıyor? İşler ters gitmeye başladığında cevabı bilmek faydalı olabilir.

#### **Curious how it all works?**[**¶**](https://www.kaggle.com/code/colinmorris/working-with-external-libraries#Curious-how-it-all-works?)

Hiç bir nesne üzerinde help() veya dir() fonksiyonunu çağırıp, çift alt çizgi içeren tüm o isimlerin ne olduğunu merak ettiniz mi?

print(dir(list))

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

Bunun doğrudan operatör aşırı yüklemesiyle ilgili olduğu ortaya çıkıyor.

Python programcıları, operatörlerin türlerinde nasıl davrandığını tanımlamak istediklerinde, \_\_lt\_\_, \_\_setattr\_\_ veya \_\_contains\_\_ gibi iki alt çizgiyle başlayıp biten özel adlara sahip yöntemler uygulayarak bunu yaparlar. Genellikle, bu çift alt çizgi biçimini izleyen adların Python için özel bir anlamı vardır.

Örneğin, [1, 2, 3] içindeki x ifadesi, aslında perde arkasında \_\_contains\_\_ liste yöntemini çağırmaktadır. (Çok daha çirkin olan) [1, 2, 3].\_\_contains\_\_(x) ile eşdeğerdir.

Daha fazla bilgi edinmek istiyorsanız, bu özel "alt çizgi" yöntemlerinin çoğunu açıklayan Python'ın resmi belgelerine göz atabilirsiniz.

Bu derslerde kendi tiplerimizi tanımlamayacağız (keşke zamanımız olsaydı!), ancak umarım ileride kendi harika, tuhaf tiplerinizi tanımlamanın keyfini yaşarsınız.