Print ve abs gibi fonksiyonları zaten gördünüz ve kullandınız. Ancak Python'da çok daha fazla fonksiyon var ve kendi fonksiyonlarınızı tanımlamak, Python programlamanın önemli bir parçasıdır.

Bu derste, fonksiyonları kullanma ve tanımlama hakkında daha fazla bilgi edineceksiniz.

Getting Help

Önceki derste abs fonksiyonunu gördünüz, peki ya ne işe yaradığını unuttuysanız?

help() fonksiyonu muhtemelen öğrenebileceğiniz en önemli Python fonksiyonudur. help() fonksiyonunu nasıl kullanacağınızı hatırlarsanız, diğer fonksiyonların çoğunu anlamanın anahtarına sahip olursunuz.

İşte bir örnek:

help(round)

Help on built-in function round in module builtins:

round(number, ndigits=None)

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None. Otherwise

the return value has the same type as the number. ndigits may be negative.

help() iki şeyi görüntüler:

round(number, ndigits=None) fonksiyonunun başlığı. Bu durumda, round() fonksiyonunun sayı olarak tanımlayabileceğimiz bir argüman aldığını söyler. Ayrıca, isteğe bağlı olarak ndigits olarak tanımlanabilecek ayrı bir argüman da verebiliriz.

Fonksiyonun ne işe yaradığına dair kısa bir İngilizce açıklama.

Yaygın hata: Bir fonksiyonu ararken, fonksiyonun adını iletmeyi unutmayın, fonksiyonun çağrı sonucunu değil.

round() fonksiyonuna yapılan bir çağrıda help komutunu çalıştırırsak ne olur? Aşağıdaki hücrenin çıktısını görmek için görünür hale getirin.

help(round(-2.01))

Help on int object:

class int(object)

| int([x]) -> integer

| int(x, base=10) -> integer

|

| Convert a number or string to an integer, or return 0 if no arguments

| are given. If x is a number, return x.\_\_int\_\_(). For floating point

| numbers, this truncates towards zero.

|

| If x is not a number or if base is given, then x must be a string,

| bytes, or bytearray instance representing an integer literal in the

| given base. The literal can be preceded by '+' or '-' and be surrounded

| by whitespace. The base defaults to 10. Valid bases are 0 and 2-36.

| Base 0 means to interpret the base from the string as an integer literal.

| >>> int('0b100', base=0)

| 4

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_ceil\_\_(...)

| Ceiling of an Integral returns itself.

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floor\_\_(...)

| Flooring an Integral returns itself.

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(self, format\_spec, /)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(self, /)

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

| Rounding an Integral returns itself.

| Rounding with an ndigits argument also returns an integer.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

Python, bu tür bir ifadeyi içten dışa doğru değerlendirir. Önce round(-2.01) değerini hesaplar, ardından bu ifadenin çıktısı için yardım sağlar.

(Ve tam sayılar hakkında söylenecek çok şey olduğu ortaya çıktı! Python'daki nesneler, metotlar ve öznitelikler hakkında daha sonra konuştuğumuzda, yukarıdaki yardım çıktısı daha anlamlı hale gelecektir.)

round, kısa bir belge dizisine sahip çok basit bir fonksiyondur. help, print gibi daha karmaşık ve yapılandırılabilir fonksiyonlarla uğraşırken daha da öne çıkar. Aşağıdaki çıktı anlaşılmaz görünüyorsa endişelenmeyin... şimdilik, bu yardımdan yeni bir şey çıkarabiliyor musunuz bir bakın.

help(print)

Help on built-in function print in module builtins:

print(...)

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Eğer bunu arıyorsanız, print'in sep adında bir argüman alabileceğini ve bunun, yazdırdığımızda diğer tüm argümanların arasına ne koyduğumuzu tanımladığını öğrenebilirsiniz.

## **Defining functions**

Yerleşik işlevler harikadır, ancak kendi işlevlerimizi tanımlamaya başlamadan önce bunlarla ancak bir yere kadar ilerleyebiliriz. Aşağıda basit bir örnek verilmiştir.

def least\_difference(a, b, c):

diff1 = abs(a - b)

diff2 = abs(b - c)

diff3 = abs(a - c)

return min(diff1, diff2, diff3)

Bu, a, b ve c olmak üzere üç argüman alan en az\_farklılık adlı bir fonksiyon oluşturur.

Fonksiyonlar, def anahtar sözcüğüyle tanıtılan bir başlıkla başlar. : işaretini izleyen girintili kod bloğu, fonksiyon çağrıldığında çalıştırılır.

return, fonksiyonlarla benzersiz şekilde ilişkilendirilen başka bir anahtar sözcüktür. Python bir return ifadesiyle karşılaştığında, fonksiyondan hemen çıkar ve sağ taraftaki değeri çağıran bağlama iletir.

least\_difference() fonksiyonunun kaynak koddan ne yaptığı açık mı? Emin değilsek, birkaç örnek üzerinde deneyebiliriz:

print(

least\_difference(1, 10, 100),

least\_difference(1, 10, 10),

least\_difference(5, 6, 7), *# Python allows trailing commas in argument lists. How nice is that?*

)

9 0 1

Ya da belki help() fonksiyonu bize bu konuda bir şeyler söyleyebilir.

help(least\_difference)

Help on function least\_difference in module \_\_main\_\_:

least\_difference(a, b, c)

Python, kodumu okuyup güzel bir İngilizce açıklamaya dönüştürecek kadar akıllı değil. Ancak bir fonksiyon yazdığımda, docstring adı verilen bir alanda bir açıklama sunabiliyorum.

### **Docstrings**

def least\_difference(a, b, c):

*"""Return the smallest difference between any two numbers*

*among a, b and c.*

*>>> least\_difference(1, 5, -5)*

*4*

*"""*

diff1 = abs(a - b)

diff2 = abs(b - c)

diff3 = abs(a - c)

return min(diff1, diff2, diff3)

docstring, bir fonksiyonun başlığından hemen sonra gelen üç tırnaklı bir dizedir (birden fazla satıra yayılabilir). Bir fonksiyonda help() çağırdığımızda, docstring gösterilir.

help(least\_difference)

Help on function least\_difference in module \_\_main\_\_:

least\_difference(a, b, c)

Return the smallest difference between any two numbers

among a, b and c.

>>> least\_difference(1, 5, -5)

4

Not: Belge dizisinin son iki satırı, örnek bir fonksiyon çağrısı ve sonucudur. (>>>, Python etkileşimli kabuklarında kullanılan komut istemine bir referanstır.) Python örnek çağrıyı çalıştırmaz; sadece okuyucunun faydalanması için oradadır. Bir fonksiyonun belge dizisine 1 veya daha fazla örnek çağrı ekleme kuralı evrensel olarak benimsenmese de, birinin fonksiyonunuzu anlamasına yardımcı olmakta oldukça etkili olabilir. Gerçek dünyadan bir örnek için, numpy fonksiyonu np.eye için bu belge dizisine bakın.

İyi programcılar, kullandıktan kısa bir süre sonra kodu çöpe atmayı düşünmedikleri sürece (ki bu nadirdir) docstring'leri kullanırlar. Bu yüzden siz de docstring yazmaya başlamalısınız!

## **Functions that don't return**

Fonksiyonumuza return anahtar kelimesini dahil etmeseydik ne olurdu?

def least\_difference(a, b, c):

*"""Return the smallest difference between any two numbers*

*among a, b and c.*

*"""*

diff1 = abs(a - b)

diff2 = abs(b - c)

diff3 = abs(a - c)

min(diff1, diff2, diff3)

print(

least\_difference(1, 10, 100),

least\_difference(1, 10, 10),

least\_difference(5, 6, 7),

)

None None None

Python, bu tür fonksiyonları tanımlamamıza olanak tanır. Bunları çağırmanın sonucu özel bir değer olan "None"dir. (Bu, diğer dillerdeki "null" kavramına benzer.)

Bir return ifadesi olmadan, en\_küçük\_fark tamamen anlamsızdır, ancak yan etkileri olan bir fonksiyon hiçbir şey döndürmeden de faydalı bir şey yapabilir. Bunun iki örneğini daha önce gördük: print() ve help() hiçbir şey döndürmez. Bunları yalnızca yan etkileri (ekrana metin yerleştirmek) için çağırırız. Faydalı yan etkilere örnek olarak bir dosyaya yazma veya bir girdiyi değiştirme verilebilir.

mystery = print()

print(mystery)

None

## **Default arguments**

help(print) işlevini çağırdığımızda, print işlevinin birkaç isteğe bağlı argümanı olduğunu gördük. Örneğin, yazdırılan argümanlarımızın arasına özel bir dize koymak için sep için bir değer belirtebiliriz:

print(1, 2, 3, sep=' < ')

1 < 2 < 3

Ancak bir değer belirtmezsek sep'in varsayılan değeri ' ' (tek bir boşluk) olarak kabul edilir.

print(1, 2, 3)

1 2 3

Tanımladığımız fonksiyonlara varsayılan değerlere sahip isteğe bağlı argümanlar eklemek oldukça kolaydır:

def greet(who="Colin"):

print("Hello,", who)

greet()

greet(who="Kaggle")

*# (In this case, we don't need to specify the name of the argument, because it's unambiguous.)*

greet("world")

Hello, Colin

Hello, Kaggle

Hello, world

## **Functions Applied to Functions**

İşte ilk başta çok soyut görünse de güçlü bir şey. Fonksiyonları diğer fonksiyonlara argüman olarak kullanabilirsiniz. Bazı örnekler bunu daha açık hale getirebilir:

def mult\_by\_five(x):

return 5 \* x

def call(fn, arg):

*"""Call fn on arg"""*

return fn(arg)

def squared\_call(fn, arg):

*"""Call fn on the result of calling fn on arg"""*

return fn(fn(arg))

print(

call(mult\_by\_five, 1),

squared\_call(mult\_by\_five, 1),

sep='**\n**', *# '\n' is the newline character - it starts a new line*

)

5

25

Diğer fonksiyonlar üzerinde işlem yapan fonksiyonlara "üst düzey fonksiyonlar" denir. Muhtemelen bir süre kendi fonksiyonunuzu yazmayacaksınız. Ancak Python'da çağırmayı faydalı bulabileceğiniz üst düzey fonksiyonlar mevcuttur.

İşte max fonksiyonunu kullanan ilginç bir örnek.

max, varsayılan olarak argümanlarının en büyüğünü döndürür. Ancak isteğe bağlı key argümanını kullanarak bir fonksiyona geçersek, key(x) değerini en üst düzeye çıkaran x argümanını (yani 'argmax') döndürür.

def mod\_5(x):

*"""Return the remainder of x after dividing by 5"""*

return x % 5

print(

'Which number is biggest?',

max(100, 51, 14),

'Which number is the biggest modulo 5?',

max(100, 51, 14, key=mod\_5),

sep='**\n**',

)

Which number is biggest?

100

Which number is the biggest modulo 5?

14