Functions

معرفی توابع:

این قسمت شامل توضیح کلی درمورد توابع در پایتون است و این که چطور یکی از آن‌ها را خودتان تعریف کنید. توابع یکی از بلوک‌های اساسی است که در کد نوشتن از آن‌ها استفاده خواهیم کرد، درست زمانی که نیاز داریم برای حل مسأله انبوهی کد بنویسیم توابع به کمکمان می‌آیند! **خب حالا یک تابع چی هست؟** تعریف رسمیش این هست که تابع یک ابزار سودمند است که یک مجموعه از عبارات و دستورها را به صورت یک گروه درمی‌آورد، و وقتی که می‌خواهیم همه آن‌ها را بیشتر از یک بار اجرا کنیم، توابع می‌توانند خط‌هایی که نیاز به نوشتن داریم را کاهش دهند. همچنین توابع به ما اجازه می‌دهند که پارامترهایی را به عنوان ورودی در نظر بگیریم. اگر یک خرده پایه‌ای‌تر به موضوع نگاه کنیم، توابع اجازه می‌دهند که مجبور نباشیم که آن مجموعه کد را مداوم در برنامه تکرار کنیم! اگر به خاطر بیاورید در درسی که مربوط به list و string بود، ما با تابع len() آشنا شدیم که طول یک دنباله را به ما می‌داد.

دستور def

خب بیایید ببینیم چطور در پایتون می‌توانیم که تابع را تعریف کنیم؟

In [1]:
```
def name_of_function(arg1,arg2):
    '''
    This is where the function's Document String (docstring) goes
    '''
    # Do stuff here
    a = 0
    # Return desired result
    return a
```

In [2]:
```
help(name_of_function)
```

```
Help on function name_of_function in module __main__:

name_of_function(arg1, arg2)
    This is where the function's Document String (docstring) goes
```

In [3]:
```
name_of_function
```

Out[3]:
```
<function __main__.name_of_function>
```

In [5]:
```
name_of_function(1,2)
```

Out[5]:
```
0
```

ما با یک **def** و سپس یک فاصله و اسم تابع شروع کردیم. هر چیزی که درباره اسم گفتیم هم اینجا مفید هست. سعی کنید اسمی که انتخاب می‌کنید، به کاری که تابع انجام می‌دهد مربوط باشد، مثلا همان تابع len که سرواژه length هست و طول یک دنباله را برمی‌گرداند. اما احتیاط کنید. شما ممکن هست اسمی را برای تابع خود انتخاب کنید که قبلا در پایتون استفاده شده باشد، مثلا همین تابع len یا print در این صورت تابع جدیدی که تعریف کردید جای قبلی را می‌گیرد.

در ادامه نام تابع یک جفت پرانتز است که در داخل آن ورودی‌ها با ویرگول جدا شده‌اند. شما از این متغیرها می‌توانید درون تابع استفاده کنید. در نهایت بلوک یک تابع با دو نقطه شروع می‌شود. حالا قدم مهمی در این جا لازم است، این است که باید کدهای درون تابع را با یک تو رفتگی شروع کنید. و این تورفتگی‌ها باید در همه خطوط یکسان باشند. بعد از آن یک راهنمای متنی نوشته شده است و در ادامه کاری که از تابع می‌خواهیم انجام بدهد را نوشتیم.

Example 1: A simple print 'hello' function

In [6]:
```
def say_hello():
    print('hello')
```

Call the function:

In [7]:
```
say_hello()
```

```
hello
```

Example 2: A simple greeting function

Let's write a function that greets people with their name.

In [8]:
```
def greeting(name):
    print('Hello %s' %(name))
```

In [10]:
```
greeting('john')
```

```
Hello john
```

Using return

Let's see some example that use a `return` statement. `return` allows a function to *return* a result that can then be stored as a variable, or used in whatever manner a user wants.

Example 3: Addition function

In [11]:
```
def add_num(num1,num2):
    return num1+num2
```

In [12]:
```
add_num(4,5)
```

Out[12]:
```
9
```

In [13]:
```
# Can also save as variable due to return
result = add_num(4,5)
```

In [14]:
```
print(result)
```

```
9
```

In [15]:
```
add_num('one','two')
```

Out[15]:
```
'onetwo'
```

In [16]:
```
add_num(1, 'two')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-91623f343319> in <module>()
----> 1 add_num(1, 'two')

<ipython-input-11-4e98b39fa5d2> in add_num(num1, num2)
      1 def add_num(num1,num2):
----> 2     return num1+num2

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [ ]:
```

```

What happens if we input two strings?

Note that because we don't declare variable types in Python, this function could be used to add numbers or sequences together! We'll later learn about adding in checks to make sure a user puts in the correct arguments into a function.

Let's also start using `break`, `continue`, and `pass` statements in our code. We introduced these during the `while` lecture.

Finally let's go over a full example of creating a function to check if a number is prime (a common interview exercise).

We know a number is prime if that number is only evenly divisible by 1 and itself. Let's write our first version of the function to check all the numbers from 1 to N and perform modulo checks.

In [24]:
```
def is_prime(num):
    '''
    Naive method of checking for primes.
    '''
    sqrtnum = int(num**0.5)+1
    for i in range(2,sqrtnum):
        if num % i == 0:
            print('is not prime!')
            break
    else:
        print('is prime!')
```

In [25]:
```
is_prime(16)
```

```
is not prime!
```

In [29]:
```
is_prime(3)
```

```
is prime!
```

Note how the `else` lines up under `for` and not `if`. This is because we want the `for` loop to exhaust all possibilities in the range before printing our number is prime.

Also note how we break the code after the first print statement. As soon as we determine that a number is not prime we break out of the `for` loop.

We can actually improve this function by only checking to the square root of the target number, and by disregarding all even numbers after checking for 2. We'll also switch to returning a boolean value to get an example of using return statements:

In [38]:
```
def is_prime2(num):
    '''
    Naive method of checking for primes.
    '''
    if num == 2:
        return True
    if  num % 2 == 0:
        return False
    sqrtnum = int(num**0.5) + 1
    for i in range(3, sqrtnum, 2):
        if num % i == 0:
            return False
    return True
```

In [39]:
```
is_prime2(2)
```

Out[39]:
```
True
```

In [40]:
```
def no_return(a):
    a += 2
```

In [41]:
```
d = no_return(3)
```

In [42]:
```
print(d)
```

```
None
```

Why don't we have any `break` statements? It should be noted that as soon as a function *returns* something, it shuts down. A function can deliver multiple print statements, but it will only obey one `return`.

Great! You should now have a basic understanding of creating your own functions to save yourself from repeatedly writing code!