

Python Chapter 6: Functions

This guide has been written to reinforce your learning with the actual chapter lessons and not to serve as alternative way of learning python. I have tried my best to explain and provide explanations; however, if you find anything that needs to be corrected, please inform me.

- Maruthi Basava

BEFORE YOU START WRITING CODE, PLEASE BE AWARE THAT WHITESPACES MATTER IN PYTHON. LEAVING STRAY SPACES IN VARIOUS AREAS COULD CRASH YOUR PROGRAM. PLEASE MAKE SURE THAT THE SYNTAX IS CORRECT.

In the last chapter, we learned about the while loop. In this chapter, we will learn about **functions**, but this time, more in depth.

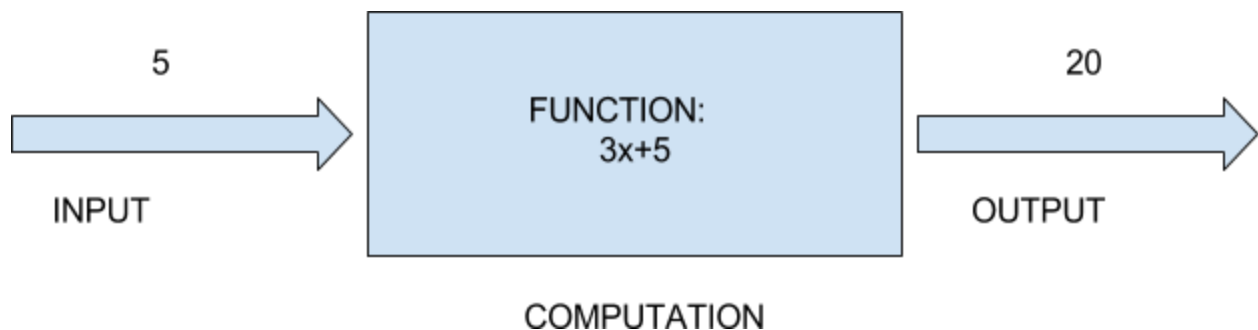
Returning values from a function.

Remember in your math class when you learned about functions?

Like how you have to plugin in an X value to get a Y value back?

It's kind of like that in Python.

Take a look at this diagram:



Hopefully you recognize and understand it.

In the function, $f(x) = 3x+5$, we are plugging in a 5 into the x and through the computation (by plugging in 5 in place of x, we multiply 3 to 5 then add 5) we get back a 20.

Functions in python work the same way!

Take a look at this code.

```
def calculate(x):  
    return 3*x + 5  
  
print calculate(10)
```

Terminal:

```
35
```

What is this function doing?

1. This function is taking in a variable x, then multiplying 3 to it, then adding a 5 to it.
2. Then we call the function with 10 as its argument then printed a 35 on the terminal.

Easy to understand right?

The **return** keyword allows the function give back a value once it's done computing.

Here is another example of a function returning a value once it's done computing.

```
def convert_raw_score(math, reading, writing):  
    return (math * 20) + (reading + writing) * 10  
  
print convert_raw_score(35,35,40)
```

Terminal:

```
1450
```

In this program, we are converting an SAT raw score into a composite score.

If you get a raw score of 35 in math, 35 in reading, and a 40 in writing, you would get a 1450 on your SAT.

Introduction to Imports:

In python, there are many functions that have been made for us to use. However, for organizational purposes, many of them have been stored into libraries.

To use these functions, you must import them using the **import** keyword.

In this example, we will **import** a library that enables us to generate a random number.

```
import random
```

Remember: All imports must be on the top of the file before any code is written.

Using the random library:

```
import random  
  
random_number = random.randint(1,100)  
print random_number
```

Notice how the import statement is always at the top of the file? This is because the libraries must be imported before you can use them in your code.

Now run this code in the terminal.

```
26
```

Run it again.

```
73
```

Run it again.

```
7
```

You get it now?

Everytime we run this code in the terminal, we get a new, random number.

Let's take a closer look at the code.

```
import random

random_number = random.randint(1,100)
print random_number
```

To get a random number, we use the random library stored into the variable **random**, DOT (dot means that you are able to use functions inside the library), randint(1,100). Those two numbers represent the minimum number and the maximum number respectively.

When we run this, we get a number between 1 and 100.

For extra practice, let's generate a number between -100 and 100

```
import random

random_number = random.randint(-100,100)
print random_number
```

Terminal:

```
-57
```

Other functions in the random library.

There are two other functions that you need to know for this chapter.

randrange function

The randrange function also generates a random number, but it has various forms.

The simple randrange function takes in one value and generates a number from 0 to that assigned value but not including it.

I think it would be easier if you see it in code.

```
import random  
  
random_number = random.randrange(100)  
print random_number
```

Terminal:

```
78
```

This function would generate a random number from 0 to 99 but not 100.

The second variation of this function takes in two numbers, the minimum value and the maximum value. The function generates a number from the minimum (which is included) to the maximum number (not included).

```
import random  
  
random_number = random.randrange(5, 100)  
print random_number
```

Terminal:

```
5
```

After a billion terminal runs, I finally got a 5 because it generates a number from 5 to 99.

The third variation of this function takes in three numbers! The minimum value, the maximum value, and the step-up value. The step-up value allows the function to generate numbers in increments of that value rather than 1.

```
import random

random_number = random.randrange(5, 100, 10)
print random_number
```

Terminal:

```
95
```

Run it again

```
65
```

As you can see, we always will have that 5 there because that is our minimum value, but every time we generate a new number, it will be through the increments of 10.

uniform function

The uniform function also generates a random number, but instead of integers (no decimals), it will generate floating point numbers (numbers with decimals).

The function takes in two values: minimum value and the maximum value.

```
import random

random_number = random.uniform(1.0,100.0)
print random_number
```

Terminal:

```
60.6016389905
```

Using the math library:

One of the most important libraries in any programming language is the math library.

I'm not going to elaborate as much on this library, but if you learned everything in this guide, you should be fine.

Just remember to import the math library just like so:

```
import math
```

And use this chart as a reference.

Table 6-2 Many of the functions in the math module

Function	Description
<code>math.acos(x)</code>	Returns the arc cosine of <i>x</i> , <i>in radians</i> .
<code>math.asin(x)</code>	Returns the arc sine of <i>x</i> , <i>in radians</i> .
<code>math.atan(x)</code>	Returns the arc tangent of <i>x</i> , <i>in radians</i> .
<code>math.ceil(x)</code>	Returns the smallest integer that is greater than or equal to <i>x</i> .
<code>math.cos(x)</code>	Returns the cosine of <i>x</i> in radians.
<code>math.degrees(x)</code>	Assuming <i>x</i> is an angle in radians, the function returns the angle converted to degrees.
<code>math.exp(x)</code>	Returns e^x
<code>math.floor(x)</code>	Returns the largest integer that is less than or equal to <i>x</i> .
<code>math.hypot(x, y)</code>	Returns the length of a hypotenuse that extends from (0, 0) to (<i>x</i> , <i>y</i>).
<code>math.log(x)</code>	Returns the natural logarithm of <i>x</i> .
<code>math.log10(x)</code>	Returns the base-10 logarithm of <i>x</i> .
<code>math.radians(x)</code>	Assuming <i>x</i> is an angle in degrees, the function returns the angle converted to radians.
<code>math.sin(x)</code>	Returns the sine of <i>x</i> in radians.
<code>math.sqrt(x)</code>	Returns the square root of <i>x</i> .
<code>math.tan(x)</code>	Returns the tangent of <i>x</i> in radians.</TB></TBL>

Formatting Numbers:

Let's say that you have a float point (a number with a decimal), and you want to round it to the nearest hundredth or whatever. By avoiding convoluted math to round it, we can easily use the **format** function.

By giving the function two inputs (first one is the decimal value that you want to round, the second is a string expression that tells the function what decimal place to round)

```
a = 9978.7899
b = format(a, '.2f')
print ''
print b
print ''
```

Terminal:

```
9978.79
```

The breakdown of **‘.2f’**

1. The **.2** means that we are rounding the number to two decimal places
2. The **f** means that we are formatting a float (a number with a decimal. YOU CAN ONLY DO THIS WITH A FLOAT.)

Formatting with commas:

Let's say that you have a number 1000000 and your boss tells you to make it look like 1,000,000, what are you going to do?

We can easily solve this problem by using the **format** function.

```
a = 9978.7899
b = format(a, ',.2f')
print ''
print b
print ''
```

Terminal:

```
9,978.79
```

But this time, you have to insert a comma in the string.

Getting a String's Length

Let's say that we want to find out how many characters a string has, we can do this using the **len** function.

```
name = 'Maruthi Basava'
print len(name)
```

Terminal:

```
14
```

By the way, even spaces are considered as a character.

Upper and Lower functions:

In python, there are two cool string functions that allow you to make all the characters either uppercase or lowercase.

To uppercase all characters, use the **upper** function:

```
a = "wombo"  
b = a.upper()  
  
print a  
print b
```

Terminal:

```
wombo  
WOMBO
```

To lowercase all the characters, use the **lower** function:

```
a = "HIS POWER IS OVER 9000!!!!"  
b = a.lower()  
  
print a  
print b
```

Terminal:

```
HIS POWER IS OVER 9000!!!!  
his power is over 9000!!!!
```

The **find** function:

In python the **find** function works exactly as a contains function in other languages. The main purpose of the **find** function is to find a substring inside of a string.

The find function takes in one parameter: the keyword substring.

In the code below, we are calling the **find** function on the string **lyrics** to find the keyword **“dreamer”**. If it exists, then it will **return the index** of the very first character of the keyword (the first character of “dreamer” is “d”, so the function will return the position of ‘d’). If for any reason the function DOES not find the keyword, it will return a -1. When it returns a -1, then you should know that there are no instances of that keyword in the string.

Remember how all strings are an array of characters?

(If you don’t know what an array is, don’t worry, I’ll go over it.)

```
lyrics = "You may say I\'m a dreamer\nBut I\'m not the only one\nI hope some day you\'ll join us\nAnd the world will be as one"
keyword = "dreamer"

print ''
print lyrics
print ''
print "The keyword is found: check index " + str(lyrics.find(keyword))
print ''
```

Terminal:

```
You may say I'm a dreamer
But I'm not the only one
I hope some day you'll join us
And the world will be as one

The keyword is found: check index 18
```

Here is a little diagram of what I mean.

HELLO

0 1 2 3 4

So in the string "HELLO"

The index of H is 0

The index of E is 1

The index of L is 2

The index of the other L is 3

The Index of the O is 4

String slicing:

If you want to use a small part of a bigger string, you can do just that by using this format.

```
string[start : end]
```

The example below shows you how to use it.

```
lyrics = "You may say I'm a dreamer\nBut I'm not the only one\nI hope some day you'll join us\nAnd the world will be as one\nYou are not alone\nYou I'm a dreamer"
keyword = "dreamer"

print ''
print lyrics
print ''
print lyrics[10:20]
print ''
```

```
You may say I'm a dreamer
But I'm not the only one
I hope some day you'll join us
And the world will be as one
You are not alone
You I'm a dreamer
```

So what's happening?

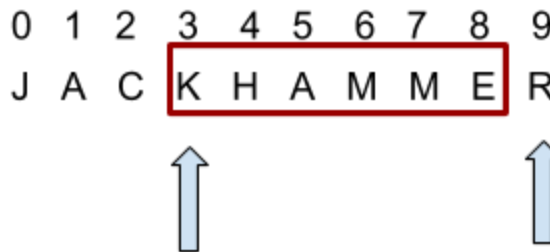
1. We made a variable lyrics that contains lyrics from a song
2. Then we print out the lyrics variable
3. Then we print out the sliced string that only contains the characters we asked for by their index numbers except for the last character.
4. Try counting them out for yourself, start with 'Y' in "You", which will be index of 0, then for every character after add one.

Look at this diagram, it might help you understand this concept a bit more.

```
Item = 'JACKHAMMER'
```

```
print item[3:9]
```

0	1	2	3	4	5	6	7	8	9
J	A	C	K	H	A	M	M	E	R



Everything in the red box is going to be printed.

START HERE
WITH THIS
CHARACTER
INCLUDED

END HERE
WITH THIS
CHARACTER

NOT

INCLUDED

The second statement assigns the string 'Lynn' to the `middle_name` variable. If you leave out the *start* index in a slicing expression, Python uses 0 as the starting index. Here is an example:

```
full_name = 'Patty Lynn Smith'
first_name = full_name[:5]
```

The second statement assigns the string 'Lynn' to `first_name`. If you leave out the *end* index in a slicing expression, Python uses the length of the string as the *end* index. Here is an example:

```
full_name = 'Patty Lynn Smith'
last_name = full_name[11:]
```

The second statement assigns the string 'Smith' to `first_name`. What do you think the following code will assign to the `my_string` variable?

```
full_name = 'Patty Lynn Smith'
my_string = full_name[:]
```

The second statement assigns the entire string 'Patty Lynn Smith' to `my_string`. The statement is equivalent to:

```
my_string = full_name[0 : len(full_name)]
```

The slicing examples we have seen so far get slices of consecutive characters from strings. Slicing expressions can also have step value, which can cause characters to be skipped in the string. Here is an example of code that uses a slicing expression with a step value:

```
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print(letters[0:26:2])
```


The third number inside the brackets is the step value. A step value of 2, as used in this example, causes the slice to contain every 2nd character from the specified range in the string. The code will print the following:

```
ACEGIKMOQSUY
```

You can also use negative numbers as indexes in slicing expressions, to reference positions relative to the end of the string. Here is an example:

```
full_name = 'Patty Lynn Smith'
last_name = full_name[-5:]
```

Recall that Python adds a negative index to the length of a string to get the position referenced by that index. The second statement in this code assigns the string 'Smith' to the `last_name` variable.

Note: Invalid indexes do not cause an error. For example:

- If the *end* index specifies a position beyond the end of the string, Python will use the length of the string instead.
- If the *start* index specifies a position before the beginning of the string, Python will use 0 instead.
- If the *start* index is greater than the *end* index, the slicing expression will return an empty string.