Python Chapter 3: Modularizing Programing with 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 explains, but 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.

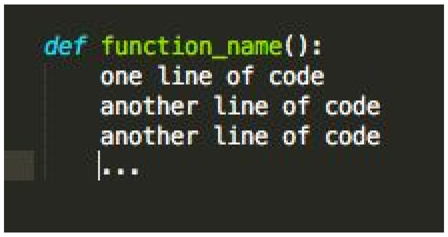
In the last chapter, we learned how to use functions and variables. In this chapter we will be learning how to make your own functions and how to use them.

Creating and using functions

Correlates to Assignments (Program 3-1)

Remember what functions are? Functions are just pre-written code or blocks of code that can be used multiple times without re-writing code a million times.

How to create a function:



```
def function_name():
    one line of code
    another line of code
    another line of code
    ...
```

In python, to create a function, you need to first start with def, then the function name, then a pair of parenthesis, and a colon. After the colon, you need to press [**ENTER**] or move on to the next line. It is very important that every line of code after that colon has a whitespace (space) in front of it to make sure it is underneath the function header.

Think of the time when you outlined your notes with bullet points,

Section Header

- Information
- Information
- Information
- ...

It is literally the same thing in python.

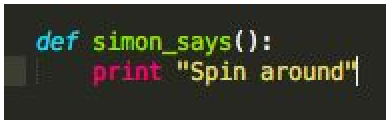
Your Section Header becomes def function_name():
And your bullet points become the lines of code.

So it would look like:

```
def function_name:
        Line of code
        Line of code
        Line of code
        …
```

Ok, I hope that made sense, Let’s actually make a function that does something.

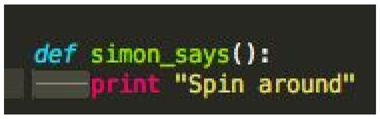
Here is a function that prints what simons says



```
def simon_says():
    print "Spin around"
```

.

Keep in mind of the white space in front of your line of code:

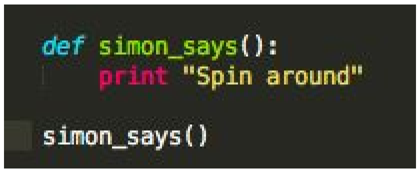


```
def simon_says():
    print "Spin around"
```

It is very very very very important! If you don’t have space in the front, your code will crash.

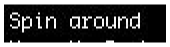
Ok, so we made a function, but how do we call it?
Remember all those times you used the str() convert an integer to a string?
It's the same thing. To use a function, you just need to have the function name then a pair of parenthesis.

For example:



```
def simon_says():
    print "Spin around"

simon_says()
```

In the terminal:

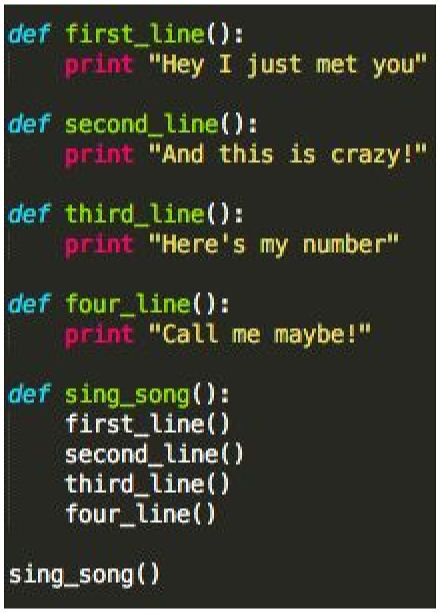


```
Spin around
```

Unlike the str() function, we didn't put anything in between the parenthesis, that's because we didn't tell the function to take something in. When functions don't need anything inside their parenthesis, you don't put anything in between.

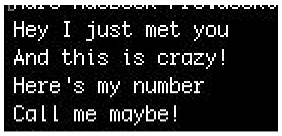
Take a look at this:

Correlates to Assignments (Program 3-2 to 3-3)



```
def first_line():
    print "Hey I just met you"

def second_line():
    print "And this is crazy!"

def third_line():
    print "Here's my number"

def four_line():
    print "Call me maybe!"

def sing_song():
    first_line()
    second_line()
    third_line()
    four_line()

sing_song()
```

Terminal:



```
Hey I just met you
And this is crazy!
Here's my number
Call me maybe!
```

Here we have five functions, but only sing_song() is actually called by me.

That’s because sing_song() function calls the other four functions.
You can do this by just calling the others functions like you’ve have before.

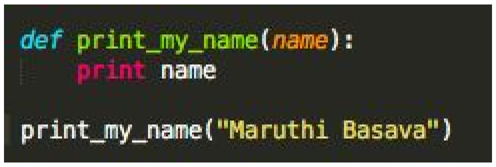
Just make sure that you maintain that whitespace in front of all the other lines of code underneath the function header.

Passing arguments to functions:
Correlates to Assignments (Program 3-4 to 3-5)

Passing arguments might sound but is nothing new to you. Remember the str() function, remember how you had to put in a number inside the parenthesis?

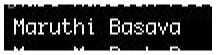
Now you can make your own functions that can exactly do that.

Take a look at this function here:



```
def print_my_name(name):
    print name

print_my_name("Maruthi Basava")
```

Terminal:



```
Maruthi Basava
```

Here, we have a parameter ***name*** in between the parenthesis. This means that this function now expects us to pass in something when it is called.

We can clearly see that the function print_my_name takes in a ***name*** parameter then prints it.

Remember: parameters are just variables waiting to be used inside of the function.

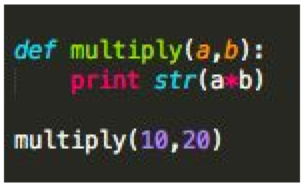
When we actually call the function print_my_name, we had to pass in a string in the between the parenthesis. This means that the string is then stored in ***name*** parameter of the function and is later used, in this case, we printed it out.

Passing multiple arguments:

Correlates to Assignments (Program 3-6)

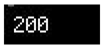
Instead of passing just one arguments, we can pass as many as we want.

Take a closer look at the code below:



```
def multiply(a,b):
    print str(a*b)

multiply(10,20)
```

Terminal:



```
200
```

In the function multiple, we passed in two arguments, a and b. When we called the function, we had to pass in two values to satisfy the function’s parameters.

Inside of the function, we have two variables: a and b.

We first multiply a and b by using the the multiplication operator (*).
Then we converted it into a string using the **str** function.
Then we print it.

Easy isn’t it?

We can ask any function that we make to take in as many parameters as we desire, as long as we pass them in when we call the function later.

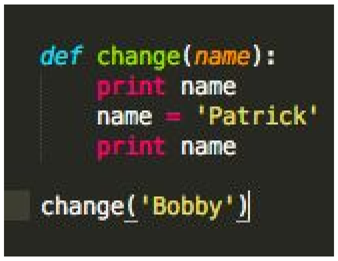
With this knowledge, we are now able to do many cool things!

Changing the value of parameters inside a function:

Correlates to Assignments (Program 3-7)

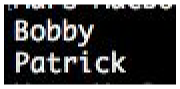
When functions have parameters, we have the power to change their values inside.

Take a look at the code below:



```
def change(name):
    print name
    name = 'Patrick'
    print name

change('Bobby')
```

Terminal:



```
Bobby
Patrick
```

When we first printed out the variable name, it printed ‘Bobby’ (which is what we passed into the function). But when we printed out the variable name again, it printed out ‘Patrick’. This is because we told the variable name to equal another value.

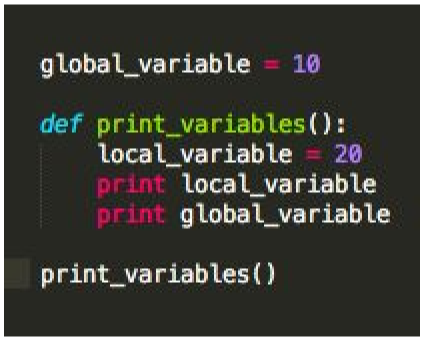
Local and Global Variables:

Correlates to Assignments (Program 3-8 to 3-9)

Variables created inside of a function are called local variables.

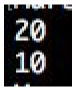
Variables created outside of a function are called global variables.

Take a look at the code here:



```
global_variable = 10

def print_variables():
    local_variable = 20
    print local_variable
    print global_variable

print_variables()
```

Terminal:

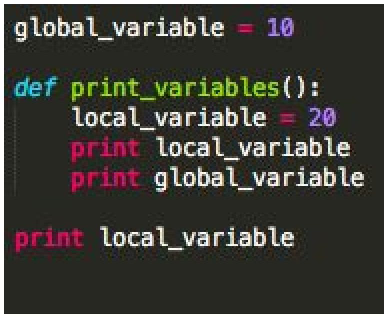


```
20
10
```

From looking at the code here, we are able to access both the global and local variables inside the function.

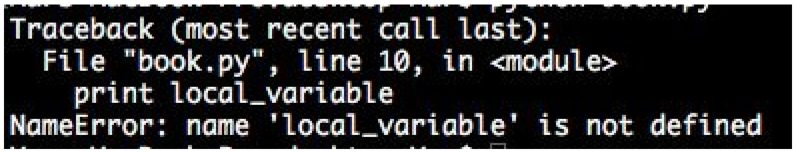
But can we access the local variable outside of the function?

Let's give it a try.



```
global_variable = 10

def print_variables():
    local_variable = 20
    print local_variable
    print global_variable

print local_variable
```

Terminal:



```
Traceback (most recent call last):
  File "book.py", line 10, in <module>
    print local_variable
NameError: name 'local_variable' is not defined
```

Nope! The program crashes.

That's because local variables cannot be accessible outside of the function, only on the inside. However, global variables can be accessed both inside and outside of a function.