Eduardo Ottoni

Roteiro de estudos

Date: 06/12/2021

Time: 15:19:12

# Topics:

* Return values
* Incremental development
* Composition
* Boolean functions
* Recursion

# Learning Objectives

* Develop example code to debug a function.
* Utilize a recursive function to compute a result.
* Develop a function using the concept of incremental development.

# Video:

The video talks about functions, return and calls.

The mains aspect of the video is a function that returns a value, and a call that store this value in a variable.

The aspects of the importance of return values are that it can be good for code reusing, increment software complexity, turn programming easier, and more.

Other main aspect of the video is the debug option, present in most IDE’s, that can pause the code running, in a chosen line, alter values and see the program running line by line.

# Defining the topics

* Return values

Return value is a property of the function, that allows it to share a result with a caller. The caller is the part of the software which calls the function and can receive or not a value in return. If you ask a function with no return, in Python, it will return None. In other languages it can create a syntax error.

When the function returns it stop the execution immediately. All code above will not run, it is called dead code.

A function can has more than one return, if this has some conditions, each condition can have his own return. Otherwise, is a good habit to have a default return, if for some reason the function must return some value for the program, but it ends with no return the code can crashes.

* Incremental development

Incremental development is a technique that uses scaffolding code, what means that you use some strategies to increment the code bit by bit, testing it.

For create a complex function that adds, subtract, and return logs, is a good way to use incremental development.

First you need to choose the variables, and just run, then add some algebra and run again, you can use print(variables) to verify if the values are as expected.

After some iteration, you will be with a great function, that may need less debugging, since you already solve a lot of the problems in the creation of the code.

Otherwise, if you write the entire function as you expect it should work you can be fooled by the computer syntax, and after all are written, will be more difficulty to figure out where exactly is the problem with the code.

Incremental development is more a software engineering technique than a hard code concept. When the function works properly, you can erase the scaffolding part of the code, because it is just for help the task, and is not part of the final product.

One example of scaffolding is to print intermediate variables, intermediate expressions, and run the function to realize if it works as expected.

* Composition

As we saw in the texts, a return or a function can have inside another function. For instance, print(abs(-3)) is an example of composition of functions. The most inside function will be run first, so, abs(-3) will turn itself in 3, since abs function in Python returns the absolute value of a number, and the return property of the functions turns abs(-3) in just 3. In the core of the computer, now we have print(3) that prints in the console log the value that it has inside, and then return void.

Composition is used to simplify the code, otherwise would be needed a lot of intermediate variables In this case, could be some like.

absolute\_value = abs(-3)

print(absolute)

Now we can realize it could be written as print(abs(-3)), because of composition property.

* Boolean functions

Boolean functions are important to respond if a comparison is true or not. There exist a lot of numerical examples of it. But in practice, it is useful. You may want to know if a user needs to pay taxes. So, you create a function mustPayTaxes(income).

mustPayTaxes(income):

if income > 40000:

return True

elif income <= 40000

return False

In this simple example, we can use mustPayTaxes in other parts of the code, using as a tool to conditional stataments. If musPayTaxes(user\_income) == True, return “The user needs to pay taxes”.

* Recursion

Recursion is at the same time a complex and an understandable behavior of the software.

If you want to know how much prime number there are between an integer and 2, you can use:

verifyIsPrime(number)

if number > 2:

if isPrime(number) == True:

print(number)

verifyIsPrime(number-1)

else:

print (2)

return

After all iterations, all prime number will be printed. We suppose in this case that there is a function that verify if the number is prime, the function isPrime(number).

So, the software creates a pile that adds a new item in each call of the function, and then resolve all from the below to the top. This can turn programming in amazing tasks of the most advanced levels.

Keywords: guardian, scaffolding, leap of faith, vorpal, dead code, incremental development