Return Values

Surya Duggirala

January 2017

1 Introduction

So one of the more important and cornerstone ideas in computer science is that of the return value. Once you understand what it is you'll never forget it. The thing is that it can be a little difficult to understand in the scope of all the other concepts and terminology being thrown around. So in shorthand, a return value is just the value that the function returns. It can be anything. Primitive types like strings, ints, bools and more complex classes like lists, sets, and dictionaries just to name a few. You can even return other functions. Stupidly simple right? That's the idea. Let's go further into some (very basic) examples to really get an idea of how to use them. You can manipulate what you learn here to return anything you want as long as its within the scope of the language you're working in (except exceptions which we'll get to later on in the course). If you really master the concept of the return value, then the what would python print questions will become near second nature (with some exceptions).

2 Examples and Intuition

```
>>> # So let's assign a value to a variable
>>> num = 5
>>> type(num)
<class 'int'>
>>> # type is a really useful function that tells
>>> # you the class to which the parameter belongs
>>> # It's helpful for debugging among other things.
>>> # For instance when you want to see why you're
>>> # getting a TypeError, type can give you a good
>>> # idea.
>>> string = "string"
>>> type(string)
<class 'str'>
>>> boolean = False
>>> type(boolean)
```

```
<class 'bool'>
>>> # Now I'll put all these things into one function
>>> def return_values(inp):
...     if inp == True:
...         return False
...     if inp == 4:
...         return 10
...     if inp == "Hello World!":
...         return "I love computer science!"
...
>>> return_value(True)
False
>>> type(return_value(True))
<class 'bool'>
>>> return_value(4)
10
>>> type(return_value(4))
<class 'int'>
>>> # See how this is starting to work?
>>> return_value("Hello World!")
'I love computer science!'
>>> # Important to note you're returning
>>> # a string and not printing it so the
>>> # single quotes stay.
>>> type(return_value("Hello World!"))
<class 'str'>
>>> # This is where it gets interesting
>>> return_value([1,2,3,4,5])
>>>
>>> # See how I passed in a list instead
>>> # of a string, boolean, or integer value?
>>> # There's no condition in the function that
>>> # allows for a list so we return...
>>> type(return_value([1,2,3,4,5])
<class 'NoneType'>
>>> # On a more cautionary note, it's very
>>> # important to remember that this only
>>> # works if you call the function.
>>> # otherwise you get something like this
>>> def f():
...     return 1
...
>>> f
<function f at 0x102929d08>
>>> type(f)
<class 'function'>
```

So based on this demo we're left with a few questions. First and foremost, what can we do with this discovery? Well, anything honestly. You can look at functions almost like placeholders. Obviously the function does something and it more often than not won't return some constant value, but we can look at the end result of a function as something we can work with concretely. If that doesn't make much sense to you check out the next demo.

```
>>> def placeholder(num):
...     if type(num) == int:
...         return 1
...     return "Hello "
...
>>> number = placeholder(4)
>>> string = placeholder(False)
>>> number
1
>>> string
'Hello '
>>> number + 5
6
>>> string + "World!"
'Hello World!'
>>> placeholder(4) + placeholder(True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

So at a very fundamental level you can look at return values as what you're actually working with and the function itself being a placeholder. I want to be perfectly clear though that I say placeholder in the loosest sense of the word. Functions can do a wide variety of things and the way multiple functions interact makes the word 'placeholder' seem like an injustice and misnomer, container may be a better word. But as far as return values go, think of functions as I just said. I highly suggest playing around with this concept to gain true mastery of it. As always, practice makes perfect. As a final piece of knowledge, it's important to realize that print statements return None. Do with that what you will.

```
>> print(10000)
10000
>>> type(print(10000))
10000
<class 'NoneType'>
>>> print(print(100))
100
None
```

These concepts are a common thread more or less through computer science. You'll probably see different variations of these as you grow as a programmer

and student.