Functions and Abstractions in Python

Guillaume Macneil

October 6, 2020

Abstract

This is the second lesson on the 'Introduction to Python' course. This course is loosely based around the 'MIT Introduction to Computer Science - Fall 2016' course. The course is more focused towards the Python programming side of things.

1 Abstraction and Decomposition

Abstraction is an important idea in the realm of programming. Abstraction (in programming) is the concept of using a block of code without necessarily knowing exactly how it works, only knowing how to use it.

Decomposition goes hand-in-hand with abstraction as it is the act of breaking up your code into reusable sections called *functions*, which can be used later on in your program.

Abstraction and decomposition can be used together in order to achieve self-contained, reusable and coherent code that hides the minutia and tedious detail in favour of higher-level *functions* and *classes* (we will discuss classes later).

```
def is_even(int_input):
    return int_input % 2 == 0

is_even(6)
# Returns 'True'
is_even(3)
# Returns 'False'
```

There are already multiple things of note in this example. Firstly, the *'def'* is the precursor to writing a function, it tells the interpreter that you are going to *define* a function. Secondly is the *'is_even'* part, this is the name of the function, this is what you will *call* when you want to use the function. Thirdly, the *'(int_input):'* section lists the arguments or parameters that will be used by the function, these are the function's inputs. In this example we have one argument, 'int_input', all arguments are placed inside the two parentheses and

the colon tells the interpreter that the next indented block of code is the content of the function. Finally, the *'return'* defines the output of the function, which in this case is the Boolean output of whether the input is divisible by 2 without any remainder.

Underneath the function, you can see that it was called twice with the values of 6 and 3. 6 and 3 were the *int_inputs* and the output was whether they were even or not.

```
def print_smiles(smile_num):
    for i in range(smile_num):
        print(":)")

print_smiles(14)
# Prints ":)" 14 times vertically
```

"Well hang on a second!" you might be thinking, "There's no *return* here, only a *print()*". Well... You're right, both *print()* and *return* can be used, but they do have slight differences.

Output type:	**return**	**print**
Usage:	inside functions	inside and outside functions
Frequency:	can only be used once	can be used multiple times
Effects:	code afterwards is not exectuted	code afterwards is executed
Returned to:	function caller	the console

```
def squared(a):
    return a**2

def is_even(int_input):
    return int_input % 2 == 0

# Provided that the output of one is valid as the input of another,
# functions can be chained:

number = 4
is_even(squared(number))
# Returns True

squared(is_even(4))
# Returns "1" (This is a strange consequence of True also being equal to 1)
```

As you can see from the above example, the output of certain functions can also be used as the input of another function (provided that you use *return*). An important thing to remember is that the output of one function **has to match the desired input type** of the function you want to chain it to.

```
a = 5
b = 3

def complex_operation():
    return ((a ** b) * (a - b))/a + b

complex_operation()
# Returns "53.0"
```

As you can see from the above example, functions can access external variables that are not given as arguments (a and b in this case) and can also not have any arguments. **It is important to note that external variables cannot be defined inside functions.** Global variables can be used, but I will not teach them to you and generally you should not use them.

```
def is_even(a):
    return a % 2 == 0

def multiple_of_5_and_even(a):
    if is_even(a) and a % 5 == 0:
        print(f"{a} is both even and a multiple of 5.")
    else:
        print(f"{a} is not even and a multiple of 5.")

multiple_of_5_and_even(10)
# Prints "10 is both even and a multiple of 5."
```

This example illustrates that functions can be called inside functions, with *is_even()* being called inside *multiple_of_5_and_even()*. Also, functions can even be defined inside other functions, but this is rarely very useful.