Working with Functions

NSC 1002 Workshop 6

18 Nov 2020

Outline

Python is a sequence of instruction

- The **point of execution** of a python program follows the line-by-line sequence of instructions and *always begins at left-most column of the python program;*
- There is a difference between the **point of definition** of a function and the **point of invocation**;
- How functions work within a python program;
- When a line is indented. it means it is part of a function or a loop or a conditional or similar. **It may or may not be executed, depending on the conditional.**

The Point of Execution

- Python code is a sequence of instructions that is processed one line at a time.
- The python language starts in the left-most column of the first line and processes the instruction from left to right.
- As python executes the instructions on each line, the **point of execution** moves downward, one line at a time.

```
""" This program adds two numbers together """
x = 2.
y = 2.
z = x + y
print('The value is {}'.format(z))
```

Understanding functions

- If Python encounters **key words**, such `def` for functions, it expects indentation to mark the block of code as belonging to the function.
- There is a **point of definition** where the function is defined.
- And then there is a **point of invocation** where the function is invoked (or 'called'). These two concepts are demonstrated on the next couple of pages.

Defining and Invoking Functions

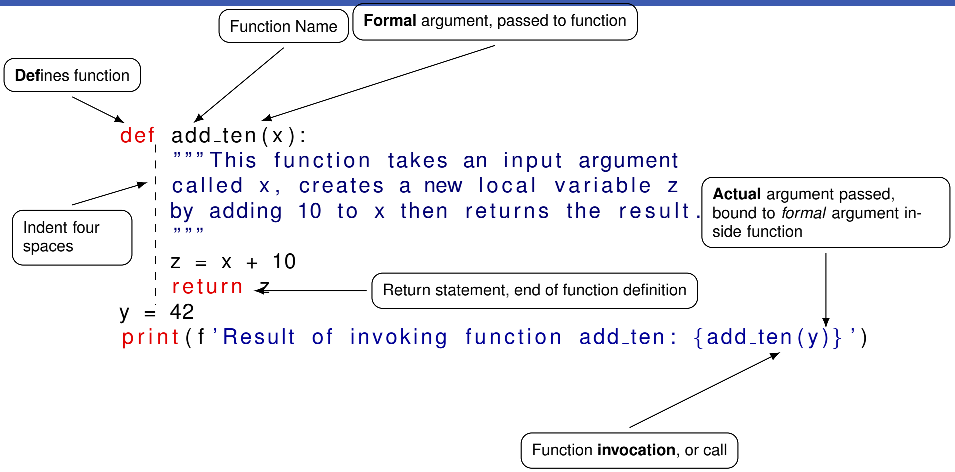
Using the Lambda function:

```
# First we define the function:
add_ten = lambda x: x + 10

# And now we invoke it:
y = 10
z = add_ten(y)

# The value of z should be 20, right?
print('z = ', z)
# or, printing another way:
print(f'z = {z}')
# or, calling the function from the print statent,
print(f'z = {add_ten(y)}')
# It works!
```

Function Definition versus Function Invocation



```python
def add_ten(x):
    """This function takes an input argument
    called x, creates a new local variable z
    by adding 10 to x then returns the result.
    """
    z = x + 10
    return z
y = 42
print(f'Result of invoking function add_ten: {add_ten(y)}')
```

Function Name
Formal argument, passed to function
Defines function
Indent four spaces
Actual argument passed, bound to *formal* argument inside function
Return statement, end of function definition
Function **invocation**, or call

Figure: The syntax for a function called add_ten()

Where does the function definition end?

The function ends when the indentation ends. You can use a *pass* command (which does nothing) to demarcate this if you like.

The execution of the function terminates when Python either encounters the key word RETURN or there are no more statements in the indented block of code to execute.

A RETURN statement can occur inside a function (for example, inside a conditional). But usually it at the end.

Look at an EXAMPLE CODE STRUCTURE 1

```
def my_max(x, y):
    """
    Finds the maximum of two variables
    """
    if x > y:
        max_value = x
    else:
        max_value = y
    return max_value
pass       # This line does nothing but the lack of indentations
           # demarcates where a function ends

v1 = 2.
v2 = 26.
max_of_v1_and_v2 = my_max(v1, v2)
print(max_of_v1_and_v2)          # 26. is printed
```

Look at an EXAMPLE CODE STRUCTURE 2

```
def my_max(x, y):
    """
    Finds the maximum of two variables
    """
    if x > y:
        max_value = x
    else:
        max_value = y
    return max_value

def test():
    v1 = 2.
    v2 = 26.
    max_of_v1_and_v2 = my_max(v1, v2)
    print(max_of_v1_and_v2)          # 26. is printed

test()
```

Questions?

Questions?

Now work through the examples!