Functions in Python

Leigh J. Little SUNY Brockport December 3, 2018

Contents

1 Introduction

This document will discuss the how to write your own functions in Python.

2 Tuples

Python has three native list-type structures. We previously discussed lists. The two other two list-type structures are *dictionaries* and *tuples*. Before we can discuss functions, we need to briefly touch on touples. A tuple is similar to a list, except the elements cannot be modified. A tuple is created using the () constructors:

```
>>> x = ('car',1,4.7)
>>> x
('car', 1, 4.7)
>>> x[0]
'car'
>>> x[1] = 17
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x[1] = 17
TypeError: 'tuple' object does not support item assignment
```

List a list, the elements of a tuple can be anything. The first element of the tuple has index number 0. Note that if you attempt to change `x[1]`, an error is generated.

While the elements of a tuple cannot be modified once they are assigned, it is possible to append elements to a tuple.

3 Functions

You can create your own functions in Python. This is best exhibited with an example.

```
def testfun(x):     # Function name is myfun and takes in 1
                      argument.  This argument can be anything
                      (a scalar, list, etc).  Note the : at the end.
    z = x**2 + 3*x  # Note the indenting
    return z        # The return statement goes back to the calling
                      routine
```

The above code is saved in a file called `myfun.py` Some notes about functions:

1) The name of the file containing the function can be anything.

2) You can have as many functions in the file as you like.

3) Note the keyword `def` and the (`:`) at the end of the `def` line.

4) The entire function body needs to indented (in addition to the usual indenting requirements).

5) The `return` keyword tells the function to return to the calling function.

6) A function can have numerous `return` statements.

7) A return statement can be empty (*i.e.*, you do not have to return a variable).

7) The variable `z` is the return variable. This is the output of the function.

8) There is no keyword that ends the function.

To use the function, create a driver script, then import the file/s containing the function into the workspace (all of the files should be in the same directory on your computer).

```
import myfun
x = 1.5
y = myfun.testfun(x)
print('x, y = ',x,y)
```

```
Output:
    x, y =  1.5 6.75
```

Note that the function we wrote (`testfun`) needs to be preceeded by the `myfun.` indicator.

There is yet another version of the `import` function that allows you to shorten the indicator name. For example, suppose someone named a module as `somemodulewithareallylongname`. You would not want to have to do

```
import somemodulewithareallylongname
y = somemodulewithareallylongname.other_function(x)
```

everytime you needed to call a function from the module. You can always override the indicator name with a modified version of the `import` command. Using the previous example:

```
import myfun as m      # Indicator name will now be m.
x = 1.5
y = m.testfun(x)
print('x, y = ',x,y)
```

```
Output:
    x, y =  1.5 6.75
```

In the event you need to pull several modules into the workspace, make sure they have unique indicator names. For example, the sequence of statements below will 'work' in the sense that it will not flag an error, but it is a really bad idea:

```
import myfun as m
import math as m
import os as m
```

4 Function with Multiple Inputs/outputs

Functions can have any number of inputs and/or outputs (including none). Multiple inputs are handled in the usual way. If you need multiple outputs, you add them to the return statement

```
def testfun2(x,y):
    a = x**2 + y**2 + x - y
    b = x**2 - y**2 - x - y
    return a,b
```

When you call the function, you can have the output returned in one of two ways.

```
import myfun as m
x = 1.5
y = -5.6
v1,v2 = m.testfun2(x,y)
print('v1, v2 = ',v1,v2)
```

```
Output:
    v1, v2 =  40.71 -22.009999999999998
```

In the above example, the two output values are returned in two separate variables (`v1` will contain the value of `a` and `v2` will contain the value of `b`).

An alternative to returning the variables individually is to return them in a tuple:

```
import myfun as m
x = 1.5
y = -5.6
t = m.testfun2(x,y)
print('t = ',t)
print('t[0] = ',t[0])
print('t[1] = ',t[1])
```

```
Output:
    t =  (40.71, -22.009999999999998)
    t[0] =  40.71
    t[1] =  -22.009999999999998
```

Which of these two methods you use depends on what you intend to do with the output values. Tuples are handy if you have numerous values that you would like to always be grouped together. For example, if the output were a set of polar coordinates, a tuple would make sense because you would want to keep them together.

Note that no matter which method you choose to return the outputs from a function, the function itself remains the same.