ECMM455 Python Worksheet 14: Functions

Prof Hywel Williams

September 12, 2019

1 Aims

- Practice writing and using functions

2 Further reading

Comprehensive information about functions can be found in Chapter 3 of *Think Python*.

3 Functions

A function is a reusable block of code for a particular purpose. Functions are called from other programs and return a set of data. So far you have used:

- Built-in functions, e.g. *print, max(), min(), len()*
- Method functions associated with strings, e.g. *count(), replace()*

In this worksheet you will learn how to create user-defined functions

3.1 Anatomy of a function

Any Python function has three parts:

- *Header*: The function header contains the *def* keyword, the name of the function, and some arguments (aka parameters) in parentheses.
- *Body*: The function body contains some indented programming instructions that may make use of the arguments. Variables defined here are local to the function.
- *Return:* The return statement specifies which value (if any) should be passed back to the main program when the function has finished running.

Look at the very simple function below.

```
def basicfunction():
  print("hello world")
```

Here there are only two lines but the function is still complete:

- Header: *def basicfunction():*

  - There are no arguments in this case.

- Body: *print("hello world")*
  - The only operation is a simple print statement.

- Return: *<blank>*
  - Here there is no return value specified, so Python will implicitly use the *None* value as a null return value.

Once defined, it can be called by name, e.g.

```
>>> basicfunction()
hello world
>>>
```

Now look at the slightly more complicated function below:

```
1 def anotherfunction(arg1,arg2):
2
3   print("The first argument is: " + str(arg1))
4   print("The second argument is: " + str(arg2))
5   sum_args = arg1+arg2
6
7   return sum_args
```

It has five lines (plus some whitespace for layout). Again the three components are present:

- Header: *def anotherfunction(arg1,arg2):*
  - Here there are two arguments *arg1* and *arg2.*

- Body: *<lines 3,4,5>*
  - Three lines specifying two print statements and an addition to create a new variable called *sum_args.*

- Return: *return sum_args*
  - Specifies that the return value will be the variable *sum_args*

Calling this function is also simple, as long as we give it the right number of arguments - look at the three example calls below:

```
>>> anotherfunction()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: anotherfunction() missing 2 required positional arguments: 'arg1' and 'arg2'
>>>
```

```
>>> anotherfunction(5)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: anotherfunction() missing 1 required positional argument: 'arg2'
>>>
```

```
>>> anotherfunction(5,7)
The first argument is: 5
The second argument is: 7
12
>>>
```

3.2 Arguments

Arguments are the values passed to a function for it to work with. The number of arguments is specified in function header. The function header also specifies the "local names" of the argument variables within the function. When arguments are passed to a function, Python uses several rules to interpret them:

- If un-named values are given, order is used to identify them
- If named values are given, order is ignored
- If default values are given in definition, then the function call does not need to supply a value, but any supplied value will overwrite the default

3.2.1 Exercises

1. Copy the code below into a new program and run it. Carefully look at the output from each example function call and make sure you understand the differences in how the arguments were interpreted.

```python
#function definition with three arguments
def arg_demo1(arg1,arg2,arg3):
    print arg1
    print arg2
    print arg3

#EXAMPLE FUNCTION CALLS

#call with un-named arguments
print "First call:"
arg_demo1('a','b','c')

#call with un-named arguments (same values, different order)
print "Second call:"
arg_demo1('c','a','b')

#call with named arguments
print "Third call:"
arg_demo1(arg1='x',arg2='y',arg3='z')

```

```
#call with named arguments (same values, different order)
print "Fourth call:"
arg_demo1(arg2='y',arg3='z',arg1='x')
```

2. Now copy the code below and run the program. Again make sure you understand the outputs.

```
#function definition with default values for arguments
def arg_demo2(arg1='a',arg2='b'):
    print arg1
    print arg2

#EXAMPLE FUNCTION CALLS

#call with no arguments (uses defaults)
print "First call:"
arg_demo2()

#call with supplied arguments (uses supplied values)
print "Second call:"
arg_demo2('x','y')
```

3.3 Return values

Return values are returned by a function after it has finished (i.e. the "answer" calculated by the function). The number of return values is specified in the return statement. Return values can vary:

1. Fixed value (not often useful)
2. Value of a variable that exists inside the function
3. An expression (will be evaluated and result returned)
4. A list/dictionary/tuple
5. Multiple values (separated by commas)

Every function has a return value – if it is not specified, a default return value of *None* is used (specifies "null result" - this is not necessarily an error, many functions are not intended to return a value).

The item that is returned is a value (not a variable) but can be assigned to a variable. Return values can be used in different ways:

- Ignored
- Used directly
- Assigned to a variable

3.3.1 Exercises

1. Copy the code below into a new program and run it. Carefully examine the outputs and make sure you understand what is happening.

```
#SOME FUNCTION DEFINITIONS
#function with no return statement (no return value specified)
def ret_demo1(name):
    print "hello " + name

#function that returns a value
def ret_demo2(name):
    print "hello " + name
    n = len(name)
    return n

#function that returns two values
def ret_demo3(name):
    print "hello " + name
    n = len(name)
    return n, n*2

#SOME EXAMPLE FUNCTION CALLS

#call 1 - ignores return value
print "\nFirst call:"
ret_demo1("barry")

#call 2 - uses return value directly
print "\nSecond call:"
print ret_demo1("billy")

#call 3 - uses return value directly
print "\nThird call:"
print ret_demo2("benny")

#call 4 - assigns return value to a variable for further use
print "\nFourth call:"
x = ret_demo2("bunty")
print x, x, x

#call 5 - uses multiple return values directly
print "\nFifth call:"
print ret_demo3("barny")

#call 6 - assigns multiple return values to variables
print "\nSixth call:"
v1,v2 = ret_demo3("barny")
print v1
print v2
```

3.4 Tuples

Functions with multiple return values send them back as a *tuple* unless otherwise specified. A tuple is a Python data structure. It is an ordered set of data in round brackets e.g. *(1,2,3)* is a tuple containing three numerical values. Sometimes we use the terminology N-tuple to refer to a tuple with N elements e.g a 2-tuple has two elements, a 3-tuple has three elements, etc.

Tuples behave similarly to lists in some ways:

- You can assign a tuple to a variable: *x = (1,2,3,4)*

- You can read an element using an index, e.g. here *x[2]* is *3*.

However, tuples are different to lists in other ways - primarily that they cannot be changed once created:

- You cannot change an element by assignment:
  - Creating list *X = [1,2,3]* followed by *X[1]=5* changes *X* to hold *[1,5,3]*
  - Creating tuple *Y = (1,2,3)* followed by *Y[1]=5* results in an error
- Tuples have no append method
- Tuples cannot be extended
- You cannot remove elements from a tuple.

Tuples are useful – you have already seen them used with strings e.g.

my_string = "dogs have %s legs and %s ears" % (4,2)

More information on tuples and how they differ from lists can be found here:
http://stackoverflow.com/questions/1708510/python-list-vs-tuple-when-to-use-each

4 Exercises on writing and using functions

For all the exercises below, you need to define a function and also write some suitable code that calls the function to test if it is working correctly. That is, write a program that contains the function definition and some simple code that uses the function. They are taken from the excellent set of exercises compiled by Torbjorn Lager (find it here: http://www.ling.gu.se/~lager/python_exercises.html).

1. Define a function *max_of_three()* that takes three numbers as arguments and returns the largest of them.
2. Define a function *my_sum()* and a function *my_multiply()* that sums and multiplies (respectively) all the numbers in a list of numbers. For example, *my_sum([1, 2, 3, 4])* should return 10, and *my_multiply([1, 2, 3, 4])* should return 24.
3. Define a function *is_palindrome()* that recognizes palindromes (i.e. words that look the same written backwards). For example, *is_palindrome("radar")* should return *True*.
4. Define a function *histogram()* that takes a list of integers and prints a histogram to the screen. For example, *histogram([4, 9, 7])* should print the following:

```
****
*********
*******
```

5. Write a function that given the name of a text file as an argument will create a new text file in which all the lines from the original file are numbered from *1* to *n* (where *n* is the number of lines in the file). (You can use *happy.txt* from ELE as a sample text file if you like.)
6. Write a function that given the name of a text file will calculate the average word length of the text stored within (i.e the sum of all the lengths of the word tokens in the text, divided by the number of word tokens). (*Hint:* look up the Python string method called *split()*.)

7. Write a function that recognises phrase palindromes such as "Go hang a salami I'm a lasagna hog.", "Was it a rat I saw?", "Step on no pets", "Sit on a potato pan, Otis", "Lisa Bonet ate no basil", "Satan, oscillate my metallic sonatas", "I roamed under it as a tired nude Maori", "Rise to vote sir", or the exclamation "Dammit, I'm mad!". Note that punctuation, capitalization, and spacing are usually ignored when identifying phrase palindromes. (*Hint:* your phrase palindrome recogniser can call on your word palindrome recogniser function from question 3.)