Workshop 5 Functions, Modules and Inputs

17 November 2020

Goals:

- To write Python programs that accept simple inputs from the User;
- To know what SciPy and NumPy are.
- To understand what a function is;
- To be able to write and use your own Python functions;

You should ensure that you have completed all exercises before the next workshop.

Accepting user input from the terminal

To accept user input from the terminal you need use the function `input`. To demonstrate this, type the following into your Spyder editor:

```
d = input("Please enter a number between 1 and 10: ")
print(f"You typed in the number {d}")
print(type(d))
```

Introduction to NumPy and SciPy

NumPy and SciPy are *modules* written specifically for performing mathematical and numerical operations. NumPy (Numeric Python) provides powerful, fast ways of handling large multidimensional arrays and matrices of numeric data. SciPy (Scientific Python) extends NumPy with additional functionality, including some standard algorithms. In this workshop we'll be working with NumPy.

To include NumPy and SciPy routines in your programs you use the key word **import**. There are many ways to use the import key word but the two main ways we will do so is:

```
import numpy
 or
import numpy as np
```

When you access NumPy functions, you will be required to type: "`numpy.<functionname>`" or "`np.<functionname>`" For example, if you wanted to use NumPy's function for the mean of a sequence, you would use the syntax "numpy.mean()". If you import numpy the second way, you would refer to it as "np.mean()".

One of the important aspects of NumPy is its ability to create and manipulate *arrays*, including multidimensional arrays, which we shall study later this term. A NumPy array can be created from a list shown at right

```
import numpy as np
a  = np.array([1, 2, 4,
    7])
b = np.array([1, 2, 4,
    7], float)
print(b)
print(type(b), type(b[2])
    )
```

This will produce the following output:

```
[1. 2. 4. 7.]
<class 'numpy.ndarray'> <class 'numpy.float64'>
```

What will it produce for ?

```
print(a)
print(type(a), type(b[2])
```

There are numerous ways to create numpy arrays: look up the following numpy functions using a search engine:

1. `a = np.arange([start,] stop, [step,] dtype=None)`: This function returns an array, `a`, of evenly spaced values in an interval defined by `start` and `stop`. <start> and <step> are optional (default are 0 and 1 respectively)
2. `a = np.ones(N)`: This function will return an array of size N filled with ones. This is useful for initialisation!
3. `a = np.random.rand(N)`: This function will return an array of size N filled with random numbers over $[0, 1)$. This is useful for initialisation!
4. `a = np.zeros(N)` This function returns an array of size N filled with zeros. Also useful for initialisation.

NumPy array elements can be accessed just like Python lists: indexed, sliced, and manipulated.

Creating a function

In programming, we often need to repeat sections of code many times. Python, and other programming languages, have the capacity to treat such sections of code in a special way so they can be easily repeated. In some programming languages these are called `subroutines`. In Python they are called `functions`.

- A function is a collection of statements that performs a task.
- Allows us to combine multiple statements into one instruction.
- Can allow us to repeat and modularize sections of code.
- Python has a number of inbuilt functions
  - Examples include: `abs()`, `print()`, `str()`, `int()`, `range()`, ...
- NumPy has a collection of functions we can use to manipulate arrays.
  - Examples include: `np.arange()`, `np.mean()`, `np.zeros()`, ...

Lambda functions

Functions can be quite complicated, but we will start with a simple special kind of function, called 'lambda functions'. These are one-line functions that have the form:

```
function_name = lambda arguments: function
```

Where `lambda` is a special name in Python indicating it is a function. Here are some examples.

```
# A function that evaluates the square of a number
g = lambda x: x**2
print(g(3))
print(g(-7))
```

and

```
# A function that evaluates the square of a number multiplied by a another number:
myname = lambda a, x: a*x**2
print(myname(2,3))
print(myname(-1,5))
```

A lambda function can have many arguments, but they should all fit on one line. The lambda function is very useful for simple functions, but to make more general ones we have to use a more complex form, as we describe below.

General functions

The following figure illustrates the **syntax** for creating a function more generally — see figure 1 for a diagram of a particular function example.

```
def function_name(parameters):
    """
    Documentation string (called a docstring)
    """
    block of code
    return result
```

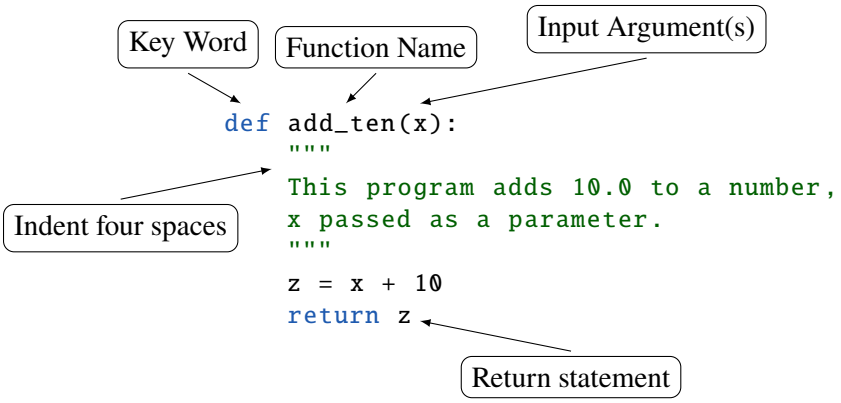

Figure 1: Diagram of the syntax for a specific function called add_ten()

This function does the same as the lambda function:

```
add_ten = lambda x: x + 10
```

The lambda function is easier in this simple case, but is not as flexible in general.

Creating and using functions: the structure of a Python code that uses functions

Functions must be defined *before* they are used. Good style when writing a program is to put all your initialisations at the top, the definitions right after so that they are easy to find. See the example, below.

```
import ...

# Functions
def myfun_1(x1):
"""
Describe what the first function does.
"""
    statements
    return
pass

def myfun_2(x2):
```

```
"""
Describe what the second function does.
"""
    statements
    return
pass

# Body of the code (uses initialisations and functions)
# Initialisations
x1 = 1.
x2 = 32.4
a = np.arange(10.)

# Loop calling function
for i in a:
    print(myfun_1(i), myfun_2(a[2]))
```

Appendix: What is a namespace and why is it important when using functions?

The concept of a `namespace` is used widely in computer programming because it assigns collections of names (variables, function names, etc) to a specific parts of a program. Some examples:

- You can use the function `print()` in any part of your program because that function belongs to any program written in the Python computing language.
- The function 'array' is part of numpy and is only available though `numpy.array` or `np.array` (depending on how you first initialized numpy).
- A *local namespace* is created when a function is called. This local namespace allows you to define variables in the function that are not seen in other parts of the program. For example, if we assign the variable `b = 2` in a new function, our main() routine would not recognise that variable. We shall look at examples in the workshop exercises.

This concept is important when programming because variables with the same name can be defined throughout a program, but knowing that functions have their own local namespace makes it clear that variables defined within a function live only within that function. This helps to write clear, reproducible code, and also helps with debugging.

Exercises

These exercises should be completed before the next workshop. For each program that you write you should make sure you test each aspect e.g., each branch when using conditional statements (*conditional statements* are also called *if statements*) and ensure that you have used **suitable comments and Docstrings**.

1. Create a new Python file called numpy_arithmetic.py. This exercise works with arrays. Open a new window in the Spyder Editor and create two new arrays:

```
x1 = np.arange(0., 2. * np.pi, (2. * np.pi) / 20.)
x2 = np.arange(0., 4. * np.pi, (4. * np.pi) / 10.)
print(len(x1),len(x2))
```

1. Is the size of the x1 array equal to the size of the x2 array? [1]
2. Which one is longer and why? [2]
3. What would you have to change so that they are equal?
4. (a) Write a lambda function that computes the value of $ax + bx^2$ for arbitrary values of a, b and x., and call it 'quadfunc' or similar. Make sure it works by called quadfunc(1,2,3) – your answer should be 21. (Why?)

   (b) Using matplotlib, make a plot of quadfunc(a,b,x) for $a = 1, b = 2$ and x going from 0 to 10.
5. Below is a section of code for a function that reads in user input and computes the circumference of a circle. Copy this into your Spyder editor, in the format outlined above (see the section on *Creating and using functions*), and run it. Compute the circumference of a circle for radii of 2, 5, 12, and 42.

```
import numpy as np
def circumference():
    """
    A function that calculates and displays the circumference of a
    circle with a diameter d where d is float given by the user.
    """
    d = np.float(input('Please enter the diameter of your circle'))
    total = np.pi * d
    print('The circumference of your circle is {}'.format(total))
pass

circumference()
```

[1] **Hint:** Use the `shape` attribute of x1 and x2. (e.g. `print(x1.shape)`) or `np.shape(x1)`

[2] **Hint:** Look up the syntax for `np.arange` online and examine each of the attributes carefully.

(a) Use Spyder to execute the program.

(b) Update the code so that the function returns the circumference. [3]

(c) Update the code so that the function has an input parameter, instead of asking for a number when you are running the function.

(d) Introduce a syntax error by removing one of the """ (i.e., the Docstring triple quotation marks).

- Do you understand the error message? The reason we introduce errors ourselves is so that we learn the meaning of the different types of error messages.

6. Investigate `np.linspace` and `np.arange` using a search engine. Write a program that uses both of these functions to produce two arrays of 10 values each. The values should be between -1 and 1. What do you need to do so that both functions give the same answer?

7. Write a function that prints out 'Hello' x number of times where x is given by the user. [4]

8. Write a function that takes in two integers from the user, x and y, and calculates whether x is divisible by y (i.e., no remainder is left). A suitable message should be given to the user.

9. Write a function that prints out the first x numbers of the Fibonacci sequence where x is given by the user.

- You created a program that computed the Fibonacci numbers. Use this program as a basis for your new function.

10. Edit *moon.py* to produce a function that performs the same task. [5]

- Edit the function so that the user is prompted to enter how much weight they gain each year.
- Edit the function so that the user is also prompted for how many years they are planning on visiting the moon.

[3] **Hint:** in the function, add a line at the end that says `return` and the variable which holds the circumference.

[4] **Hint**: consider using a similar structure to that of your circumference program.

[5] **Hint**: create a function that does the same thing as your moon.py program does, then call the function. This is a structure just like the circumference program.