1 Custom Function and Modules

Python supports a wide range of programming styles including procedural (imperative), object oriented and functional. While object oriented programming and functional programming are powerful programming paradigms, especially in large, complex software, procedural is often both easier to understand and a direct representation of a mathematical formula. The basic idea of procedural programming is to produce a function or set of function (generically) of the form:

$$y = f(x).$$

That is, the functions take one or more inputs and produce one or more outputs.

1.1 18.1 Functions

Python functions are very simple to declare and can occur in the same file as the main program or a standalone file. Functions are declared using the def keyword, and the value produced is returned using the return keyword. Consider a simple function which returns the square of the input, $y = x^2$.

```
from __future__ import print_function, division

def square(x):
return x**2
# Call the function
x = 2
y = square(x)
print(x,y)
```

In this example, the same Python file contains the main program– the final 3 lines – aswell as the function. More complex function can be crafted with multiple inputs.

```
from __future__ import print_function, division
def l2distance(x,y):
return (xy)**
2
# Call the function
x = 3
y = 10
z = l2distance(x,y)
print(x,y,z)
```

Function can also be defined using NumPy arrays and matrices.

```
from __future__ import print_function, division
import numpy as np
```

```
def l2_norm(x,y):
d = x y
return np.sqrt(np.dot(d,d))
# Call the function
x = np.random.randn(10)
y = np.random.randn(10)
z = l2_norm(x,y)
print(xy)
print("The L2 distance is ",z)
```

When multiple outputs are returned but only a single variable is available for assignment, all outputs are returned in a tuple. Alternatively, the outputs can be directly assigned when the function is called with the same number of variables as outputs.

```
from __future__ import print_function, division
import numpy as np
def l1_l2_norm(x,y):
d = x y
return sum(np.abs(d)),np.sqrt(np.dot(d,d))
# Call the function
x = np.random.randn(10)
y = np.random.randn(10)
# Using 1 output returns a tuple
z = l1_l2_norm(x,y)
print(xy)
print("The L1 distance is ",z[0])
print("The L2 distance is ",z[1])
# Using 2 output returns the values
l1,l2 = l1_l2_norm(x,y)
print("The L1 distance is ",l1)
print("The L2 distance is ",l2)
```

All of these functions have been placed in the same file as the main program. Placing functions in modules allows for reuse in multiple programs, and will be discussed later in this chapter.

MORE