Introduction to Functions

Introduction to Using Functions

In this section we shall discuss some of the details on how to use functions from the Standard Library. We have already come across a few functions, namely `print()` and `type()`. We shall cover how to define your own functions in a later section.

Python functions essentially take variables or values/objects as arguments, perform a task and them return values/objects. As we have seen before the syntax of a function call is:

`function_name(argument, argument, ...)`

Note that some functions return a `None` type when a return value isn't necessary. For example, the `print()` function:

[4]:
```
print_return = print('print out')

print('print function return:', print_return)
```

```
print out
print function return: None
```

If you want to know what a particular function does or how to use it, a quick way to find out is to pull up the docstring. In an IPython environment (such as a Jupyter notebook) this can be done by typing a `?` symbol after the function name and pressing enter. For example:

[5]:
```
print?
```

```
[0;31mDocstring:[0m
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
[0;31mType:[0m      builtin_function_or_method
```

In a default Python shell or script you can print out the docstring using the `help()` function. For example:

```
[8]: help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Alternatively you can refer to the Python documentation.

Now, let's take a look at the `print()` docstring itself. The first line of the docstring shows us the function name; and the name of the function arguments and their default values (if they have). Following this is a description of what the function does. Following this is a list of optional keyword arguments.

Sometimes a docstring will also contain examples on how to use the function, though none are present in this one.

We will see more examples of how optional keyword arguments work (in particular in the Chapter discussing Matplotlib…), for now let's use one of them. Let's change the argument `sep`, which is the string inserted between the values you put into the `print()` function. As we can see, it's default value is a space `' '`. As a first example, let's change the separation to an empty string (no space):

```
[1]: print('There', 'are', 'no', 'spaces', sep = '')
```

```
Therearenospaces
```

As another example, let's put commas (`','`) in between the values:

```
[2]: print('There', 'are', 'commas', 'between', 'values', sep = ',')
```

```
There,are,commas,between,values
```

The `input()` Function

Another important function in the Python Standard Library is the `input()` function. This function allows us to collect user inputs from the terminal.

`input()` takes a string as an argument, this gets printed to the terminal and the script is halted until the user has entered a string and pressed enter. This string that the user has entered is returned by the `input()` function; and can, therefore, be used in the rest of the script.

As a first example, consider the following code that asks the user for their name:

[5]:
```python
user_name = input('What is your name?')

print('Hello', user_name, ', nice to meet you!')
```

```
What is your name? Mayhew

Hello Mayhew , nice to meet you!
```

Remember that the program will wait for your input before it continues. If you are using a Jupyter Notebook, this means that other cells from the notebook will not run until the code has been fully executed or terminated.

Now, something that is important to note is that the return value from input is a string, even when a number is typed into the terminal:

[7]:
```python
user_number = input('Enter a number: ')

print(user_number, type(user_number))
```

```
Enter a number:  12

12 <class 'str'>
```

If you intend for the input to be used as a number, you must remember to convert it to one (an `int` or `float`):

[9]:
```python
user_int = int(input('Enter an integer: '))

print('Entered:', user_int, type(user_int))

user_float = float(input('Enter a number: '))

print('Entered:', user_float, type(user_float))
```

```
Enter an integer:  6

Entered: 6 <class 'int'>

Enter a number:  57.5

Entered: 57.5 <class 'float'>
```