Functions

Functions

What is a function?

Sometimes, similar tasks keep coming up in your programming work.
In such a case, defining your own **function** for such a task can save a lot of copy-pasting.

- Python has many **built-in functions**, some of which we have already seen: range and len, for example.
- In addition, you can define your own functions.
- As in math, a function can take variables as input and can return variables as output (though it does not have to).

For example, let's define a function square which gives back the square of any number you put in...

Functions

What is a function?

Let's define a function square which gives back the square of any number you put in:

```
def square(x):
        y = x*x
        return y
```

Enter this in your interpreter (with the proper indentation!).
What happened?
...Nothing?

- Python is now remembering the function you just 'taught' it. Try it out by entering: square(3), square(8), etc.

Functions

What is a function?

NB: As usual, we could have called our function (almost) anything we want. Try it:

```
def Robert(x):
    y = x*x
    return y
```

- Now you can enter Robert(3), Robert(8), etc. and get back squared numbers.

- However, after exiting Python and restarting it, it has forgotten all about Robert and square. Try.

Functions

Importing from script files

You can store functions in a script file and load them from the interpreter.

- Create a plain txt file called myfirstfunctions.py.
  In it put our function:

```
def Robert(x):
        y = x*x
        return y
```

- When python is running in the terminal (or equivalent), enter:
  import myfirstfunctions (NB: without the .py extension!)

- Now you can use your function, like this: myfirstfunctions.Robert(7)

After importing the function, Python has loaded the function definition in its memory, and you can use it as long as Python is running.

Functions

Importing from script files

You can put several functions into one script file and load them all at once.

- Add another function to your file myfirstfunctions.py:

```
def Susan(z):
    a = z+4
    return a
```

- Save your file and import it again: import myfirstfunctions
- Now you can use the function Susan as well as Robert.

A script containing multiple functions is called a module.

Functions

Importing from script files

- If you want to import only one function from a module containing many functions, you can do it like this:

  ```
  from myfirstfunctions import Susan
  ```

- After this, you can call the function directly from the interpreter: `Susan(13)`, for example.

- To import all of the functions in a module at once:
  ```
  from myfirstfunctions import *
  ```

  (However, be careful with this; it can make your code hard to check.)

Functions

Built-in modules

Python has many built-in modules as well. For example, the module math. This module contains functions such as sine, cosine, exponent, logarithm, etc. If you want to know which functions are available, try help(math).

Try using the math module:

- import math
  math.sin(0.5)

- from math import sin
  sin(0.5)

- from math import *
  sin(0.5)

For a list of available modules, see http://docs.python.org/2.7/py-modindex.html.

Functions

Getting Around the Interpreter

- Exit python from the terminal.
- Create another script file called testfunctionimport.py.
  In it, put the code:

```
from myfirstfunctions import Robert
print "The square of 11 is ", Robert(11)
```

- Run the file (python testfunctionimport.py).

Functions

Input and output

Our first function, Robert, has one input argument.

```
def Robert(x):
        y = x*x
        return y
```

When Python reads a = Robert(8),

- it evaluates the function Robert for input 8 and
- it assigns the returned value to variable a.

Try:

```
a = Robert(8)
print a
a - 5
```

Functions

Input and output

Functions can have more than one input argument. For example:

```
def Bob(x,y):
    z = x*x + y
    return z
```

- How does Python know which of the inputs is x, and which is y?
- When you call the function (e.g. Bob(7,8), the first item in the brackets will be assigned to variable x, the second to y.

Functions

Input and output

```
def Bob(x,y):
    z = x*x + y
    return z
```

- Add this definition for the function Bob to your file myfirstfunctions.py.
- Create another plain txt file named callbob.py, with the following code inside:

```
from myfirstfunctions import Bob
a = Bob(2,11)
print a
```

- This file you can use to play around with; you can change things in the function Bob and test them by entering python callbob.py in the terminal.

Functions

Input and output

- Remove the return command from your function:

```
def Bob(x,y):
    z = x*x + y
```

Call the function with your python callbob.py script. What happened to the output?

- What if you want to return more than one value? Does the following work?

```
def Bob(x,y):
    z = x*x + y
    w = x*y
    return z
    return w
```

Call the function again; what happens now?

Functions

Input and output

If you want to return more than one output variable, you can use a tuple:

```
def Bob(x,y):
        z = x*x + y
        w = x*y
        return z,w
```

- You can 'unpack' the tuple again when you want to get out separate variables:
  `a,b = Bob(3,4)`, for example

(Lists, dictionaries or sets are also possible as function output. Try it.)

Functions

Input and output

You can define functions without input or output, too. Try:

```
def blabla():
        print "Good morning! How are you today?"
```

Try out what happens if you call this function without an argument:
`blabla()`
... and with an argument, e.g.: `blabla(2)`

- Note that even though this function prints stuff to the screen, it does not return anything as output. Try:
  ```
  a = blabla()
  a
  print a
  a+5
  type(a)
  ```

Functions

Scope of variables

What happens inside a function, stays inside that function, unless it is returned with the return command.

- Variables defined inside the function are not accessible outside of it. We call this *local*.

For example, try the following in the interpreter:

```
def Bob(x,y):
        z = x*x + y
        w = x*y
        return z
Bob(7,8)
print z
```

Functions

Scope of variables

However, Python starts searching for previously defined variables outside of a function, when it does not find them inside:

```
y=12

def Bob(x):
        z = x*x + y
        return z

Bob(7)
```

Functions

Scope of variables

But check the following:

```
y=12

def Bob(x):
        y=3
        z = x*x + y
        return z

Bob(7)
print y
```

- When Python finds the variable y inside the function, it does not bother to look outside for it.
- But once outside, the *'global'* value for y is valid instead.
- Check that the same behavior occurs when y is part of the defined function input (def Bob(x,y): ...).

Functions

Scope of variables

You can call functions from inside other functions; the same scope issues apply here. For example:

```
def Ron(a,b):
        from myfirstfunctions import Bob
        z = Bob(a) + b
        return z
```

- How would you pass a function to another function as an argument? Try it.

Functions

Scope of variables

You can call functions from inside other functions; the same scope issues apply here. For example:

```
def Ron(a,b):
        from myfirstfunctions import Bob
        z = Bob(a) + b
        return z
```

- How would you pass a function to another function as an argument? Try it.

```
def Ron(a,b,c):
        z = c(a) + b
        return z
Ron(7,8,Bob)
```

Functions

Script file formatting: Docstrings

To document your function, you can add a *docstring* below the function definition in your script file.

For example:

```
def examplefunction(a,b):
" " "This function adds two arguments." " "
        z = a + b
        return z
```

Docstrings show up when using the help() function.

Functions

Script file formatting: `if name == __main__:`

- In a scipt file containing multiple function definitions, the line
  if name == __main__:
  only evaluates to True when the file is executed directly; when the file is imported as a module, it evaluates to False.
- Moreover, the block underneath if name == __main__: is evaluated before the rest of the script. Variables defined here are *global* to the script.
- Underneath this line, you can call the functions from your script and define values for their arguments; you can use this to test your script.

Functions

reload(), for example reload(myfirstfunctions), might be a solution for the import problems of this morning... try it.

Functions

Getting user input from the keyboard

To make an interactive program, you can use the function raw_input().
Check the following code:

```
def interactive():
        z = raw_input('Say the magic word...?')
        if z == 'Please':
                print 'Thank you!!'
```