CFG Advanced Python Course - Session 1 (Autumn 2015)
Basics of Python, Part 1

Introducing Python

Python is a powerful, fast, open-source & easy to learn programming language - Python is ideal for first time programmers.

Python's design philosophy has an emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

The core philosophy of the language is summarized by "PEP 20 (The Zen of Python)", which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

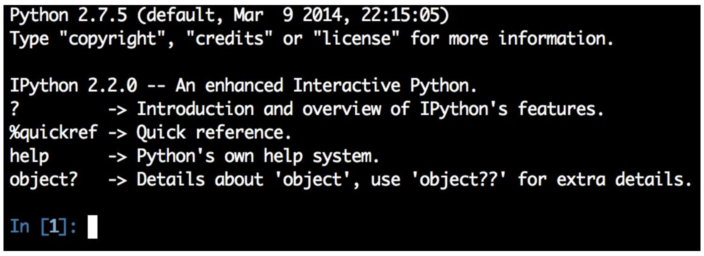
In this course, we will be learning the basics of Python, in a web development context. We will be using a very simple web framework by the name of Flask, which is very lightweight and easy to use.

pip

pip is a package management system used to install and manage software packages written in Python. We will be using pip to install things like Flask and any other dependencies we want to make use of. You should already have pip installed as per the course preparatory work.

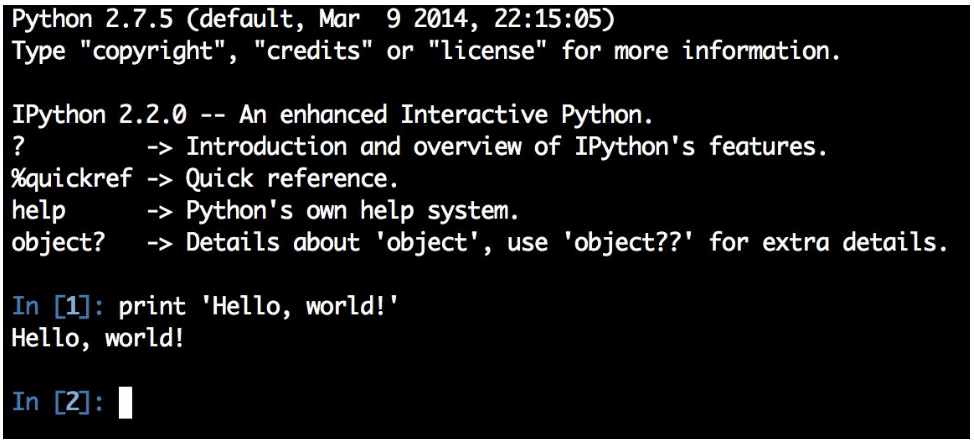
Using Python interactively

You can simply type *python* inside your Terminal / Command Line and hit enter to find yourself in an interactive Python session:



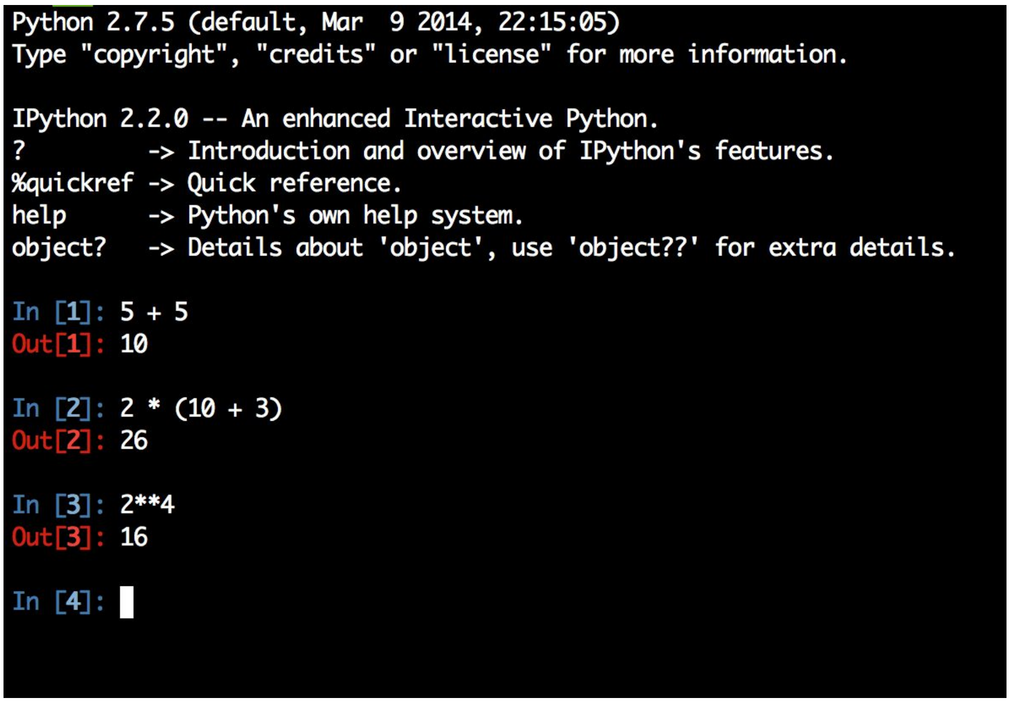
```
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 
```

You can start typing some Python commands - for example you can use Python's *print* command:



```
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: print 'Hello, world!'
Hello, world!

In [2]: 
```

You can also do some maths - give it a go!



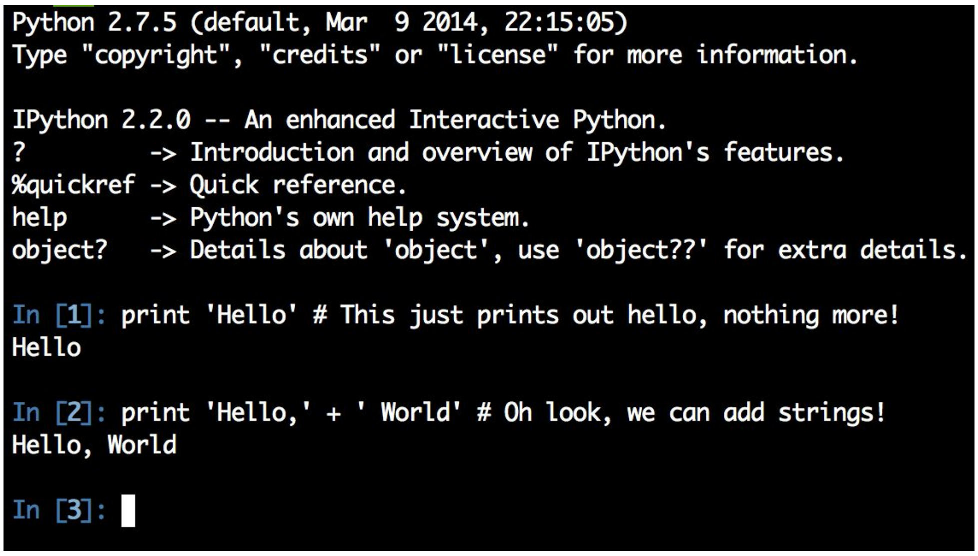
```
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 5 + 5
Out[1]: 10

In [2]: 2 * (10 + 3)
Out[2]: 26

In [3]: 2**4
Out[3]: 16

In [4]: 
```

In a nutshell, you can interactively code in Python, without having to write a program, save it and run it. It's especially useful when you want to try something out.

Comments

In Python, any part of a line that comes after a **#** is ignored. This is useful when you are writing complicated programs, as it allows you to write human-readable comments to document your code - this makes it easier for others to follow your code.

You can see comments being used in action:



```
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: print 'Hello' # This just prints out hello, nothing more!
Hello

In [2]: print 'Hello,' + ' World' # Oh look, we can add strings!
Hello, World

In [3]:
```

Task:

1. Pair up with someone
2. Open an interactive python session on one of your laptops
3. For each of the following Python expressions, try to agree what value you think they evaluate to. Check if you are right in IPython.
    - 1 + 2
    - 5 - 6
    - 8 * 9
    - 6 / 2
    - 5 / 2
    - 5.0 / 2
    - 5 % 2
    - "hello " + "world"
    - "Bob" * 3
    - "Bob" + 3
    - "hello".upper()
    - "GOODBYE".lower()
    - "the lord of the rings".title()

Task Summary

Before we move on to variables, we'll quickly look at a few of the things that you found out in the previous exercise:

```
6 / 2 # 3
5 / 2 # 2
5.0 / 2 # 2.5
5 % 2 # 1
```

If you give Python integers (whole numbers), it will do integer division.

For example, 5 / 2 gives 2 as it is the largest whole number of times you can remove 2 from 5. The partner to integer division is the remainder 5 % 2, giving 1. Together these two operations tell you that 2 goes twice into 5 with remainder 1. If you give Python decimals (floats) it will do normal division.

```
"hello " + "world" # "hello world"
"hello".upper() # "HELLO"
```

Here you met another type of value: "hello" is a string. As you see here, you can add strings together. Like all Python values, strings are also objects and therefore have methods. Here we used the upper method, to change the string into uppercase. You can find out more
about the methods that strings have in the Python docs.

```
"Bob" + 3
TypeError: cannot concatenate 'str' and 'int' objects
```

Turns out that you can't add a string to an integer. Have another read of the message that IPython gave you. Can you figure out what it is saying? When something goes wrong, Python tries to be as helpful as it can.

Learning to interpret the errors that Python gives is an important part of learning to become a programmer.

Names & Variables
So far we have used IPython as a clever calculator, by working with values directly. Programming becomes a lot more powerful when you're able to give values names. A name is just something that you can use to refer to a value. In Python you create a name by using the assignment operator, `=`.

```
age = 5
```

You can change the value associated with a name at any point. It doesn't even have to be the same type of value as the first value assigned to the name. Here we change the value associated with age to a string:

```
age = "almost three"
```

Variables should start with a lower-case letter.

String formatting
String formatting is a way of taking a variable and putting it inside a string. To write a string in Python you can either use ' or ".

```
string1 = 'hello'
string2 = "hello"
```

In the code above, string1 and string2 are exactly the same.

```
age = 5
like = "painting"
age_description = "My age is {0} and I like {1}.".format(age, like)

=> "My age is 5 and I like painting."
```

So what just happened here? Essentially, {} act as placeholders - you can number them, but you don't have to, we could have simply written age_description as:

```
age_description = "My age is {} and I like {}.".format(age, like)

=> "My age is 5 and I like painting."
```

The benefit of using numbers is that you can reuse the same variable multiple times:

```
age_description = "My age is {0} and I like {1}. Did I mention I am {0}?".format(age,
painting)

=> "My age is 5 and I like painting. Did I mention I am 5?"
```
Task:

With your pair, decide what each of the following instructions will do. Test to see if you're right in IPython.

```
a = 1

a+1

a

a = a + 1

a

b = "hello"

b

c = b.title()

b

c

d = "hello"

e = d.title()

d

e

name = "Dave"

f = "Hello {0}! ".format(name)

f

name = "Sarah"

f

f * 5
```

Extra Task for those who finish early:
Note that this exercise has a lot more to do with maths than programming. If you don't get it don't worry!

Consider the expression **x = (2 + 5 * x - x**2) / 5**

- Let x = 1.1
- Write x = (2 + 5*x - x**2) / 5 and then evaluate this multiple times (using the up arrow in Python)
- What happens? Can you explain why?
- Let x = 1 and do the same thing. What happens and why?

Task summary

```
x = 1 # 1
x = x + 1 # 2
```

This might seem really obvious, but it's worth pointing out: = is an assignment operator; it means 'set name on the left equal to the value on the right'. It isn't the same equals as you see in maths! In maths x = x + 1 doesn't really make sense - it's an equation with no (finite) solutions. In Python x = x + 1 makes perfect sense - just set x to be one more than it was before.

```
name = "Dave" # "Dave"
f = "Hello {0}!".format(name) # "Hello Dave!"

name = "Sarah" # "Sarah"
f
"Hello Dave!"
```

The above shows that string formatting happens when you write it down. When you first write f = "Hello {0}!".format(name) Python immediately looks up name and bakes it straight into the string. Setting name to something different later on, won't change this.

In the extra challenge, the expression gave an iterative approximation to sqrt(2). You can tell by rearranging and solving the equation, that any fixed point must be a sqrt of 2. In the final part, by giving it x=1 you forced Python to do integer arithmetic. If you're into that sort of thing, you might like to try and find the fixed points in this case!

Homework

- Finish all exercises from today's class
- Make sure everything we went through makes sense
- Head over to this online exercise book and work your way from **Exercise 2** up to and including **Exercise 10**
- Get excited for our next class