CFG Advanced Python Course - Session 1
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 (libraries) written in Python. We will be using pip to install things like Flask and any other libraries we want to make use of. A library is a collection of code we want to re-use. As an example, we don't want to re-invent or re-write a web server from scratch for each of our web projects. Instead we will use a library to provide us with these functionalities.

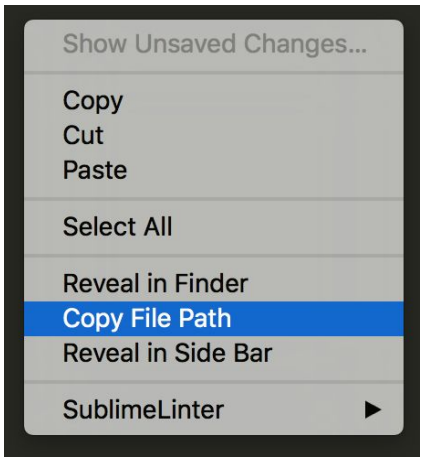
You should already have pip installed as per the course preparatory work.

Creating a Python file
In your preferred Code Editor, e.g. Sublime Text or Atom, create a new file. In this file, let's just type one thing:

```
print 'Hello, World!'
```

Go ahead and save that file now (in a directory / folder that you'll be putting your Python work in, do not call this directory python though), under the name hello.py - the .py extension denotes that we are dealing with a Python file!

Now that we have a Python file - let's run it! To begin with, right click anywhere inside the file you just created - you should see a "Copy File Path option", pick that:



Next up, we will need to bring up the Command Prompt / Terminal. For instructions on how to open the Command Prompt on Windows, read this. On OS X, open your Applications folder, then open the Utilities folder. Open the Terminal application; you may want to add this to your dock.

Once the Command Line is open, type python (all in lowercase, always) followed by a space and then paste the file path you just copied. This will look something like this:

python /users/andreas/cfg-python-work/hello.py

All you need to do now is hit enter and you should see "Hello, World!" being printed out. This is the workflow we will go for; make changes to our Python file, make sure the file path is copied, go to the Terminal / Command Prompt and repeat the above step to run the Python file.

Note: If you are unable to get Python working locally, ask an instructor how to get started with repl.it which lets you write and run Python code from within your browser.

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

```
1
2 print 5 + 5
3
4 print 2 * (10 + 3)
5
6 print 2**4
7
```

Each of the things you've typed is a single Python "statement" - a small piece of code that Python can evaluate to either produce a result or to do something. Python programs are

simply long lists of statements spread across one or more (sometimes thousands!) of files. Python reads and performs each of these statements one after another.

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:

```
1  print 'Hello' # This just prints out Hello, nothing more!
2
3  print 'Hello,' + ' World' # Oh look, we can 'add' strings!
```

Task:

1. Pair up with someone
2. Create a new Python file 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 by printing the result of each expression from within your Python file:
    - 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 decimal numbers (known as "floating point numbers" or "floats" in many programming languages) it will do normal division. Any number with a decimal point is considered to be a float:

```
10.0 / 3.0
=> 3.3333333333333335
```

Here's an example of a different kind of value:

```
"hello " + "world" # "hello world"
```

Here, "hello" is a piece of text. In most programming languages, values like this are called "strings", because they are formed from a string of individual characters. As you see here, you can combine strings using "+", which appends the second one to the first. Python can also do some more clever things to strings:

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

Here, ".upper()" is what's known as a "method" attached to the string. This causes Python to do some processing on the string; in this case, it converts all of the characters to uppercase.
You can find out more
about the methods that strings have in the Python documentation, here. What do you think the ".swapcase()" method does? Look it up in the docs to find out.

Sometimes when you type things in a Python file and run it, you'll see an error message instead of a result:

```
"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 error message that was given out. 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

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 the future. 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"
```

In Python it is conventional for variables names to start with a lower-case letter. If you want to use multiple words in a name, you can separate them with an underscore:

```
a_longer_name = "hello, CFG!"
```

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. It doesn't matter whether you use a single quote or a double quote, but pick one and be consistent!

```
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 which Python fills in using the values of the variables "age" and "like". The placeholders can be numbered, but they don't have to be, 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, like)

=> "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 over time. Test to see if you're right.

```
a = 1

a+1

print a

a = a + 1

print a

b = "hello"

print b

c = b.title()

print b

print c

d = "hello"

e = d.title()

print d

print e

name = "Dave"

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

print f

name = "Sarah"

print f
```

```
print f * 5
```

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!

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 by repeatedly doing this over and over again
- What happens? Can you explain why?
- Let x = 1 and do the same thing. What happens and why?

Useful (and free!) resources to check out when at home

You might find these resources helpful during the course.

- **Code Academy** (www.codeacademy.com) for interactive tutorials on Git, Python, and command line. You'll need to set up a free account if you want to save your progress.
- **Code School** for interactive Python tutorials (www.codeschool.com/courses/try-python) and interactive Git tutorials (www.try.github.io). You may need to set up a free account to get access.
- **Learn Python the Hard Way** (www.learnpythonthehardway.org/book/) for exercises to try out. You don't have to do the study drills he suggests, but it's a great overview of concepts in Python
- **Atlassian** (www.atlassian.com/git/tutorials/) for simple explanations on how to do loads of stuff with Git
- **Stack Overflow**, for help getting unstuck
- **Python's official documentation** (www.python.org) with explanations and examples of how Python works

If you find any sites or tutorials you think are really good, please let us know!