Introduction to Programming with Python
Before we start

The interpreter

Open a terminal window:

- **Windows**
  Press the "Windows key" and the letter "R". You will see the "Run Dialog Box". Type "cmd", and press "OK". Type `python` and press enter. You should see `>>>` appear.

  **OR**: Start Enthought Canopy by clicking on the icon. Locate the console window (where you see `>>>` waiting for you at the top left).

- **Mac**
  Go to Applications > Utilities > Terminal. Type `python` and press enter.
  `>>>`

- **Linux**
  Go to Applications > Terminal. Type `python` and press enter.
  `>>>`
Goals of the course

- To teach you a universal understanding of how programs are written using the basic ingredients of code.

- To provide you with the ability to write programs using the Python language.
Wikipedia: A *programming language* is a formal language designed to communicate instructions to a machine, particularly a computer.

- Formal language: rigid rules, precise statements, unambiguous meaning.

  ...So that a computer can understand it.
  But how??

- Very roughly: 'High-level language' -> 'lower-level language' -> ...-> 'machine code'.

What happens along the arrows?

  - A computer program (a *compiler*) translates the instructions from a high-level language to a lower-level language.
Programming languages

Assembly
.model small, C
.586
.data
mov eax,5
mov ebx,10
add eax,ebx
mov ecx,eax
end

C Language
int main()
{
// assign to the variable result the value of 5 + 10
int result = 5 + 10;
return 0;
}

Python
x = 5 + 10
Basic ingredients of code

- Operators and operands.
- Iteration and flow control
- Logic testing

The above are combined into algorithms which dictate what your code does.
Getting started

Print command

▶ Type print "Hello World" and press enter.
Exiting Python

- To get out of python again: `exit()`
You can write a program in a text file and run it whenever you want

- Make a text file containing the line `print "Hello World"
- Save it in your working directory (as plain text) under the name `helloworld.py`
  - The current working directory is specified in the terminal window when Python is not running.
- To run the file, enter `python helloworld.py` in the terminal window
Getting started
Running a script file – interactive mode

To stay in python after running the file:
▶ Enter `python -i helloworld.py`
  ▶ `-i` stands for 'interactive mode'
Lines preceded by # are comments and are ignored

- In this way, you can include *comments* describing what your code does.

For example, you can write in your txt file:

# This is a practice script
Type `a = 7`, press enter
   ▶ Now you have defined a variable named "a". Type in `a` and press enter to get back its value.

Make another variable called `b`, with value 8
   ▶ Now you can ask Python what is the value of `a+b`, `a-b`, `a*b`...

You can change the value of variable by specifying a new value (e.g. type `a=3` and press enter)
   ▶ Check that `a+b` now has a different value
**Getting started**

**Variables**

**Variable names**

- Variables can have longer names as well, e.g. `temperature=20`, `day1=0.005`
- Variable names can contain any letters or numbers, but cannot start with a number.
- Some words are Keywords and are forbidden from being used as variable names: words which are part of the Python language, such as `print`.

You can assign values to a new variable by setting it to another variable:

- `x=5`
- `y=x`
- `print y`
Basic operators

Standard Math

Add (+), Subtract (-), Divide (/), Multiply (*), Power (**), Modulus (%)
Try:
  ▶ a=5
  ▶ b=3
  ▶ a/b

What’s going on?
  ▶ Now try:
  ▶ b=3.0
  ▶ a/b

Variables defined as $x=3$, $x=5$ etc. are read as integers. $x=3.0$ defines a floating-point number, float for short.
Basic types

Strings

Variables can also be pieces of text:

- `a='hello'`
- `a`
- `print a`

`a` is now of the variable type `string`. 
Basic operators
Increment and decrement operators

- Define a new variable called z, with value 7.
- Try z += 1. What is the value of z now?

+= is called the increment operator,
-= the decrement operator

- Try z = ’yourfirstname’
- z += ’yourlastname’

Note: the statement z += 1 does the same as z = z+1.
3. Basic ingredients of code: Iteration

Loops
3. Iteration

For Loop

With a 'for loop', you can make your program do an action repeatedly.

- Make a txt file with the name testforloop.py, containing the following code (including the tab!):

```
z=1
for i in range(0,5,1):
    z += 1
```

- Run the program from the terminal by entering `python -i testforloop.py`

- What is the value of z now?
3. Iteration
For Loop

What is happening exactly?

► Exit python by entering `exit()`

► Adjust your program to look like this:
  
  ```python
  z=1
  for i in range(0,5,1):
      z += 1
      print "i =",
      print i
      print "z =",
      print z
  
  ► Save it and run it again.
The indented space indicates what is part of the for loop, and what falls outside. For example, check that:

```python
z = 1
for i in range(0,5,1):
    z += 1
    z = z/2
print z
```

is not the same as:

```python
z=1
for i in range(0,5,1):
    z += 1
    z = z/2
print z
```
3. Iteration

Nesting loops

Inside a for loop, you can put another for loop. Try for example:

```python
z = 1
for q in range(0,5,1):
    z += 1
    print "q =",
    print q
    print "z =",
    print z
for r in range(0,3,1):
    z += 0.1*q
    print "r =",
    print r
    print "z =",
    print z
```

- The tab indentation shows which loop is inside which other loop.
- You can nest as many loops inside one another as you want.

Save the example as a script file and run it to see in which order the loops are read.
You can make your program do something if a specified condition is true. For example:

```python
a = 5; b = 6
if a < b:
    print "a is smaller"
```

Try running this code with different values of a and b filled in.

- Note: you can nest if loops inside for loops too, and vice versa.
Often, you want to specify an alternative as well:

```
a = 6; b = 5
if a > b:
    print "a is smaller"
else:
    print "a is larger, or equal to b"
```
If you want to cover several alternatives, you can use `elif` ('else, if'):

```python
a = 5; b = 6
if a < b:
    print "a is smaller"
elif a > b:
    print "b is smaller"
else:
    print "a = b"
```
Flow control - logic testing

Comparison operators

```
a == b
a != b
a <> b
a < b
a > b
a <= b
a >= b
```

These operators check if the condition you specify is true.

- Check how they work, filling in different values (numbers or strings) for a and b. For example:
  ```
a = 5; b = 6
a == b
a != b
```
  etc.
Flow control - logic testing
Logical operators

To combine logical statements:

\[
\begin{align*}
(c \text{ and } d) \\
(c \text{ or } d) \\
\text{not } d
\end{align*}
\]

- For example, try:
  \[
  a = 5; \ b = 6 \\
  (a \neq b) \text{ and } a < b
  \]

- And try:
  \[
  a = 5; \ b = 6 \\
  (a \neq b) \text{ or } a < b
  \]
  Note: is this how you would use or in speaking language?
Make a new txt file called `testwhileloop.py`, containing the following code:

```python
f=7
while f>0:
    f -= 1
    print "working..."
if f==0:
    print "done!"
```

- Run it. What does it do?
- Try adjusting the code and running it with different values for `f`. 
Watch out for the following variant of the program:

```python
f=7
while f!=0:
    f -= 1
    print "working..."
if f==0:
    print "done!"
```
Watch out for the following variant of the program:

```python
f=7
while f!=0:
    f -= 1
    print "working..."
if f==0:
    print "done!"
```

▶ If you fill in a negative number for f, this program will never stop!
The statement **break** enables you to break out of a loop before it has finished. For example:

```python
f = -1

while f!=0:
    f -= 1
    print "working..."
    if f < 0:
        print "I give up."
        break

if f==0:
    print "done!"
```