Introduction
Over the next 6 weeks ...

• An introduction into Computer Science with the aim to improve your overall understanding of the subject.
• An introduction to Python and Greenfoot programming languages.
• An insight into programming, programming constructs and how to utilize these to solve problems.
• Applying programming and problem solving the to Coursework and Controlled Assessments of the GCSE.
General House Keeping

• No Fire Drills Planned! Assembly point is at the car park at the front of the building.

• Each floor is laid out as a square, the toilets on this floor are located on the opposite side of the square.

• Breaks:
  ▪ 15:30 Coffee.
  ▪ 17:30 Coffee and Pizza.
Programming in Python
What is Computer Science?
• **Algorithms**  
  – A cook book for computers.

• **Programming & Development**  
  – Implementing and running recipes.

• **Data & Data Representation**  
  – How do we store numbers?

• **Hardware & Processing**  
  – The components to run our programs.

• **Communication & Networks**  
  – Sending numbers between computers?

• **Information Technology**  
  – Using technology.
The Process of Programming

What is programming?
The Process of Programming

- A computer program is a sequence of instructions and decisions.
- Computers execute very basic instructions in rapid succession.
- Programming is the act of designing and implementing computer programs.

All pictures taken from Python for Everyone by Cay Horstmann et al.
An Introduction to Python

What is Python?
Python is a high-level programming language.

Why learn Python?
It is used a lot within the education of programming as the code is very readable and easy to learn.
print "Hello, welcome to Technocamps!"

response = raw_input("Would you like to learn Python?")
if response == "yes":
    print "Great! Have a look on the interactive zone to get started!"
else:
    print "No problem, how about some of our other workshops?"

Python is a programming language that looks like this:

We use quote marks to input strings!

See the use of indentation to show what code is part of the if statement!

Python is a language that is easy to read and is a great language for beginners!
Today

We will be learning:

- The fundamentals of Programming.
- Input and Output.
- Variables.
- Conditionals.
- Loops.
- Functions.
Before we Begin ... Editing Files

To edit files: **TextWrangler**

‘Source code’ is stored in .py files, e.g. “program.py”
Create one file per program

To compile files: **Terminal**

“cd” – change directory, e.g. cd Desktop
“ls” – list files, e.g. ls Desktop
“python” – to run Python, e.g. python “MyProg.py”

Notes:

PyTHon iS CaSe SeNsItiVe
Commented: `#comment`
Interactive Mode vs Files

Like other languages you can write/save a complete Python program in a file and execute the instructions all in one run.

Alternatively you can run instructions one at a time using interactive mode.

This allows quick ‘test programs’ to be written.
Exercise – Set-up

• Create a folder on the Desktop with your first name e.g. Phil.
• Open a terminal and change the directory into the folder e.g. cd Desktop then cd Phil
• Run “Python” and enter “3+4”. What happens?
• To exit Python type the instruction is “exit()” or press (ctrl-d)
A Hint: Syntax

The programming language syntax relates to the structure, format, combination of symbols/numbers/letters and even spelling!

Don’t worry about making mistakes!
Let's Get Coding
Print Statements
Basic Printing

When learning a new programming language, or learning to program for the first time traditionally the first task to achieve is to output text (usually “Hello world”).

**CODE:**

```plaintext
print "Hello world"
```

**OUTPUT:**

Hello world

The above command will output (print) whatever is between the “” and “.”
Printing more...

<table>
<thead>
<tr>
<th>Syntax</th>
<th>print()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>print(value_1, value_2, ..., value_n)</code></td>
</tr>
</tbody>
</table>

All arguments are optional. If no arguments are given, a blank line is printed.

```python
print("The answer is", 6 + 7, ")")
```

The values to be printed, one after the other, separated by a blank space.
Warning!

Python 2.x and Python 3 are different (Slightly!)

print ("Hello World",2)

Gives:

("Hello World", 2) vs Hello World 2

Today: Forget the brackets as we are using 2.x!
Printing Tasks

**Task 1:**
Start a new file. Save your file as HelloWorld.py. Write and run the traditional “Hello World!” program:

```
print "Hello World!";
```

**Task 2:**
Start a new file, then type out and run the following program:

```
print "My name is", x, "and I am", y, "years old";
```

**Task 3:**
Write a program that prints:

```
Hello
World
```

**Challenge Task:** Write a program that computes and prints the result of:

```
(42*67) / (89*4)
```
Variables
Variables

A variable is a part of the program that can store data. This storage can be named anything, what’s important is the contents of this storage. Fortunately with Python, we do not need to declare what type of content it is e.g. integer, string etc.

**CODE:**

```python
output_text = "Hello world"
print(output_text)
```

**OUTPUT:**

Hello world
A variable is defined the first time it is assigned a value.

```plaintext

total = 0

\cdot \cdot \cdot

\cdot \cdot \cdot

\cdot \cdot \cdot

total = bottles \times BOTTLE\_VOLUME

\cdot \cdot \cdot \cdot \cdot

\cdot \cdot \cdot \cdot \cdot

\cdot \cdot \cdot \cdot \cdot

total = total + cans \times CAN\_VOLUME

\cdot \cdot \cdot \cdot \cdot

\cdot \cdot \cdot \cdot \cdot

\cdot \cdot \cdot \cdot \cdot

Names of previously defined variables

The expression that replaces the previous value

The same name can occur on both sides. See Figure 2.

Names of previously defined variables
Types of Variables

1) A whole number (no fractional part) 7 (integer)
2) A number with a fraction part 8.88 (float)
3) A sequence of characters "Bob" (string)

Conversion examples:

String to Int: `number1 = int("4")`

String to Float: `price = float("9.0")`

String Concatenation: “Hello” + “World”
Variable Tasks

Task 1:  
Write the following program:

```
result = 3+4;
print result;
```

Task 2:  
Modify the above to compute and print result2 which adds 5 to result one.

Task 3:  
Convert result2 to a float and print the float value.

Challenge Task:  
Use the format table at the end of the notes to print result2 to 3 decimal places.
User Input
Raw Input

Raw_input is one of many built in functions for Python, functions that have already been defined. Using raw_input allows us to ask the user a question and collect an input response. E.g. How old are you? The user would answer and whatever they typed would be saved in the variable called “answer”.

**CODE:**

```python
answer = raw_input("How old are you?");
```

**OUTPUT:**

How old are you? *input required*
Printing and Raw Input

**CODE:**
```python
answer = raw_input("How old are you?")
print "You are", answer, "years old"
```

**OUTPUT:**

How old are you?

*user types 17*

You are 17 years old
Warning!

Remember to convert ints and floats!

Strings that look like numbers are not ints or floats!

For example:

```python
number = int(raw_input("Enter a number"));

number2 = float(raw_input("Enter a float"));
```
User Input Tasks

**Task 1:**
Write a program that asks the user for their age and then prints “you are x years old” to the screen. (Hint, you might want to use variables!)

**Task 2:**
Modify the above to ask the user for both their name and age. Then print them.

**Challenge Task:**
Ask the user for the value of an item (Hint item = 10). Then compute and print the amount of VAT (20%) for that item (e.g. 2 for the above item). You may need to use floats!
Libraries
Python Libraries (Modules)

A standard library is a library that is considered part of the language and must be included with any Python system.

For example, to use the `sqrt()` function, which computes the square root of its argument:

```python
# First include this statement at the top of your
# program file.
from math import sqrt

# Then you can simply call the function as
y = sqrt(x)
```
Library Tasks

Task 1:
Write a program that asks the user for the length of two sides of a right angled triangle. Then calculate the length of the hypotenuse. Hint: sqrt((a*a) + (b*b)).

Challenge Task:
Write a program that asks the user for a temperate in degrees Celsius and then converts this to degrees Fahrenheit. Hint: F = (9/5)*C + 32.
Conditionals
Conditional Statements

An if statement can be used to state “if a certain condition occurs, execute the following code. Else, if anything other than that condition occurs, execute this code instead”. This statement can also be expanded to become an IF, ELIF & ELSE statement, so 2 different conditions can be described within IF and ELIF. The IF statements can also combine and nest loops and/or built in functions of Python’s, such as raw_input.

**CODE:**

```python
my_var = 37
if my_var < 50:
    print "It is less than 50!"
else:
    print "It is more than 50!"
```

**OUTPUT:**

It is less than 50!
Conditionals (2)

```python
my_var = 37
if my_var < 50:
    print "It is less than 50!"
elif my_var < 80:
    print "It is more than 50 but less than 80!"
else:
    print "It is more than 80!"
```

Many comparisons, see notes at the end!

**Condition operators:** and, or, not
Conditional Execution

floor > 13?

True

actualFloor = floor - 1

False

actualFloor = floor

if floor > 13:
    actualFloor = floor - 1
else:
    actualFloor = floor
Conditional Tasks

**Task 1:**
Write a program that asks the user their age and calculates whether the user is old enough to drive (17 years +).

**Task 2:**
Write a program that asks the user for the wind speed in miles per hour and converts it to text according to the following:

<table>
<thead>
<tr>
<th>Wind Speed (mph)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 20</td>
<td>Breeze</td>
</tr>
<tr>
<td>20 to 50 (inclusive)</td>
<td>Gale</td>
</tr>
<tr>
<td>More than 50</td>
<td>Storm</td>
</tr>
</tbody>
</table>

**Challenge Task:**
Write a program that asks the user for a amount and whether they want to convert pounds to dollars or dollars to pounds. Then print the result.
Loops
Loops

Sometimes we want to repeat instructions, or get the program to execute a line of code or numerous lines.

**For loop**

This statement enables us to set the amount of times we want to execute the code within the statement (indented).

**While loop**

Repeats a line/lines of code, while a certain condition is true. It tests if the condition is true and executes the code within the statement. If the condition becomes false, this code is not executed.
Loops: While

**CODE:**

counter = 0

while (counter < 10):
    print "The counter is ", counter
    counter = counter + 1

print ("The End.")

**OUTPUT:**

As you can see, counter is the variable being used here. It was created on the first line and assigned the value 0. It will only execute the lines within the while loop, as long as the counter variable value is less than 10. Each time this is true, the variable counter adds 1. As you can see, the number 10 is not printed, only values LESS THAN 10.

This can be expanded to become an WHILE...ELSE statement that can execute lines of code when the condition is no longer true.
Loops: For

**CODE:**

```python
for letters in "I love Python":
    print letters
```

**OUTPUT:**

```
I
l
o
v
e
P
y
t
h
o
n
```

As you can see, `letters` is the variable being used here.

If there are no more letters left in the sequence, the for loop is terminated and no more letters are printed as output.
Loop Execution

while balance < TARGET :
    year = year + 1
    interest = balance * RATE/100
    balance = balance + interest
Warning!

Indentation is key everywhere!

Try the above without indentation!
Range – A Handy Function

Syntax: for variable in range(...) :

statements

This variable is set, at the beginning of each iteration, to the next integer in the sequence generated by the range function.

The range function generates a sequence of integers over which the loop iterates.

- With one argument, the sequence starts at 0. The argument is the first value NOT included in the sequence.

- With two arguments, the sequence starts with the first argument.

- With three arguments, the third argument is the step value.

for i in range(5) :
    print(i)    # Prints 0, 1, 2, 3, 4

for i in range(1, 5) :
    print(i)    # Prints 1, 2, 3, 4

for i in range(1, 11, 2) :
    print(i)    # Prints 1, 3, 5, 7, 9
While Loop Tasks

**Task 1:**
Write a program that prints the sequence 1 to 10 using a while loop.

**Task 2:**
Write a program that prints the sequence 10 to 1 using a while loop.

**Task 3:**
Write a program that sums up the numbers 1 to 10 using a while loop.

**Challenge Task:**
Write a program that asks the user to enter 10 numbers and prints the sum of them.
For Loop Tasks

Task 1:
Write a program that prints the sequence 1 to 10 using a for loop.

Task 2:
Write a program that prints the sequence 10 to 1 using a for loop.

Task 3:
Write a program that sums up the numbers 1 to 10 using a for loop.

Challenge Task:
Write a program that asks the user to enter 10 numbers and prints the sum of them.
A function is a block of re-usable code. It could be coded and re-used numerous times throughout the program you are creating. Python does have some built-in functions that have already been declared, but you can create your own using code similar to below.

**CODE:**

def print_name(x):
    print "Your name is" , x

print_name("Rob")

**OUTPUT:**

Your name is Rob
Function Example: Round

Pass 6.8275 and 2 to round

Compute 6.8275 rounded to two decimal places

Return 6.83 to caller

Store returned value in price variable
Function Syntax

Syntax: `def functionName(parameterName1, parameterName2, ...):` statements

Function header

Function body, executed when function is called.

```python
def cubeVolume(sideLength):
    volume = sideLength ** 3
    return volume
```

Name of function

Name of parameter variable

Return statement exits function and returns result.
Function Tasks

**Task 1:**
Modify your program for computing the hypotenuse so that the code is now inside a function called `computeHypotenuse(a,b)`. Call this function on various inputs.

**Challenge Task:**
Write a function that takes as a parameter the radius of a circle. The function should compute and return the area of a circle with that radius. Call this function on various inputs.

Hint: The math library allows you to import the constant `pi` and `circ=pi*(r*r)`.
Summary

Today we have covered:

- The fundamentals of Programming.
- Input and Output.
- Variables.
- Conditionals.
- Loops.
- Functions.

We have not covered: Pragmatics (rules for using Python) and Language Details.

Next Week: Main Functions and Complex Programs.
Some Useful Information
# Variable Types

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>int</td>
<td>An integer has no fractional part.</td>
</tr>
<tr>
<td>-6</td>
<td>int</td>
<td>Integers can be negative.</td>
</tr>
<tr>
<td>0</td>
<td>int</td>
<td>Zero is an integer.</td>
</tr>
<tr>
<td>0.5</td>
<td>float</td>
<td>A number with a fractional part has type float.</td>
</tr>
<tr>
<td>1.0</td>
<td>float</td>
<td>An integer with a fractional part .0 has type float.</td>
</tr>
<tr>
<td>1E6</td>
<td>float</td>
<td>A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type float.</td>
</tr>
<tr>
<td>2.96E-2</td>
<td>float</td>
<td>Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$</td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td>Error: Do not use a comma as a decimal separator.</td>
</tr>
<tr>
<td>3 1/2</td>
<td></td>
<td>Error: Do not use fractions; use decimal notation: 3.5.</td>
</tr>
</tbody>
</table>
# String Operations

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string = &quot;Py&quot;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>string = string + &quot;thon&quot;</code></td>
<td>string is set to &quot;Python&quot;</td>
<td>When applied to strings, + denotes concatenation.</td>
</tr>
<tr>
<td><code>print(&quot;Please&quot; + &quot; enter your name: &quot;)</code></td>
<td>Prints</td>
<td>Use concatenation to break up strings that don’t fit into one line.</td>
</tr>
<tr>
<td><code>team = str(49) + &quot;ers&quot;</code></td>
<td>team is set to &quot;49ers&quot;</td>
<td>Because 49 is an integer, it must be converted to a string.</td>
</tr>
<tr>
<td><code>greeting = &quot;H &amp; S&quot;</code></td>
<td>n is set to 5</td>
<td>Each space counts as one character.</td>
</tr>
<tr>
<td><code>n = len(greeting)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>string = &quot;Sally&quot;</code></td>
<td>ch is set to &quot;a&quot;</td>
<td>Note that the initial position is 0.</td>
</tr>
<tr>
<td><code>ch = string[1]</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>last = string[len(string) - 1]</code></td>
<td>last is set to the string containing the last character in string</td>
<td>The last character has position <code>len(string) - 1</code>.</td>
</tr>
</tbody>
</table>
Formatting Strings

Syntax: `formatString % (value_1, value_2, ..., value_n)`

The format string can contain one or more format specifiers and literal characters.

```python
print("Quantity: %d Total: %.2f" % (quantity, total))
```

- **Format specifiers**
  - It is common to print a formatted string.
  - The values to be formatted. Each value replaces one of the format specifiers in the resulting string.

- **No parentheses are needed to format a single value.**
# String Formatters

## Table 9 Format Specifier Examples

<table>
<thead>
<tr>
<th>Format String</th>
<th>Sample Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%d&quot;</td>
<td>2 4</td>
<td>Use d with an integer.</td>
</tr>
<tr>
<td>&quot;%5d&quot;</td>
<td></td>
<td>Spaces are added so that the field width is 5.</td>
</tr>
<tr>
<td>&quot;%05d&quot;</td>
<td>0 0 0 2 4</td>
<td>If you add 0 before the field width, zeroes are added instead of spaces.</td>
</tr>
<tr>
<td>&quot;Quantity:%5d&quot;</td>
<td>Quantity: 2 4</td>
<td>Characters inside a format string but outside a format specifier appear in the output.</td>
</tr>
<tr>
<td>&quot;%f&quot;</td>
<td>1.21997</td>
<td>Use f with a floating-point number.</td>
</tr>
<tr>
<td>&quot;%.2f&quot;</td>
<td>1.22</td>
<td>Prints two digits after the decimal point.</td>
</tr>
<tr>
<td>&quot;%7.2f&quot;</td>
<td></td>
<td>Spaces are added so that the field width is 7.</td>
</tr>
<tr>
<td>&quot;%s&quot;</td>
<td>Hello</td>
<td>Use s with a string.</td>
</tr>
<tr>
<td>&quot;%d %.2f&quot;</td>
<td>24 1.22</td>
<td>You can format multiple values at once.</td>
</tr>
<tr>
<td>&quot;%9s&quot;</td>
<td>Hello</td>
<td>Strings are right-justified by default.</td>
</tr>
<tr>
<td>&quot;%-9s&quot;</td>
<td>Hello</td>
<td>Use a negative field width to left-justify.</td>
</tr>
</tbody>
</table>
| "%d%%"               | 2 4 %         | To add a percent sign to the output, use %%.

# Math Library

## Table 5  Selected Functions in the math Module

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt(x)</td>
<td>The square root of ( x ).  ( x \geq 0 )</td>
</tr>
<tr>
<td>trunc(x)</td>
<td>Truncates floating-point value ( x ) to an integer.</td>
</tr>
<tr>
<td>cos(x)</td>
<td>The cosine of ( x ) in radians.</td>
</tr>
<tr>
<td>sin(x)</td>
<td>The sine of ( x ) in radians.</td>
</tr>
<tr>
<td>tan(x)</td>
<td>The tangent of ( x ) in radians.</td>
</tr>
<tr>
<td>exp(x)</td>
<td>( e^x )</td>
</tr>
<tr>
<td>degrees(x)</td>
<td>Convert ( x ) radians to degrees (i.e., returns ( x \cdot 180/\pi ))</td>
</tr>
<tr>
<td>radians(x)</td>
<td>Convert ( x ) degrees to radians (i.e., returns ( x \cdot \pi/180 ))</td>
</tr>
<tr>
<td>log(x)</td>
<td>The natural logarithm of ( x ) (to base ( e )) or the logarithm</td>
</tr>
<tr>
<td>log(x, base)</td>
<td>of ( x ) to the given ( base ).</td>
</tr>
</tbody>
</table>
Operators for Conditionals and Loops

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Relational Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Python</strong></td>
<td><strong>Math Notation</strong></td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
</tr>
</tbody>
</table>

Compare strings: $x == y$ for equal and $x != y$ for not equal.