Programming in Python
Today...

We will:

- Look at WJEC samples.
- Greenfoot overview (Brief for now!).
- Look at lists.
- Cover for loops a little more.
- Look at reading and writing to files.
Re-cap
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.**
• **Comments:** `# comment`
A Note On Global Variables

Variables declared inside a function can only be assigned to within that function.

If you want a “global” variable, simply put it at the top of your file (after imports).

You can then use this within a function using the “global” keyword.

For Example:

```python
x = 5

def f():
    global x
    x = x+1
    print x
```
Slides Online! 😊

The slides are available at:

http://cs.swan.ac.uk/~cspj/teaching/pythonCPD/

They will be updated weekly.
WJEC Samples
Previously: Design a Calculator

The teacher has decided that pupils’ would benefit from an on screen calculator they could use in lessons to carry out these calculations. He thinks that they need a calculator with the following functions:

-- addition.
-- subtraction.
-- multiplication.
-- division.
-- memory store.
-- memory recall.
-- memory clear.

Your task is to write an application to produce the on screen calculator.
Previously: Exam Marks

The Head of History has asked you to write an application that will help her store and analyse the test marks for her Year 11 pupils.

Your task is to write an application to:
-- allow the teacher to input the pupils’ name and the pupils’ test mark
-- calculate the grade (based on some grade table)
-- store the data
-- output the number of pupils who achieve each grade
-- output the names of pupils who achieve an A or A* grade.
Current: Sports Club

The Head of Sports has asked you to write an application that will help her store and analyse team scores and results.

Your task is to write an application to:
-- Input the scores from each match.
-- Calculate and store the points earned by each team.
-- Record if the points are home or away points.
-- Display all the teams and their total number of points.
-- Calculate the number of away points in the case of a draw at the end of the competition.
Current: The Caesar Cipher

Write a program that computes the Caesar shift cipher. This encodes messages by “shifting” each letter a number of places through the alphabet.

Your task is to write an application to:
-- Input a message to be coded or decoded.
-- Input the offset.
-- Display a message that has been coded or decoded.
-- Store coded messages and their offsets.
-- Recall, decode and display a stored message.
Greenfoot and Java
Java vs Python

Some major differences!

- Static vs dynamic typing:
  
  E.g. `Int x = 5;` vs `x = 5`

- Braces vs indentation:
  
  E.g. `if(x==5){
      //do something
    }` vs `if(x==5):
    //do something`

- Semi colons to end expressions in java.
- Forced main method to run! (also must use classes/objects!)
Greenfoot Demo

Greenfoot is a visual tool to teach Java and object orientation.

Idea:

1. Create objects (worlds and actors) to make a game.

2. Add Java code to each actor to make them move etc.
Lists
Why Lists?

A Computer Program often needs to store a sequence of values and then process them...

For example, if we are reading a file line by line how many variables would we need? line1, line2, line3....

Lists to the rescue!

Note: often called ArrayList (similar but not quite Arrays).
# List Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To create a list:</td>
<td>[ value_1, value_2, \ldots ]</td>
</tr>
<tr>
<td>To access an element:</td>
<td>listReference[index]</td>
</tr>
</tbody>
</table>

**Name of list variable**

- `moreValues = []`
  - Creates an empty list

- `values = [32, 54, 67, 29, 35, 80, 115]`
  - Creates a list with initial values
  - **Initial values**

**Use brackets to access an element.**

- `values[i] = 0`
- `element = values[i]`
Example: Creating/Accessing

# 1: Creating a list
values = [32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]

# 2: Accessing a list element
values[5] = 87

Note: Strings can be accessed like lists!
List Length

It is often useful to get the number of elements in a list.

For this there is a built in function!

\[
\text{numElements} = \text{len(values)}
\]

Very useful for looping through all elements of a list...
Many Ways to Traverse Lists

# First version (list index used)
for i in range(10) :
    print(i, values[i])

# Better version (list index used)
for i in range(len(values)) :
    print(i, values[i])

# Third version: index values not needed (traverse
# list elements)
for element in values :
    print(element)
## Lists Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l.pop()</code></td>
<td>Removes the last element from the list or from the given position. All elements following the given position are moved up one place.</td>
</tr>
<tr>
<td><code>l.pop(position)</code></td>
<td></td>
</tr>
<tr>
<td><code>l.insert(position, element)</code></td>
<td>Inserts the element at the given position in the list. All elements at and following the given position are moved down.</td>
</tr>
<tr>
<td><code>l.append(element)</code></td>
<td>Append the element to the end of the list.</td>
</tr>
<tr>
<td><code>l.index(element)</code></td>
<td>Returns the position of the given element in the list. The element must be in the list.</td>
</tr>
<tr>
<td><code>l.remove(element)</code></td>
<td>Removes the given element from the list and moves all elements following it up one position.</td>
</tr>
<tr>
<td><code>l.sort()</code></td>
<td>Sorts the elements in the list from smallest to largest.</td>
</tr>
</tbody>
</table>
Example: Append

```python
#1
friends = []

#2
friends.append("Harry")

#3
friends.append("Emily")
friends.append("Bob")
friends.append("Cari")
```
Example: Remove

```
friends = ["Harry", "Cindy", "Emily", "Bob", "Cari", "Bill"]
friends.pop(1)
```
List Tasks

Task 1:
Complete the following tasks in a main function:
(a) Create a list \( x \) of integers containing the following numbers: 8, 4, 5, 21, 7, 9, 18, 2, and 100.
(b) Print the number of items in the list (by using an appropriate function).
(c) Print the third item in the list.
(d) Print the last item in the list (do not hard code the last index).
(e) Append the value 42 to the end of the list.

Task 2:
Create a function that uses a for loop to print all the values in the list (Hint pass the list to the function).

Challenge Task:
Create a function that uses a for loop to print all the values in the list with labels to indicate the index of each element. For example: \( x[2] = 5 \) or \( x[4] = 21 \).
Reading and Writing Files
Text Files

Text Files are very commonly used to store information:

1. They are one of the most ‘portable’ types of data files.
2. Allow for persistent storage between program runs.

Can anyone think of an example use?
Reading Files

To access a file, you must first open it.

Suppose you want to read data from a file named input.txt, located in the same directory as the program.

To open a file for reading, you must provide the name of the file as the first argument to the open function and the string "r" as the second argument:

```python
infile = open("input.txt", "r")
```

The open function returns a “file object” which represents the file.
Reading Files (2)

To read an entire line python provides:

```python
line = infile.readline()
```

Can be used within a loop to read the entire file!

```python
line = infile.readline()
while line != "":
    # Process the line.
    line = infile.readline()
```
Writing to Files

Writing to files is pretty similar:

To open the file for writing:

```python
outfile = open("output.txt", "w")
```

To write a string to the file:

```python
outfile.write("Hello, World!\n")
```
Some Warnings

You always need to close the files when you are finished:
E.g.

```python
infile.close()
outfile.close()
```

When reading lines, they are read with the ‘\n’ character. Use the strip() function to remove these! E.g. line.strip().
Overview: Reading and Writing

Store the returned file objects in variables.

```
infile = open("input.txt", "r")
outfile = open("output.txt", "w")
```

The name of the file to open

Specify the mode for the file:
"r" for reading (input)
"w" for writing (output)

Read data from `infile`.
Write data to `outfile`.

Close files after the data is processed.

```
infile.close()
outfile.close()
```

If you fail to close an output file, some data may not be written to the file.
File Tasks

Task 1:
Write a function “write()” that creates a new text file called marks.txt and writes a series of marks to it. For example:
   65
   78
   99
   ...

Task 2:
Create a function “read()” that reads the numbers from the above text file and prints them.

Challenge Task:
Modify the read function to store the values in a list and return the list. Then write another function that finds and prints the highest mark in the list.
Summary

Today we have covered:

- WJEC Examples.
- Greenfoot and Java.
- Lists.
- Reading and Writing Files.

We have not covered: Object Orientation (classes etc.) and user interfaces.

Next Week: Object orientation and a chosen big task 😊