Programming in Python
Today...

We will:

✓ Explore classes further.
✓ Look at GCSE examples.
✓ Re-cap constructs.

The slides are available at:

http://cs.swan.ac.uk/~cspj/teaching/pythonCPD/
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.
GCSE Tasks
Last Week: Exam Marks

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

Your task is to write an application to:
-- Read in data from a file, each line contains a student and a mark.
-- Calculate the grade (based on some grade table).
-- Store the data (Hint: lists?)
-- Output the number of pupils who achieve each grade
-- Output the names of pupils who achieve an A or A* grade.
Re-Cap: Class Diagram

Driver
  +main()

MarkTool
  +pupils: List[Student]
  +ReadInStudents()
  +PrintAllA()
  +PrintNumberOfGrades()

Student
  +name: String
  +mark: Int
  +calculateGrade(): String
Implement The Solution: Steps

1. Create Student and MarkTool classes (without any methods).
2. Write a method to read in data from a file, where each line contains a student and a mark.
3. Extend 2 to create student objects and add them to a list (within MarkTool).
4. Add a calculate grade method to the Student class (see table on next slide).
5. Write a method in MarkTool to print students who achieved an A or an A*.
6. Write a method in MarkTool to print how many students got each grade.
7. Create a “Driver” class that creates a new MarkTool and calls the various methods above.

Extension: provide a menu to the teacher allowing her to add grades for students. These should then be written to the file.
Use the following grade table:

<table>
<thead>
<tr>
<th>Mark</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>A*</td>
</tr>
<tr>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>60</td>
<td>B</td>
</tr>
<tr>
<td>50</td>
<td>C</td>
</tr>
<tr>
<td>40</td>
<td>D</td>
</tr>
<tr>
<td>20</td>
<td>E</td>
</tr>
<tr>
<td>&lt;30</td>
<td>Fail</td>
</tr>
</tbody>
</table>
Sports Club

The Head of Sports has asked you to write an application that will help her store and analyze 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.
Sports Club Tasks

Task 1:
Discuss and draw a class diagram for the problem.

Task 2:
Break down the programming into a series of smaller tasks.

Task 3:
Implement a solution (feel free to re-use code from your other solution).
Python “Cookbook”
Printing

Syntax

\[
\text{print()}
\]
\[
\text{print(}value_1, \ value_2, \ldots, \ value_n)\]

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

\[
\text{print("The answer is", } 6 + 7, "!")
\]

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

Remember: No Brackets!
Variables

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

```
total = 0
  .
  .
  total = bottles * BOTTLE_VOLUME
  .
  .
  total = total + cans * CAN_VOLUME
```

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”
User 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”))
```
Formatting Strings

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

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

No parentheses are needed to format a single value.

It is common to print a formatted string.

Format specifiers

The values to be formatted. Each value replaces one of the format specifiers in the resulting string.

Example:

```python
print("Quantity: \%d Total: \%10.2f\" % (quantity, total))
```
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.

Condition operators: and, or, not
Loops: WHILE

**CODE:**
```python
counter = 0
while (counter < 10):
    print "The counter is", counter
    counter = counter + 1
print "The End."
```

**OUTPUT:**
```
0
1
2
3
4
5
6
7
8
9
```
Functions

**Syntax**

```python
def functionName(parameters):  
    statements
```

- **Function header**
  - Name of function
  - Name of parameter variable

- **Function body**
  - Executed when function is called.
  ```python
def cubeVolume(sideLength):
    volume = sideLength ** 3
    return volume
  ```

- **Return statement**
  - Exits function and returns result.
## 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.

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

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

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

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

```python
for i in range(1, 11, 2) :
    print(i)  # Prints 1, 3, 5, 7, 9
```

With three arguments, the third argument is the step value.
Main Functions

By convention, main is the starting point of the program.

```python
def main():
    result = cubeVolume(2)
    print("A cube with side length 2 has volume", result)

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

main()
```

The cubeVolume function is defined below.
## List Syntax

**Syntax**

- To create a list: \([value_1, value_2, \ldots]\)
- To access an element: \(listReference[index]\)

### Name of list variable

- moreValues = []
- values = [32, 54, 67, 29, 35, 80, 115]

### Use brackets to access an element.

- values[i] = 0
- element = values[i]
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)
Reading and Writing Files

The name of the file to open

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

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

Store the returned file objects in variables.

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

Close files after the data is processed.

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

If you fail to close an output file, some data may not be written to the file.
Example Class: Counter

class Counter :

    def __init__(self) :
        self._value = 0

    ## Gets the current value of this counter.
    def getValue(self) :
        return self._value

    ## Advances the value of this counter by 1.
    def click(self) :
        self._value = self._value + 1

    ## Resets the value of this counter to 0.
    def reset(self) :
        self._value = 0
Example: Using Classes

```python
from counter import Counter

# Create a Counter object
result = tally.getValue()
print("Value:", result)
```

```python
tally = Counter()
tally.reset()
tally.click()
tally.click()
result = tally.getValue()
print("Value:", result)
tally.click()
result = tally.getValue()
print("Value:", result)
```
Further Useful Information
## Variable Types

### Table 1: Number Literals in Python

<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><strong>Error:</strong> Do not use a comma as a decimal separator.</td>
</tr>
<tr>
<td>3 1/2</td>
<td></td>
<td><strong>Error:</strong> 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>string = &quot;Py&quot;&lt;br&gt;string = string + &quot;thon&quot;</td>
<td>string is set to &quot;Python&quot;</td>
<td>When applied to strings, + denotes concatenation.</td>
</tr>
<tr>
<td>print(&quot;Please&quot; + &quot; enter your name: &quot;)</td>
<td>Prints&lt;br&gt;Please enter your name:</td>
<td>Use concatenation to break up strings that don’t fit into one line.</td>
</tr>
<tr>
<td>team = str(49) + &quot;ers&quot;</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>greeting = &quot;H &amp; S&quot;&lt;br&gt;n = len(greeting)</td>
<td>n is set to 5</td>
<td>Each space counts as one character.</td>
</tr>
<tr>
<td>string = &quot;Sally&quot;&lt;br&gt;ch = string[1]</td>
<td>ch is set to &quot;a&quot;</td>
<td>Note that the initial position is 0.</td>
</tr>
<tr>
<td>last = string[len(string) - 1]</td>
<td>last is set to the string containing the last character in string</td>
<td>The last character has position len(string) - 1.</td>
</tr>
</tbody>
</table>
# String Formatters

<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:</td>
<td>Characters inside a format string but outside a format specifier appear in the output.</td>
</tr>
<tr>
<td>&quot;%.2f&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>2 4 1.22</td>
<td>You can format multiple values at once.</td>
</tr>
<tr>
<td>&quot;%9s&quot;</td>
<td></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>
<tr>
<td>&quot;%d%%&quot;</td>
<td>2 4 %</td>
<td>To add a percent sign to the output, use %.</td>
</tr>
</tbody>
</table>
Operators for Conditional and Loops

<table>
<thead>
<tr>
<th>Python</th>
<th>Math Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>Not equal</td>
</tr>
</tbody>
</table>

Compare strings: \( x == y \) for equal and \( x != y \) for not equal.
Summary

We have explored in some depth, programming in Python:

- Basic Constructs.
- Functions, Lists and Files.
- Classes and Objects.
- Programming in the large.

You now have all the tools you need to program just about anything! Just remember:
1. Libraries and the internet are your friend.
2. Explaining your program to others will help your own understanding.
Thank You!