

**T**eaching **L**ondon **C**omputing

## **A Level Computer Science**

# **Topic 1: GCSE Python Recap**



# Aims

- *What is programming?*
  - Recap GCSE Python
    - Programming concepts
    - Learning programming
  - Outline the course
    - Contents
    - A level curricula
    - Materials
-

# What is Programming?

- What is the essence of Programming?
  - Do we learn Python or Programming?
-

# Problem: Largest Item

- Find the largest one

27	12	53	13	25	68	18	66	23	15	36
----	----	----	----	----	----	----	----	----	----	----

- Definition:
    - One of the numbers in the list
    - No other number in the list is larger
  - Programmer needs **a recipe for finding it!**
-

# Problem: Largest Item

- Here is a recipe

```
largest ← first item
while more items in list
    item ← next item
    if item greater than largest
        largest ← item
```

- Not yet a program!
-

# Programming

- Recipe / procedure / algorithm for solving a problem
- ... in a language understood by a computer



Problem solving

Writing a Program

# Computational Thinking

*“... the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent”*

- Concepts
    - **algorithmic thinking**: developing steps
    - **decomposition**: spotting the parts of a problem
    - **abstraction**: ignoring unnecessary detail
    - **generalisation**: making the specific more general
    - **evaluation**: how good is it?
-

# Practical Exercises 1.1 – 1.3

- Describe a recipe for finding the **two** largest items in a list





# Challenge Solution

- Describe a recipe for finding the **two** largest items in a list

```
flrgst ← max(first item, second item)
slrgst ← min(first item, second item)
while more items in list
    item ← next item
    if item GT slrgst
        if item GT flrgst
            slrgst ← flrgst
            flrgst ← item
        else
            slrgst ← item
```

# Challenge – Simpler Solution?

- Describe a recipe for finding the **two** largest items in a list

```
largest ← largest item of list  
remove largest from list  
second largest ← largest item of new list
```

---



# **Programming Concepts**

---

# Language Concepts

Concepts	Python Example(s)
Literal (Constant)	<code>'abc'    "william"    42    3.14159</code>
Expression	<code>2**3**4        "he"+"llo"        x &gt; 5</code>
Assignment	<code>area = length * width</code>
Sequence	<code>x = 7 y = x - 5</code>
Selection	<code>if x &gt; 5:</code>
Loop	<code>while x &lt; 7:</code>
Function call	<code>print("ans=", ans)</code>
Function definition	<code>def area(l, w):</code>

# Python Types

- Python values belong to types:
  - String
  - Integer
  - Floating point
  - Boolean
  - List
- Types are ‘dynamic’
  - Try the ‘type’ function
- Variable are not declared, but must be initialised before they are referenced

There are  
more types!

# Python Expressions

Type	Operators	Functions/Methods
Numbers	<code>+</code> <code>-</code> <code>*</code> <code>**</code> <code>/</code> <code>//</code> <code>%</code>	<code>str()</code> <code>abs()</code> <code>pow()</code>
String	<code>+</code>	<code>len()</code> <code>int()</code> <code>.append()</code> <code>.remove()</code> <code>.split()</code> <code>.count()</code>
Boolean results	<code>==</code> <code>!=</code> <code>&gt;</code> <code>&lt;</code> <code>&gt;=</code> <code>&lt;=</code>	
Boolean operands	<code>and</code> <code>or</code> <code>not</code>	
List	<code>in</code> <code>not in</code> <code>[:]</code>	<code>len()</code> <code>min()</code> <code>max()</code>

# Python Library

- The library is a large collection of functions, methods and module
    - E.g. random module
  - Standard library: all Python installations
  - Other packages: installed when needed
  - Issue
    - Very complex: need to guide students
-

# Practical Exercise 2.4

- Look up the standard library online; read the introduction
  - In Chapter 2, (Built in Functions), lookup and try
    - the `bin()` function
    - the `bool()` function
    - the `range()` function
  - In Chapter 4 (Built in Types), lookup
    - `<<` operator, e.g. `10 << 2`
    - `divmod()` function
    - Behaviour of a slice with 3 arguments: `[m:n:o]`
-



# Scope

- What variables are available at any place?
  - Python uses **dynamic scoping**:
    - Variables added by executing an assignment
    - Other languages are different
  - Local versus Global scope (*simplified*)
    - Local: inside a function
    - Global: not in a function
-

# Practical Exercise 2.5 – 2.6

- Try the following program. Can you explain it?

```
def printX():  
    print("This is x:", x)  
  
y = int(input("Enter a num> "))  
if y % 2 == 0:  
    x = y  
  
printX()
```

# Errors (Practical Exercise 2.7)

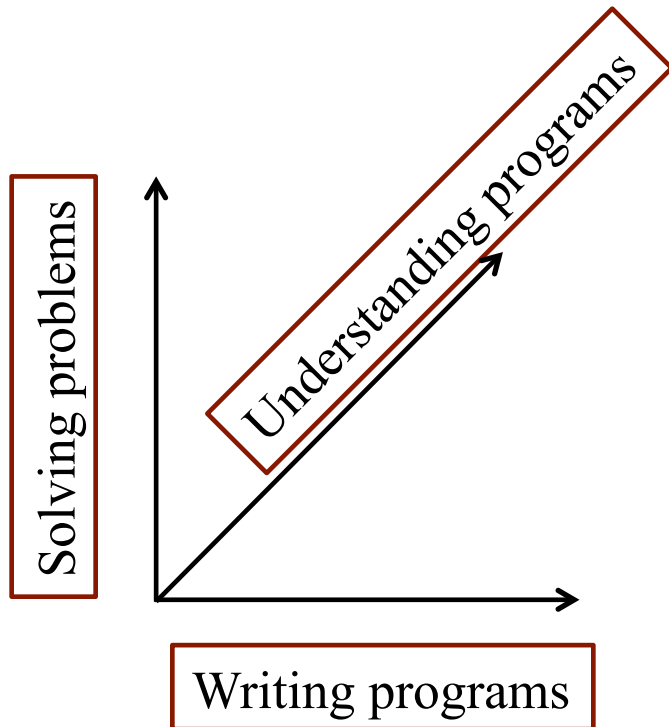
- Syntax error
    - Letters do not form a Python program
    - E.g. brackets do not match
  - Name (Scope) error
    - Using a name that is not defined
    - E.g. Reading a variable that has not been initialised
  - Type errors
    - Operation not available for these types
    - E.g. "william" – "iam"
  - Evaluation error
    - Expression cannot be evaluated
    - E.g. "william"[99]
-



# **Learning Programming**

---

# Axes of Learning



- Understanding programs
    - Can predict result of program
    - Test it: *what does this program do?*
  - Writing programs
    - Choose expressions and statements
    - Find bugs
    - Test it: *implement / complete short program*
  - Solving problem
    - Problem → recipe
    - Test it: *solve open problems*
-

# Deepening and Mental Models

- Deeper understanding
    - Initial understanding may be useful but partial
    - Challenge and refine
  - Example – assignment
    - Understanding 1: assignment gives a name to expression
    - Understanding 2: assignment overwrites an existing value in a variable (memory)
    - Understanding 3: ...
-

# Example: Understanding Assignment

- Assignment as naming:
- ... but what about:

```
width = 10
height = 17
area = height * width
print(area)
```

```
numA = ...
numB = ...
if numA < num B:
    numB = numA
    numA = numB
print("The larger number is in numA")
```

```
lst = [1,3,5,7]
lst[3] = lst[1]
lst[0] = lst[0] + lst[1]
print(lst)
```

```
myFriends = ["John", "Jo"]
yourFriends = myFriends
myFriends.remove("Jo")
yourFriends.append("Jane")
print(myFriends)
```

# Practical Exercises 3.1 – 3.2

- Explore the sequence of ideas involved in learning if statements





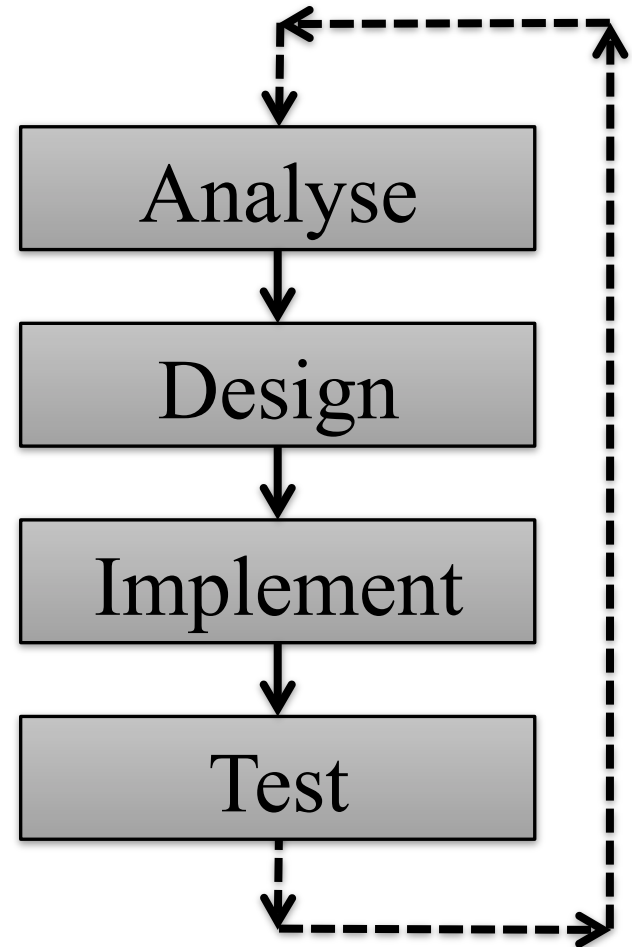
# Problem Solving

- Understand problem
    - Try example
  - Simplify the problem
  - Design the solution
    - What is the input and output?
  - Half-way techniques
    - Pseudo code
    - Flow charts
- 



# Software Development

- Do you:  
**Just Write a Program?**
- Lifecycle
  - Describes the process
- Analyse
  - Understand problem
  - → specification
- Design
  - Arrange parts
  - How to design?



# Characteristics of a Programmer

- Analytical
    - Understanding problems
    - Think like a computer
  - Imaginative
    - Invent solution
    - Design program
  - Resilient
    - Trying different solutions
    - Looking for bugs
-

# **Course Aims and Overview**

---

# Topics

1. Programming and s/w development
  - Arrays and recursion
  - Object-oriented programming
  - Software design techniques
2. Algorithms and data structures
  - Sorting and searching
  - Trees and lists
3. Computer systems
  - Computer architecture
  - How programs are run
4. Theory of computing
  - Finite state machines
  - Language syntax and processing

Computational  
Thinking

# A Level Curricula

- In transition: new from 2015
  - A2 and AS separate
  - Questions combine topics
- Differences between boards
  - Mainly looked at: OCR, AQA

Details differ: themes the same

---

# Course Materials

- Each session
    - Slides
    - Practical exercises
    - References notes
  - Same sections
  - Practical work
    - During slides
    - After presentation
  - Folder and web site
-

# Summary

- Programs are recipes for a computer
    - Recipe to solve a problem
    - Program in a particular language
  - Learning programming combines
    - Problem solving
    - Understanding programs
    - Writing programs
-



# Practical Exercises 3.3

- Planning a complex program

