Python CPD (1)

Frans Coenen
Department of Computer Science
UNIVERSITY OF LIVERPOOL

Content

- Session 1, 9:30-11:00 (Background and Sequence).
- Session 2, 11:30-13:00 (Selection, Lists, Dictionaries and File Handling).
- Session 3, 13:40-15:00 (Repetition and "Putting it all together").
- Materials at:

http://www.liv.ac.uk/computer-science/continuing-professional-development/

Background and Introduction

UNIVERSITY OF LIVERPOOL

Reminder

- A computer can be conceptualised as comprising many switches that can be set to '0' or '1'.
- We arrange these "switches" into groups of eight called Bytes.
- We can perform simple operations (for example add or subtract) on these groups using a small set of instructions called machine code or byte code.
- In the early days of computing programming was done in byte (machine) code, however it is both extremely time consuming and very error prone.

High-Level Programming Languages

- A solution to speeding up the programming process, and reduce the associated risk of errors, is to use a high level programming language such a Python.
- High-level languages tend to use natural language constructs and/or automate certain aspects of programming (such as memory management), hence they are easier to use.
- However, a program written in a high-level language such as Python cannot be *run* directly. To execute a computer program written in a high-level language it must be either *compiled* or *interpreted*.

Compilers v. Interpreters

- A compiler translates (converts) source code written in a high level language into a machine executable form.
- The advantage is that the machine executable form runs much faster than if it were interpreted (see below).
- The disadvantage is that different machines and operating systems have different machine codes associated with them, consequently to compile a program under (say) windows would require a different compiler to that needed to do the same under (say) Apple OS.
- An interpreter steps through each line of the high-level source code and "decodes", the source is never translated into machine code. Different interpreter are required for different languages (and different machines). Interpretation is much slower than compilation.
- Python is typically interpreted (although compilers exist).

Logging-on and The Python Interpreter

UNIVERSITY OF LIVERPOOL

The Python Interpreter (1)

- Logon using your password and open a "terminal window".
- In the terminal window type: `python`
- Now try the following:

```
123+222

100**2

print("Hello")
```

```
a=100
b=2
a**b
b**a
```

```
J='JAVA'
len(j)
j[0]
j[1:3]
```

The Python Interpreter (2)

- To exit the compiler type: `quit()`
- Thus we can use the Python interpreter in an interactive manner as either a calculator or (say) to test bits of code.
- However, as soon as we quit the interpreter we loose everything we have written!
- We want to create our programmes so that they are stored in a permanent manner for multiple use.
- There are a number of ways we can do this, we will be using a "text editor".

Text Editors

UNIVERSITY OF LIVERPOOL

Text Editors

- There are many different text editors available.
- Windows usually comes with a number of these.
- We will be using `IDLE` which comes with Python.

Programme Constructs

UNIVERSITY OF LIVERPOOL

Programme Constructs

- Programming is founded on three basic constructs:
  1. Sequence
  2. Selection
  3. Repetition

Programming Construct One: Sequence

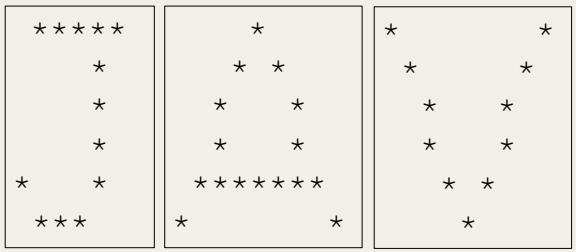
UNIVERSITY OF LIVERPOOL

Problem Example 1: Giant Letters

UNIVERSITY OF LIVERPOOL

Giant Letters Requirements

Design and implement a Python program that writes "JAVA" vertically down the screen using giant letters made up of strings of * characters and blank spaces. (Do not use the "tab" character!)



Giant Letters Source Code

- Load `PythonExampleProblems \Sequence\GiantLetters \giantLeters.py` into the `IDLE` text editor

Giant Letters Comments (1)

- Comments are important from a Software Engineering perspective (readability leads to understandability which leads to maintainability).
- Comments in Python start with a `#` character (Python does not support multi-line comments).
- Again for sound software engineering reasons we like to divide are code up into chunks.
- In Python the simplest way to do this is to define the chunks as "functions" or "methods".

Giant Letters Comments (2)

- Anatomy of a function:

```
def <FUNCTION_NAME> ( <ARGUMENTS_IF_ANY> ) :
      <CONTENT>
```

- Indenting is important (there is no end-of-function punctuation mark).
- If necessary we can break up a line using the `\` character.
- In Python functions have to be defined before they can be used.

Giant Letters Comments (3)

- Note how we output strings:

```
print("<OUTPUT>")
```

- The "escape sequence" `\n` is a newline character.
- Note that the `giantLetterA()` function is called twice, we do not write the function twice (good software engineering means writing code in an efficient manner).

Run The Giant Letters Source Code

- From the `IDLE` editor window menu select `Run – Run Module` or simply select `F5`.

Problem Example 2: Swimming Pool

UNIVERSITY OF LIVERPOOL

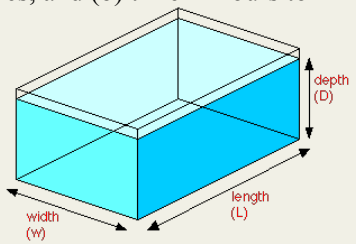
Swimming Pool Introduction

- Software programmes take "input" and transform it into "output".

- In the case of our giant letters programme the input was simply an instruction to run the code.
- Clearly to do anything more meaningful we need more sophisticated input.

Swimming Pool Requirements

Develop a Python program which; given the width, length and depth (in metres), of a swimming pool determines and outputs: (a) the volume in litres, and (b) time in hours to fill the swimming pool.
Assume that the rate of flow into the pool is 2.5 litres per second. Note: 1 litre = 1000 cubic centimetres, therefore 10 litres = 0.01 cubic metres, hence 1 cubic metre = 1000 litres.



Volume (Litres) = L x W x D x 1000

Time to fill pool (Hours) = $\frac{\text{Volume}}{\text{Rate (Litres/sec)} \times 3600}$

Swimming Pool Source Code

- Load `PythonExampleProblems\Sequence\SwimmingPool\swimmingPool.py` into the `IDLE` editor

Swimming Pool Comments (1)

- Note how we input values into a Python programme:

```
<VARIABLE_NAME> = input("<STRING>")
```

- This allows us to input a string, if we want a integer we need to "cast" it into this type:

```
<VARIABLE_NAME> = int(input("<STRING>"))
```

- By convention constants are indicated using capital letters for the item name.
- Note how we return values from functions.
- Note how we pass arguments to functions.

Swimming Pool Comments (2)

- By convention constants are indicated using capital letters for the item name. Note how we return values from functions.
- Note how we pass arguments to functions.
- Note the formatted output:

```
print('<STRING> = {<FORMAT_SPECIFIER}'. \
            format(<VARIABLE>))
```

Swimming Pool Comments (2)

- In the Python interpreter try the following:

```
x = 123.456789
print(x)
print('|',x,'|')
print('|{0:.3f}|'.format(x))
print('|{0:10.3f}|'.format(x))
print('|{0:^10.3f}|'.format(x))
print('|{0:<10.3f}|'.format(x))
print('|{0:>10.3f}|'.format(x))
```

- Exit the Python interpreter.

Run The Swimming Pool Source Code

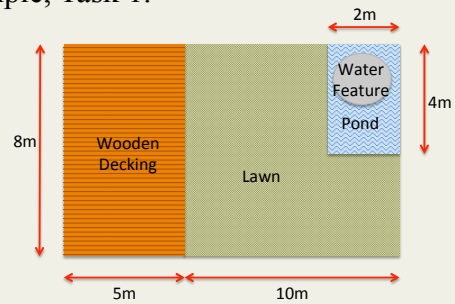
- In the `IDLE` editor window select `F5`.

Problem Example 3: Landscape Gardening I (Task 1)

UNIVERSITY OF LIVERPOOL

Landscape Gardening Introduction

- AQA GCSE Specimen Controlled Assessment example, Task 1.



Landscape Gard. I Requirements

- (Taken from AQA GCSE specimen). Customers provide a landscape gardening company with a plan. Costs are as shown in the table. There is also a labour charge of £16.49 for every hour of work done. Create a Python programme that: (a) allows a user to input lawn and patio dimensions and the number of water features required (if any); and (b) outputs the cost for each, the labour cost and the total cost.

Work to be done	Cost of materials	Time to install
Laying a lawn	£15.50 per m^2	20 mins per m^2
Laying a concrete patio	£20.99 per m^2	20 mins per m^2
Installing a water feature (e.g. a fountain)	£150.00 each	60 mins each

Landscape Gard. I Source Code

- Load `PythonExampleProblems\Sequence\LandsGardQuote1\landsGardQuote.py` into the `IDLE` editor.

Landscape Gard. I Comments

- Note how we can return multiple values from a function. (Take care because the ordering is important!)
- Note how functions are reused (good software engineering practice).

Run The Landscape Gardening I Source Code

- Note, current software only considers lawns, patios and garden lights. Try adding another item.

Adding More Items to The Landscape Gardening I Source Code

- Need to add cost and time constants for new item.
- Need to include a function call (with appropriate parameters) to required input function.
- Need to include a function call (with appropriate parameters) to required output function.
- Need to revise `totalTime` and `labourCost` calculations.
- Need to revise `materialCost` and `totalCost` calculations.

Coffee Time?

UNIVERSITY OF LIVERPOOL