*FACTFILE:*

# GCSE DIGITAL TECHOLOGY

## Unit 4

DIGITAL DEVELOPMENT CONCEPTS

Rewarding Learning

## Programming Constructs 1

### Learning Outcomes

Students should be able to:
understand and use the following constructs in a programming language:

- variables, constants, and Boolean and arithmetic operators;
- input, output and assignment statements;
- one-dimensional array structures.

### Content

- Python Preliminaries
- Variables & Constants
- Assignment Statements
- Arithmetic & Boolean Operators
- Arrays & Lists

### Python Preliminaries

The examples and explanations in this fact file use the Python programming language. The same general principles, however, apply to many high level programming language - including Java and C#. Occasional comparisons and contrasts will be drawn with other languages, where this is appropriate.

The material in this fact file extends and builds upon the Python material here:

- U4FF3: Digital Data

The code samples below assume that Python is being used in interactive mode, unless otherwise noted.

#### Variables & Constants

From the point of view of the programmer, a variable is just a name that refers to a value - this is explained in Fact File 3. To achieve this the programming language compiler/interpreter must set aside a particular memory location where the data value can be stored.

**Variable**
"[A variable] is the identifier (or name) associated with a particular memory location used to store data."
*BCS Glossary of Computing, 13th edition, p 286.*

In many programming languages it is also possible to use constants. A constant is similar to a variable except that once a constant has been given an initial value, that value cannot be changed.

**Constant**
"[A constant] is a data item with a fixed value."
*BCS Glossary of Computing, 13th edition, p 289.*

The Python language has variables, but it does not have constants.

Variables and constants are both available in the programming languages Java and C# - although, in Java the term final variable is used to mean a data item with a fixed value.

## Assignment Statements

The process of associating a variable name with a value is called *assignment*.

**Assignment**
"[Assignment] is an instruction that gives (assigns) a value, which could be the result of a calculation, to a specified variable. The value is placed in the memory location corresponding to the variable."
*BCS Glossary of Computing, 13th edition, p 264.*

Here are some example assignment statements in Python.

```
>>> length = 3
>>> width = 4
>>>
```

These have the effect of associating the numeric values 3 and 4, with the variables named length and width, respectively. Of course, variables may also be assigned string values - for example:

```
>>> name = 'Fred'
>>> day = 'Monday'
>>>
```

In some programming languages it is necessary to declare a variable's type (e.g. string, int) before assigning it a value. This is not required in Python - Python is able to infer the type of a variable from its context, in much the same way that you can probably infer that (for example) the variable name is of type string and the variable width is of type int.

## Input & Output

In Fact File 3 we saw how the print function is used to print a message to the screen. For example:

```
>>> print('Hi there!') Hi there!
>>>
```

This is an example of data output, because it involves data (the string Hi there ) being passed out of the computer to the user. Data input is the reverse process, where data passes from the user into the computer. To illustrate data input and output it is more convenient to use Python in scripted mode. See Fact File 3 for more on the distinction between interactive mode and scripted modes in Python.

Consider this Python Script.

```
name = input('Please type your name: ')
greeting = 'Hi there ' + name + '!'
print(greeting)
```

When this code is run Python displays the following text:

```
Please type your name:
```

The user is then able to respond by typing his name.

```
Please type your name: Bob
```

Finally, Python responds by printing a personalised greeting.

```
Please type your name: Bob Hi there Bob!
>>>
```

The first line of code in the script is doing a lot of work. It combines data output with data input and assignment - in a single line of code.

- **Data output:** the string *"Please type your name"* is displayed.
- **Data input:** whatever the user enters in reply is read by the script - in this case the string *"Bob"*.
- **Assignment:** the variable *name* is assigned the value *"Bob"*.

The second line of code in the script assembles a personalised greeting by concatenating three strings together using the "+" operator. See Fact File 3 for more on string concatenation.

The third, and final, line in the script uses the print function to display the personalised greeting.

The input function in Python always returns a string. This can be a little confusing if you are expecting the user to enter numeric data. Consider this example script.

```
n1 = input('Enter the first number: ')
n2 = input('Enter the second number: ')
print(n1+n2)
```

If this is run and the user enters (for example) the values 3 and 5, the following interaction will occur.

```
Enter the first number: 3
Enter the second number: 5
35
>>>
```

In this case, Python interprets the values 3 and 5 as strings (i.e. "3" and "5"). Consequently the "+" operator inside the print statement performs string concatenation rather than arithmetic. Hence the string value "35" is printed rather than the numeric value 8.

Fortunately, Python provides a function (called int) for converting strings into corresponding integers. Here's how you might use the int function if you wish to perform arithmetic on user input.

```
n1 = input('Enter the first number: ')
n2 = input('Enter the second number: ')
print(int(n1)+int(n2))
```

Alternatively, this script could be used.

```
n1 = int(input('Enter the first number: '))
n2 = int(input('Enter the second number: '))
print(n1+n2)
```

These two scripts are equivalent and either one would produce the following interaction if the user entered the values 3 and 5..

```
Enter the first number: 3
Enter the second number: 5
8
>>>
```

## Arithmetic & Boolean Operators

In Fact File 3 we learnt about the following numeric data types:

- int - used to store integer values;
- float - used to store numbers with fractional parts.

Python provides a range of operators for manipulating numeric types, including:

**Python Arithmetic Operators**

| Operator | Description | Example | |
|---|---|---|---|
| + | Addition | 6 + 2 | (= 8) |
| - | Subtraction | 6 - 2 | (= 4) |
| * | Multiplication | 6 * 2 | (= 12) |
| / | Division | 6 / 2 | (= 3) |
| ** | Exponentiation | 6 ** 2<br>2 ** 3 | (= 6 squared = 36)<br>(= 2 cubed = 8) |

**Python Arithmetic Comparison Operators**

| Operator | Description | Example | |
|---|---|---|---|
| == | Equality comparison | 6 == 6<br>6 == 4 | (= TRUE)<br>(= FALSE) |
| = | Inequality comparison | 6  = 6<br>6  = 4 | (= FALSE)<br>(= TRUE) |
| > | Greater than | 6 > 6<br>6 > 4 | (= FALSE)<br>(= TRUE) |
| < | Less than | 6 < 6<br>6 < 4<br>6 < 8 | (= FALSE)<br>(= FALSE)<br>(= TRUE) |
| >= | Greater than or equal to | 6 >= 6<br>6 >= 7 | (= TRUE)<br>(= FALSE) |
| <= | Less than or equal to | 6 <= 6<br>6 <= 7 | (= TRUE)<br>(= TRUE) |

In Fact File 3 we saw that the Boolean operators (and, or, not) are used to manipulate Boolean values (TRUE, FALSE) in much the same way that the arithmetic operators (+, -, *, /) are used to manipulate numeric values (1, 2, 3...). We also learnt that Boolean values and operators are frequently used in conditional expressions.

The following Python script illustrates the use of a conditional expression, as well as some of the other concepts that have been introduced in this Fact File.

```
r1 = 'I see that you enjoy computer programming. '
r2 = 'You have very a bright future. '
r3 = 'I see that you do not enjoy computer programming. '
r4 = 'Oh dear! '
r5 = 'You have a preference for the Python Language.'
r6 = 'You clearly have excellent taste. '
r7 = 'You have a preference for the Java Language. '
r8 = 'It takes all sorts, I suppose! '
r9 = 'You like both Python and Java. '
r10 = 'Move over Bill Gates! '

print('Answer using a scale of 0 (dislike) to 10 (like).')

progLike = int(input('How much do you like programming. '))
pythonLike = int(input('How much do you like Python. '))
javaLike = int(input('How much do you like Java. '))

if progLike >= 5 and pythonLike > javaLike:
        response = r1+r2+r5+r6
elif progLike >= 5 and pythonLike < javaLike:
        response = r1+r2+r7+r8
elif progLike >= 5 and pythonLike == javaLike:
        response = r1+r2+r9+r10
else:
        response = r3+r4

print(response)
```

The script prompts the user for some of their views on computer programming, and gives a personalised message in response.

- It begins by initialising a number of string variables (r1..r10). These will later be assembled into the personalised response.
- Next a print statement is used to instruct the user to answer using a 0 to 10 scale.
- Next come three input statements. These ask the user some questions relating to computer programming. Within these statements:
  - The int function is used to convert the user's response from a string to an int.
  - The variables progLike, pythonLike and javaLike are assigned the resulting integer values.
- Next comes a conditional statement. This assembles the personalised response message, and assigns the resulting value to the variable, response. There are four different possibilities, depending on the values of the variables progLike, pythonLike and javaLike.
  - Case 1: user likes programming (ProgLike>=5) and prefers Python to Java.
  - Case 2: user likes programming and prefers Java to Python
  - Case 3: user likes programming and likes Python and Java equally.
  - Case 4: user does not like programming.

- Finally, a print statement is used to display the response string.

Some sample interactions with the script are shown below.

```
Answer using a scale of 0 (dislike) to 10 (like). How much do you like computer programming. 2 How
much do you like Python. 8
How much do you like Java. 9
I see that you do not enjoy computer programming. Oh dear!
>>>

Answer using a scale of 0 (dislike) to 10 (like). How much do you like computer programming. 9 How
much do you like Python. 9
How much do you like Java. 8
I see that you enjoy computer programming. You have very a bright future. You have a preference for
the Python Language. You clearly have excellent taste.
>>>

Answer using a scale of 0 (dislike) to 10 (like). How much do you like computer programming. 7 How
much do you like Python. 4

How much do you like Java. 5
I see that you enjoy computer programming. You have very a bright future. You have a preference for
the Java Language. It takes all sorts, I suppose!
>>>

Answer using a scale of 0 (dislike) to 10 (like). How much do you like computer programming. 7 How
much do you like Python. 8
How much do you like Java. 8
I see that you enjoy computer programming. You have very a bright future. You like both Python and
Java. Move over Bill Gates!
>>>
```

You will notice that there are four interactions above - one corresponding to each leg of the conditional expression. As a programmer, this gives you confidence that the script does not contain any errors. It is good practice when testing any program to ensure that all possible execution paths are tested.

## Arrays & Lists

Often a programmer wishes to group together a collection of related data items. Most programming languages - Python included - provide a variety of different methods to achieve this. One widely used method uses arrays.

**Array**
"[An array] is a set of data items of the same type grouped together using a single identifier. Each of the data items is is addressed by the variable name and a subscript."
*BCS Glossary of Computing, 13th edition, p 326.*

In order to use use an array in Python it is first necessary to import the *array module* from the *Python Standard Library*. As you gain experience in the use of Python you will learn that many useful features are available via the Standard Library. Fortunately, it is very straightforward to import a module - requiring only a single line of Python code. In this case, all that is needed is the following line of code, which may be used in interactive mode or included in a script.

```
>>> from array import *
>>>
```

This instructs Python to import all of the features of the module called *array*, and make them available to the programmer.

The following code, for example, creates an array, called *my_array*, containing the integers 1, 2 and 3. The value ‘i’ means that the array is only allowed to contain integers.

```
>>> my_array = array(‘i’, [1,2,3])

>>>
```

We can use a print statement to display the array.

```
>>> print(my_array) array(‘i’, [1, 2, 3])
>>>
```

We can display individual items contained in an array. Each one is referred to by its index (within square brackets). The index of an item is simply its position within the array and, of course, we start counting at 0 rather than 1.

```
>>> print(my_array[0]) 1
>>> print(my_array[2]) 3
>>>
```

We can add an item to the end of an array using the append method.

```
>>> my_array.append(4)
>>> print(my_array) array(‘i’, [1, 2, 3, 4])
>>>
```

We can insert an item into an array at a specified location using the insert method.

```
>>> my_array.insert(0,0)
>>> print(my_array) array(‘i’, [0, 1, 2, 3, 4])
>>>
```

We can remove a specified item from an array.

```
>>> my_array.remove(2)
>>> print(my_array) array(‘i’, [0, 1, 3, 4])
>>>
```

The array module provides more methods but these are some of the most important ones.

Python arrays are a little restrictive in the data types that they may contain. Fortunately Python provides other ways of grouping related data together. One of the most commonly used is the Python *list*. In Python, lists are similar to arrays except that they may contain a wider range of data types - indeed a single list may contain data items of different types.

The following code, for example, creates a list, called *my_list*, containing the values 1, ‘two’ and 3.0. You will notice that the first value is an int, the second is a string, while the third is a floating point number. It is not possible to mix data types in this way in an array.

```
>>> my_list=[1,'two',3.0]
>>> print(my_list) [1, 'two', 3.0]
>>>
```

You can see below that accessing a data item in a list is similar to the corresponding operation in an array.

```
>>> print(my_list[0]) 1
>>> print(my_list[2]) 3.0
>>>
```

Similarly, the append, insert and remove methods work in the same way for lists as for arrays.

```
>>> my_list.append('FOUR')
>>> print(my_list) [1, 'two', 3.0, 'FOUR']
>>> my_list.insert(0,'ZeRo')
>>> print(my_list)
['ZeRo', 1, 'two', 3.0, 'FOUR']
>>> my_list.remove('two')
>>> print(my_list) ['ZeRo', 1, 3.0, 'FOUR']
>>>
```

Because of their flexibility lists are more frequently used than arrays in Python. Arrays and lists may also be used in the programming languages Java and C#.

## Resources

- Downey, AB., Think Python: How to Think Like a Computer Scientist (2nd Edition), http://greenteapress.com/wp/think-python-2e/
- Downey, AB., Think Java: How to Think Like a Computer Scientist, http://greenteapress.com/wp/think-java/
- Learn Python.org, http://www.learnpython.org/
- Python Software Foundation, https://www.python.org/
- Python 3.5.2 Documentation, https://docs.python.org/3/
- Miles, R., C# Programming Yellow Book (7th Edition), University of Hull, http://www.csharpcourse.com/