# FACTFILE:
# GCSE DIGITAL TECHNOLOGY
## Unit 4
### DIGITAL AUTHORING CONCEPTS

Rewarding Learning

## Digital Data 2

### Learning Outcomes

Students should be able to:

- describe and use appropriately the following data types: numeric (integer and real), date/time, character and string; and
- demonstrate understanding of and use Boolean operators (AND, OR and NOT) and truth tables.

### Content:

- Variables, Values and Types
- Boolean Data and Boolean Operators
- Computational Thinking

### Variables, Values and Types

The concepts of *value*, *variable* and *type* are fundamental to many programming languages. The examples and explanations in this fact file relate specifically to the *Python* programming language, but they could easily be adapted to other languages.

A *variable* is a name that refers to a *value* – for example:

- We might use the variable **shoe_size** to refer to the value **9**.
- We might use the variable **greeting** to refer to the value **'Hi there!'**.

Values belong to types – for example:

- The value **9** belongs to the type **integer**.
- The value **'Hi there!'** belongs to the type **string**.

Note that:

- String values – such as 'Hi there!' – are enclosed between quotation marks.
- Numeric values – such as 9 – are not enclosed between quotation marks.

Integers are simply whole numbers – either positive or negative. These include:

... –3, –2, –1, 0, 1, 2, 3 ...

Integers do not, however, include numbers with fractional parts, such as ½ or 1.25.

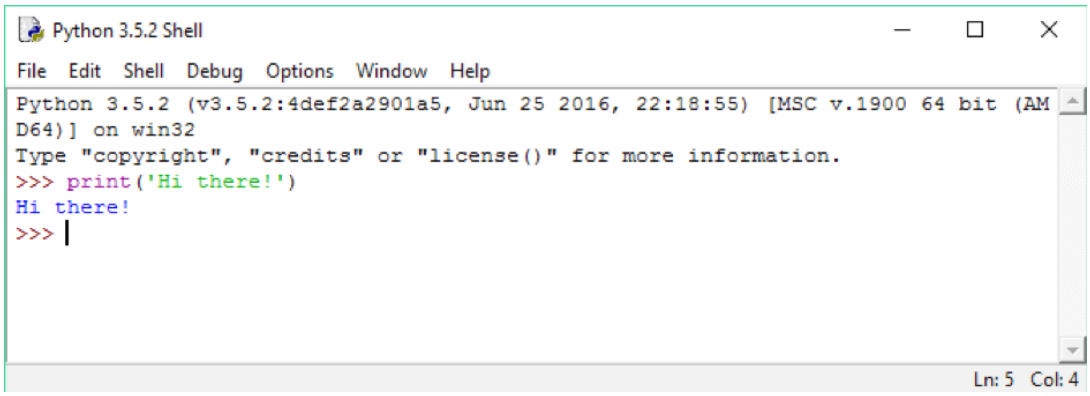
**Integer Type**
"*Integer type* data are whole numbers, either positive or negative."
BCS Glossary of Computing, 13th edition, p 338.

## Interactive Python Preliminaries

We will use Python in two different ways (or modes) in the examples in this fact file:

- Interactive mode;
- Scripted mode.

In interactive mode, instructions are typed directly into the Python shell, which looks like this.




In scripted mode, instructions are entered into a text file, which Python then interprets.

All examples below use interactive mode, unless explicitly marked as scripted.

You will get more out of this guide if you have access to an interactive Python shell while you read it – that way you can try some of the examples for yourself.

Python indicates its readiness to interact with the user by displaying what is known as the *primary prompt* in the shell window:

```
>>>
```

The user can interact with Python by typing an instruction next to the prompt. For example, the print function can be used to instruct Python to print something to the screen.

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

You can see that when the print instruction has been successfully carried out (executed), Python displays the prompt again. In an interaction with Python, such as the one above, anything preceded by the prompt should be understood to be the user's input. Everything else (including the prompt itself) should be understood as Python's output.

A variable can be assigned a value using the equality operator – for example:

```
>>> greeting = 'Hi there!'
```

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

Of course, Python can also be instructed to print integer values – for example:

```
>>> print(9)
9
>>>
```

You can ask Python to tell you the *type* of any value by using the *type* function.

```
>>> type('Hi there!')
<class 'str'>
>>>
```

Python's response indicates that **'Hi there!'** is of type **'str'** – in other words **'Hi there!'** is a **string**. Similarly, Python can tell us that **9** is an **integer** – for example:

```
>>> type(9)
<class 'int'>
>>>
```

Variables can be used with integers in much the same way that they can with strings.

```
>>> shoe_size = 9
print(shoe_size)
9
>>>
```

## Numbers

Programming languages generally supply a method for performing arithmetic, and Python is no exception. The usual arithmetic operators are used to perform *addition*, *subtraction*, *multiplication* and *division* – for example:

```
>>> print(9+3)
12
>>>
```

It turns out that it is not strictly necessary to use the print function. You can type any arithmetic expression at the prompt, and Python will respond by evaluating the expression and returning the corresponding value. For example:

```
>>> 9+3
12
>>> 9-3
6
>>> 9*3
27
>>> 9/3
3.0
>>>
```

You will notice that, in response to the final expression above, Python returns the value 3.0, where you may have expected simply the value 3. This is because division of integers, in general, produces a *fraction* rather

than another integer. For this reason Python does not use the integer data type for the result of a division – instead it uses a type known as **float** (floating point number).

```
>>> type(3)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>> 9/3
3.0
>>> type(9/3)
<class 'float'>
>>>
```

**Float type / Real type**

"*Real type* data are numbers that include a fractional part. [...] Also sometimes called *float type*."

BCS Glossary of Computing, 13th edition, p 338

You can freely mix integers and floating point numbers in arithmetic expressions and allow Python to work out an appropriate type for the resulting value. For example:

```
>>> 2+3.0
5.0
>>> type(2+3.0)
<class 'float'>
>>>
```

Python will evaluate arithmetic expressions that contain more than just one operator. For example:

```
>>> 1+2
3
>>> 1+2+3
6
>>> 1+2-3
0
>>> 1+2*3
7
>>>
```

Notice that in the final expression above the multiplication (i.e. 2*3) must be carried out before the addition (i.e. 1+6) in order to arrive at the correct result (i.e. 7). This is the normal mathematical convention. If we had intended the addition to be carried out first then parentheses could have been used to make this clear. For example:

```
>>> (1+2)*3
9
>>>
```

It is good programming style to use parentheses to help make your intended meaning clear to the reader – even if it's not strictly needed. For example:

```
>>> 1+(2*3)
7
>>>
```

## Strings

Programming languages generally supply methods for manipulating strings, and Python is no exception.

**String Data**

"*String Data* is textual data in the form of a list of characters, for example words and punctuation. String data is made up of character data and will usually vary in length."

BCS Glossary of Computing, 13th edition, p 331.

In processing strings we might want to:

- combine two strings into one;
- find out how many characters are in a given string; or
- find substrings within a given string.

**Joining Strings Together**

The example below illustrates how strings may be joined together. This is called *string concatenation*.

```
>>> laugh1='Ho'
>>> laugh2='Ha'
>>> print(laugh1)
Ho
>>> print(laugh1+laugh2)
HoHa
>>> print(laugh2+laugh1+laugh2)
HaHoHa
>>> print(laugh2*3)
HaHaHa
>>> LOL=Laugh2*2+laugh1*2+laugh2)
>>> print(LOL)
HaHaHoHoHa
>>>
```

You will notice that the arithmetic operators for addition (+) and multiplication (*) are used here with strings, but they are performing string concatenation rather than arithmetic. Python is able to determine the appropriate operation to perform by inspecting the types of values being used. When used with strings:

- The + operator performs *string concatenation* when applied to strings.
- The * operator performs *repetitve concatenation* by joining together a specified number of copies of the same string.

## How Long is a Piece of String?

Python provides a specific function – called **len()** – that returns the length of a given string – i.e. the number of characters that it contains.

```
>>> len('Ha')
2
>>> len('Ha'+'Ho')
4
>>> type('Ha')
<class 'str'>
>>> type(len('Ha'))
<class 'int'>
>>>
```

The **len()** function always returns an integer and it may be used in an arithmetic expression just like any other integer.

```
>>> len('Ha')+2
4
>>> len('Ha')+len('Ho')
4
>>> 3*len('Ha')
6
>>>
```

### Looking Inside a String

Python provides a way to look inside a string to pick out specific substrings.

```
>>> print('Fred'[0])
F
>>> print('Fred'[1])
r
>>> print('Fred'[2])
e
>>> print('Fred'[3])
d
>>>
```

You will see that these expressions pick out the first, second, third and fourth characters (respectively) of the string 'Fred'. Note that the first character is not identified by the number 1 (as you might expect) – instead it is identified by the number 0. In computing we normally start counting from zero rather than one.

The following examples are slightly different from the ones above, in that they return substrings of varying lengths.

```
>>> print('Fred'[0:1])
F
>>> print('Fred'[0:2])
Fr
>>> print('Fred'[1:4])
Red
>>> print('Fred'[0:2]+'Wilma'[1:5])
Frilma
>>>
```

Looking inside a string in this way is known as *string slicing*.

## Dates and Times

Python provides some special facilities to process dates and times. These are not, however, part of the basic Python language. Instead they are contained in an external module – called **datetime** – that must be imported before the facilities can be used. Fortunately, importing this module is very straightforward – it requires only the following instruction.

```
>>> import datetime
>>>
```

Once the module has been imported we can create new *date* and *time values*. For example, the code below assigns the value **1 April 2017** to the variable **today**, and prints the result.

```
>>> today = datetime.date(2017,4,1)
>>> print(today)
2017-04-01
>>>
```

Similarly, the code below assigns the the time value **4:30** to the variable **now**, and prints the result.

```
>>> now = datetime.time(4,30,0)
>>> print(now)
04:30:00
>>>
```

It is also possible to access the individual components (year, month & day ) of a *date value*, as well as the individual components (hour, minute & second) of a *time* value.

```
>>> print(today.month)
4
>>> print(now.second)
0
>>>
```

Python's **datetime** module is very powerful and and offers many more features than we have described here.

## Boolean Operators

Most programming languages supply a data type called **Boolean** or **bool**, which consists of the two special values, **TRUE** and **FALSE**.

**Boolean Data**

"*Boolean data* or *logical data* can only have two values, true or false. This makes it easy to use the values of Boolean variables to control the flow of a program"

BCS Glossary of Computing, 13th edition, p 331.

We can ask Python to evaluate the truth of an expression as follows:

```
>>> 3==3
True
>>> 3==2
False
>>> 'Fred'=='Fred'
True
>>> 'Fred'=='Wilma'
False

>>> type(True)
<class 'bool'>
>>> type(3==3)
<class 'bool'>
>>>
```

Note that the double equal sign (==) is used as the *relational operator* to check for equality. The following relational operators are also available.

```
x != y      x is not equal to y
x > y         x is greater than y
x < y         x is less than y
x >= y      x is greater than or equal to y
x <= y      x is less than or equal to y
```

The most common use for Boolean types is in a *conditional statement*. A conditional statement enables a programmer to determine alternative paths through her code, depending on specified conditions. For example the following short Python program has two possible paths. It will either print the word *Adult* or the word *Child* – depending upon the value of the variable *Age*.

```
Age=21
if Age>=18:
 print('Adult')
else:
 print('Child')
```

Note there are no Python prompts in this code – this is because it was designed to use scripted mode. The examples below also use Python in scripted mode.

It is possible to create *more than two* alternative execution paths, using the **elif** construct (else if). For example:

```
Age=13
if Age>=20:
 print('Adult')
elif Age>=13:
 print('Teenager')
elif Age>=3:
 print('Child')
else:
 print('Baby')
```

Just as numeric data can be manipulated using arithmetic operators (+ – * /), Boolean data may be manipulated using the Boolean operators: **and**, **or** and **not**. For example:

```
if Age>=13 and Age<19:
     print('Teenager')

if Temperature<0 or WindSpeed>40 or RainFall=='heavy':
     print('Weather is nasty - stay at home today')

if not Temperature>15:
     print('Do not wear shorts today')
```

I'm sure you can infer the meaning of these operators just by thinking about the examples. If a more formal definition is needed a *truth table* may be used.

**Truth Table 1: P1 and P2**

| P1 | P2 | P1 and P2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE |
| TRUE | FALSE | FALSE |
| FALSE | FALSE | FALSE |

The Truth Table 1 (above) defines the **and** operator, by showing how the value of the Boolean expression **P1 and P2**, depends on the individual values of **P1 and P1**.

- Row 1 tells us that, when **P1** is true and **P2** is true, the expression **P1 and P2** is also true.
- The remaining rows tell us that in all other circumstances, the expression **P1 and P2**, is false.

Truth Table 2 (below) defines the **or** operator, by showing how the value of the Boolean expression **P1 or P2**, depends on the individual values of **P1** and **P1**.

**Truth Table 2: P1 or P2**

| P1 | P2 | P1 and P2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | FALSE | FALSE |

Truth Table 3 (below) defines the **not** operator, by showing how the value of the Boolean expression **not P**, depends on the value of **P**.

**Truth Table 3: not P2**

| P | Not P |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |

## Resources

### Programming Languages


- 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/