University Of Portsmouth
BSc (Hons) Computer Science
First Year

Programming

M30299
September 2022 - May 2023
20 Credits

Thomas Boxall
up2108121@myport.ac.uk

Contents

S.1. Module Introduction

20-09-2022 14:00 Nadim & Matthew PK2.23

Module Aims

This module will build up programming skills either from scratch or from where you are currently. It will give you the basic knowledge; guidance, help and feedback to help develop programming skills.

Importantly, this module is 40 credits. It spans across the entire year.

Programming

Programming is the process of constructing computer programs, this encompasses analysing the problem, designing the algorithm, implementing the algorithm and testing the algorithm.
We write the programs in a programming language.
For the first $\frac{3}{4}$ of the year, we'll use Python 3 and for the final $\frac{1}{4}$ of the year, we'll use Dart. Dart is similar to Java. We will be the first cohort to use Dart.
Programming is a skill, which can only be developed through practice and should be fun! Having a good understanding and ability to program is important later during in the course and for careers.

Module Organisation

For this module, there will be content shared on Moodle (notes for lectures and videos complementing the notes) and timetabled sessions (in some, fundamental ideas will be covered which will make it possible to complete the weekly worksheets). Worksheets will be released weekly onto Moodle, these should be completed before the practical class of the following week.
Monday at 3pm in RB LT1 is the tutorial class. You need to go through the notes on Moodle before the sessions.
Practical classes are 1 hour 50 minute sessions in a computer lab. The main purpose of these is to get feedback on the worksheets.

Support

The academic tutors (Xia and Eleni) can be booked on Moodle.
There are drop-in sessions on Monday in the FTC. This session is optional and is designed for targeted questions or issues which can't be resolved in the tutorial/ practical classes.

Out Of Class Work

Should be spending about 8 hours per week outside of timetabled sessions working on this module. This includes working through the worksheets.

Assessments

There are three types of assessment used throughout the year

- 5x 30min programming tests (held in class, weighted 5% each)

- 2x 60min Computer based multiple choice tests (weighted 15 % each, one in January and one in May/June)
- 2x large programming assignments (weighted 20% and 25% respectively)

The programming tests will be based off of the previous weeks worksheets. There will be a practice test in week 3 (so we can understand how they work)
Each of the programming assignments will have a few weeks in which they can be worked on.

Resources

To write and execute Python programs, the recommended IDE is Pyzo. Other IDEs can be used however no support for configuration will be provided.
We will be using Python 3.x NOT Python 2.x.
The recommended book is called 'Python programming: an Introduction to Computer Science 3rd Edition'. There are a number of copies available in the library. Its ISBN number is '9781590282755'.

S.2. Writing Simple Programs

26-09-2022 15:00 Nadim RB LT1

This lecture introduces the basic steps involved in programming and provides some additional information about each stage.

Stages of Algorithm Design

When presented a problem to solve programmatically, the first stage to doing so is to understand the problem and to ensure that this understanding is correct. To aid this, it can be useful to work out how the user interacts with the system, through listing the user inputs and outputs to screen. At this stage, it can also be beneficial to make a note of some inputs and their expected outputs as this can be used to test the program at the end of development.
The next stage is to design an algorithm that accomplishes the task.

> **Algorithm**
>
> A detailed sequence of actions which acomplish a task. Cna be written in plain English or any other language.

The next stage is to implement the algorithm. This is where the plain English algorithm is converted into programming statements which can be executed by the machine.
The final stage is to test the program. This can be done ith the data noted down in stage one.

Key Program Concepts

In programming, there are a number of key concepts. These will be illustrated using examples written in Python 3.

Statements

Every line of a program is called a command or statement. These are executed (carried out) one after the other (there are ways in which the flow of the program can be altered, but this will be covered at a later date). Program execution ends after the last statement is executed.

Variables

A variable is a name for a part of the computer memory where a value is stored. The variables have names in the programs.
Statements in the program may create a new variable, use the value of a variable or change the value of a variable.

Assignment Statements

Assignment statements are used to assign a value to a variable. The syntax is as follows:

- The variable appears on the left hand size of the `=`
- The right hand side of is an expression, which has a value

LANGUAGE: Python3

```
variableName = expressionWhichHasAValue
```

Assignment statements are executed in two steps. First they evaluate the expression on the right hand side then second, assign the value to the variable on the left hand side.
If the variable on the lft hand side doesn't already exist, then it is created. If the variable exists already, its old value is replaced.

Numeric and String Values

Numeric values are numbers. They do not need any demarcation. For example, `2.2` is a numeric value.
String values are strings of characters. These can be any character. Strings need to be encased in single quotes or double quotes. Both are valid, however they can't be mixed. Lines 1 and 2 in the following example are valid, however line 3 is not.

LANGUAGE: Python3

```
1 validStringOne = "I'm in double quotes, notice I can use single quotes where I like!"
2 validStringTwo = 'Im in single quotes, notice I cant use single quotes in the string.'
3 invalidString = "Im not valid'
```

Arithmetic Expressions

Standard arithmetic expressions can be formed using `+`, `-`, `*`, `/` and `()`. Expressions are evaluated to give a value, this is commonly stored in a variable or outputted directly to the user.

Built-In Functions

Python has a number of built-in functions. These are algorithms which are part of the Python language. They can be accessed by using its name. Sometimes they have parameters, sometimes they return a value and sometimes they do both. Common examples of built-in functions are shown below.

LANGUAGE: Python3

```
1 print("I display information to the user")
2 variable = input("I allow the user to enter text, then I store it in the variable")
```

Example Execution

See Week 1, lecture 01c slides on Moodle for a detailed look at how programs execute and how the variable contents change.

Example programs from Lecture

Program 01

This program introduces a count-controlled loop (for loop) and the print statement.

LANGUAGE: Python3

```
1 total = 0
2 for i in range(34):
3     #print("banana")
4     #print(i)
5     total = total + i
6
7 print("The total is: ", total)
```

This program should output the following

```
LANGUAGE: Unknown
The total is:  561
```

The two commented out lines (lines which begin with the `#`) symbol can be un-commented so that they run.

Program 02

This program introduces the concept of `input()`, `int()` and subroutines.

```
LANGUAGE: Python3
def simpleProgram():

    value = int(input("Please enter a whole number: "))

    for loopCount in range (value):
        print(loopCount)

##################

simpleProgram()
```

The program should output the following.

```
LANGUAGE: Unknown
Please enter a whole number: 12
0
1
2
3
4
5
6
7
8
9
10
11
```

The number `12` on line one is entered by the user.

S.3. Computing with Data and Numbers

03-10-22 1500 Nadim RB LT1

Data and Data Types

There is a lot of data which programs have to process. Different types of data are stored as different 'Data Types', this allows them to be processed differently; an example of this is numerical data. In programming, we commonly distinguish between two different types of numerical data: integers (whole numbers, eg 55, 77, 88, -5) and fractional number (or floating point numbers, eg 4.6, 7.00956, -9.89). Words and other multi-character statements can be written within strings and truth values are stored as booleans.
All data values belong to one single data type and in some contexts, we use Class rather than data type.

Python Data Types

Python has all of the common data types within it. Each of the data types have a specific keyword:

Type	Python Keyword	Example
Integer	`int`	`33`
Fractional	`float`	`2.3`
String	`str`	`"Spam"`
Boolean	`bool`	`True`

Operations on Data Types

Data types have operations associated with them, some of these are language specific functions however the majority are universal across most programming languages.
For example, `int` and `float` both have the operations `+`, `-`, `*` and `/`.
Numeric data types follow the operator precedence rules, as a human would with mathematical equations. They follow BIDMAS. Where two operators have equal precedence, the calculations are carried out from left to right.

Type Conversions

It is important to be able to convert between different data types. The following example code shows the different functions.

LANGUAGE: Python3

```python
# convert 5 (int) to float, equals 5.0
floatVariable = float(5)

# convert 4.5 to int, this truncates, so will equal 4
intVariable = int(4.8)

# convert 6.8 to a string, equals "6.8"
strVariable = str(6.8)
```

It can be useful to find out what data type a variable or value is. To do this, use the `type()` function as seen below.

LANGUAGE: Python3

```python
print(type(44)) # outputs <class 'int'>
print(type("Banana")) # outputs <class 'str'>
```

```
print(type(4.67)) # outputs <class 'float'>
```

User Input

The `input()` function, returns a string. This can be really useful if we want to do something with a string. However, if we want to do something with a number, this is less useful. We can use the `float()` or `int()` functions to convert into floats or integers respectively. Examples of this can be seen below.
There is another function which can be used. The `eval()` function returns either a float or integer depending on the value passed into it. It can be useful in situations where the value entered by the user could be either floating point or integer.

LANGUAGE: Python3

```
# convert to float
floatInput = float(input("Enter a float here: "))

# convert to an integer
intInput = int(input("Enter an integer here: "))
```

Arithmetic Operations

Where an arithmetic operation involves both a float and integer, the integer is automatically converted to a float then the operation is carried out. For example, in the operation `7+1.5`, the `7` would be converted to `7.0`. Therefore, the calculation would then be `7.0 + 1.5 = 8.5`.

Division

The `/` operator always performs floating point division, hence `11 / 4 = 2.75`.
The `//` operator performs integer division, where it is given two integers as inputs, the result will be a truncated integer; as seen in the following example `11 // 4 = 2`
The `%` operator gives the remainder of an integer division, hence `11 % 4 = 3`.

Issues with Floating Point Arithmetic

Floating point numbers are represented within the computer using a fixed number of space (64 bits), this means that there is a limit to the range and accuracy of the number which is able to be stored. There are some numbers, 0.1 for example, which are unable to be represented within this size limit in binary, this can lead to issues with the value of a float number after performing mathematical operations on it.
This problem is true of all programming languages that use floating point numbers.

Python's Numeric Functions

There are a number of useful built-in functions in Python which help with maths.
The `round()` function takes a float as a parameter and returns the rounded value to the nearest int. It takes a second optional parameter which allows you to specify the number of digits after the decimal point to round to, as seen below.

LANGUAGE: Python3

```
intRound = round(5.6) # equals 6

floatRound = round(6.3345742, 3) # equals 6.335
```

The `abs()` function returns the absolute value of a number which is passed in as a parameter.
The `pow()` function takes two parameters, the first being the number and the second being the power of it we want to calculate, as seen below

LANGUAGE: Python3

```
powerTwo = pow(2, 3) # equals 8
powerThree = pow(3, 2) # equals 9
```

This function is the same as the `**` operator.

Math Module

Sometimes things we want to do mathematical things in Python which the base language can't do. To be able to do this, we import a library. This is some pre-written code which we can use in our programs.
To be able to use the math library, we first have to import it

LANGUAGE: Python3

```
import math
# or alternatively, if the line above doesn't work, use line below
from math import *
```

The math module provides a number of useful things including some constants (eg, `math.pi`) and mathematical functions (eg, `math.sqrt()`).