

ECDL COMPUTING

Syllabus 1.0

Teacher's Handbook



Provided by:
ECDL Malta

Copyright ECDL Foundation 2015 - 2019. All rights reserved. Reproducing, repurposing, or distributing this courseware without the permission of ECDL Foundation is prohibited.

ECDL Foundation, ECDL Europe, ECDL and related logos are registered business names and/or trademarks of ECDL Foundation.

This courseware may be used to assist candidates to prepare for the ECDL Foundation Certification Programme as titled on the courseware. ECDL Foundation does not warrant that the use of this courseware publication will ensure passing of the tests for that ECDL Foundation Certification Programme.

The material contained in this courseware does not guarantee that candidates will pass the test for the ECDL Foundation Certification Programme. Any and all assessment items and / or performance-based exercises contained in this courseware relate solely to this publication and do not constitute or imply certification by ECDL Foundation in respect of the ECDL Foundation Certification Programme or any other ECDL Foundation test. This material does not constitute certification and does not lead to certification through any other process than official ECDL Foundation certification testing.

Candidates using this courseware must be registered with the National Operator before undertaking a test for an ECDL Foundation Certification Programme. Without a valid registration, the test(s) cannot be undertaken and no certificate, nor any other form of recognition, can be given to a candidate. Registration should be undertaken at an Approved Test Centre.

Python is a registered trademark of the Python Software Foundation. Python and its standard libraries are distributed under the Python License. Details are correct as of December 2016. Online tools and resources are subject to frequent update and change.

ECDL Computing

With the increased use of computers in all areas of life, there is a growing interest in learning about the fundamentals of computing, including the ability to use computational thinking and coding to create computer programs.

The ECDL Computing module sets out the skills and competences relating to computational thinking and coding and will guide you through the process of problem solving and creating simple computer programs. Based on the ECDL Computing syllabus, this module will help you understand how to use computational thinking techniques to identify, analyse and solve problems, as well as how to design, write and test simple computer programs using well structured, efficient and accurate code.

On completion of this module you will be able to:

- Understand key concepts relating to computing and the typical activities involved in creating a program.
- Understand and use computational thinking techniques like problem decomposition, pattern recognition, abstraction and algorithms to analyse a problem and develop solutions.
- Write, test and modify algorithms for a program using flowcharts and pseudocode.
- Understand key principles and terms associated with coding and the importance of well-structured and documented code.
- Understand and use programming constructs like variables, data types, and logic in a program.
- Improve efficiency and functionality by using iteration, conditional statements, procedures and functions, as well as events and commands in a program.
- Test and debug a program and ensure it meets requirements before release.

What are the benefits of this module?

ECDL Computing has been developed with input from computer users, subject matter experts, and practising computing professionals from all over the world to ensure the relevance and range of module content. It is useful for anyone interested in developing generic problem solving skills and it also provides fundamental concepts and skills needed by anyone interested in developing specialised IT skills.

Once you have developed the skills and knowledge set out in this book, you will be in a position to become certified in an international standard in this area – ECDL Computing.

For details of the specific areas of the ECDL Computing syllabus covered in each section of this book, refer to the ECDL Computing syllabus map at the end of the learning materials book.

ECDL COMPUTING

OVERVIEW OF ECDL COMPUTING RESOURCES	1
GETTING STARTED	3
What is Computational Thinking?.....	3
What is Python?.....	4
HARDWARE AND SOFTWARE REQUIREMENTS	6
Hardware	6
Software	6
Installing Python	7
Installing the Pygame Python Library.....	11
Installing Boilerplate Code	12
LESSON 1 – THINKING LIKE A PROGRAMMER	13
1.1 Lesson Overview	14
1.2 Additional Resources	14
1.3 Exercises	15
1.4 Answers to Review Questions.....	19
LESSON 2 – SOFTWARE DEVELOPMENT	22
2.1 Lesson Overview	23
2.2 Additional Resources	23
2.3 Exercises	24
2.4 Answers to Review Questions.....	25
LESSON 3 – ALGORITHMS	26
3.1 Lesson Overview	27
3.2 Additional Resources	27
3.3 Exercises	27
3.4 Answers to Review Questions.....	29
LESSON 4 – GETTING STARTED	31
4.1 Lesson Overview	32
4.2 Additional Resources	32
4.3 Exercises	32
4.4 Answers to Review Questions.....	34
LESSON 5 – PERFORMING CALCULATIONS	35

5.1 Lesson Overview	36
5.2 Additional Resources	36
5.3 Exercises	36
5.4 Answers to Review Questions.....	37
LESSON 6 – DATA TYPES AND VARIABLES.....	38
6.1 Lesson Overview	39
6.2 Additional Resources	39
6.3 Exercises	39
6.4 Answers to Review Questions.....	40
LESSON 7 – TRUE OR FALSE	42
7.1 Lesson Overview	43
7.2 Additional Resources	43
7.3 Exercises	43
7.4 Answers to Review Questions.....	45
LESSON 8 – AGGREGATE DATA TYPES.....	46
8.1 Lesson Overview	47
8.2 Additional Resources	47
8.3 Exercises	47
8.4 Answers to Review Questions.....	48
LESSON 9 – ENHANCE YOUR CODE	49
9.1 Lesson Overview	50
9.2 Additional Resources	50
9.3 Exercises	50
9.4 Answers to Review Questions.....	51
LESSON 10 – CONDITIONAL STATEMENTS.....	52
10.1 Lesson Overview	53
10.2 Additional Resources	53
10.3 Exercises	53
10.4 Answers to Review Questions.....	54
LESSON 11 – PROCEDURES AND FUNCTIONS.....	55
11.1 Lesson Overview	56
11.2 Additional Resources	56
11.3 Exercises	56

11.4 Answers to Review Questions.....	60
LESSON 12 – LOOPS.....	61
12.1 Lesson Overview	62
12.2 Additional Resources	62
12.3 Exercises	62
12.4 Answers to Review Questions.....	64
LESSON 13 – LIBRARIES.....	65
13.1 Lesson Overview	66
13.2 Additional Resources	66
13.3 Exercises	67
13.4 Answers to Review Questions.....	67
LESSON 14 – RECURSION.....	68
14.1 Lesson Overview	69
14.2 Additional Resources	69
14.3 Exercises	70
14.4 Answers to Review Questions.....	71
LESSON 15 – TESTING AND MODIFICATION	72
15.1 Lesson Overview	73
15.2 Additional Resources	73
15.3 Exercises	73
15.4 Answers to Review Questions.....	74
TRAINING PLAN.....	75

OVERVIEW OF ECDL COMPUTING RESOURCES

To support teachers and learners in the implementation of the ECDL Computing syllabus a range of teaching and learning resources have been created. This implementation of ECDL Computing has been designed to use the Python programming language and these resources – a teacher handbook, learning materials and code files – are based on Python.

Teacher Handbook

The teacher handbook is designed to support teachers, in conjunction with the learning materials, in delivering the ECDL Computing syllabus. It broadly covers the following:

- Resources that give an introduction to computational thinking and the Python programming language. These are intended for teachers new to the subject who are looking to build their background knowledge in the area.
- The software and hardware requirements for delivering the module and the software installation and set-up instructions.
- A brief overview of the content covered at lesson level, along with additional resources for classroom differentiation and suggested exercises to support student learning. It is intended that teachers will review the resources and exercises and determine what is most appropriate for their students and their teaching, learning and assessment strategy.
- A suggested training plan that teachers can modify to suit their students and their teaching, learning and assessment strategy.

Learning Materials

The learning materials are aimed at learners and cover the skills and competences outlined in the ECDL Computing syllabus.

The learning materials are structured in lessons with each lesson consisting of learning objectives, learning content and review questions.

The learning content contains examples of code shown in screenshots as well as exercises that give learners the opportunity to produce small programs.

The learning content broadly follows the order of the ECDL Computing syllabus.

- Lessons 1 to 3 are theoretical and cover the concepts of computational thinking and programming.

- Lessons 4 to 15 introduce practical programming skills as well as developing problem solving to test and modify programmes. These lessons include coding exercises throughout.

On completion of the learning materials, learners should be prepared to complete the ECDL Computing certification.

Code Files

The learning materials and the teacher handbook contain exercises where learners are asked to create small programs. The solutions to the exercises are provided in the form of code files, which are organised by lesson in the following zip files.

- The **TeacherCodeFiles.zip** file contains code files relating to additional exercises outlined in the teacher handbook.
- The **StudentCodeFiles.zip** contains code files relating to some exercises in the learning materials. It also contains some pre-defined code, referred to as 'boilerplate code' that is needed to complete some exercises in the learning materials.

GETTING STARTED

Some background reading on the topic of computational thinking and a basic familiarisation with Python may be beneficial before examining each lesson in detail. The following information provides a good starting point for building knowledge in the area of computational thinking and Python. Review the information to determine what is relevant depending on your existing knowledge and experience. If you are new to the topic it is also highly recommended that you complete the student learning materials in full to gain a solid foundation in the topic.

WHAT IS COMPUTATIONAL THINKING?

The following resources provide an introduction to the concept of computational thinking.

Resource:	Jeanette Wing's Viewpoint Article
URL:	http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf
Learning Objective:	Understand the term Computational Thinking
Suggested Use:	Background information for teachers
Description	<p>The term 'computational thinking' was popularised by Jeannette Wing in her widely cited viewpoint article of March 2006. This article has many statements about what computational thinking involves, and no single clear statement about exactly what it is.</p> <p>Jeannette Wing suggests looking at solving of everyday problems through the lens of computer science. This paper argues that computational thinking is broader than just being for programming computers.</p>

Resource:	CAS Computational Thinking - A Guide for Teachers
URL:	https://www.computingschool.org.uk/ You will need to register at the site first, then locate: https://community.computingschool.org.uk/resources/2324
Learning Objective:	Understand the term Computational Thinking
Suggested Use:	Comprehensive guide for teachers. Links to collections of exercises.
Description	This is an excellent guide to the key concepts of computational thinking and contains links to online resources for teaching computational thinking.

Resource:	Research based definition of Computational Thinking
URL:	http://eprints.soton.ac.uk/372410/1/372410UnderstdC.T.pdf
Learning Objective:	Understand the term Computational Thinking
Suggested Use:	Background information for teachers/educational researchers.
Description	This paper contains a rigorous peer reviewed definition of what computational thinking has come to mean: “Computational thinking is an activity associated with problem solving, often resulting in an artefact or product. Computational thinking is a cognitive process resulting in an automation that is developed by the use of abstraction, decomposition, algorithmic design, evaluation, and generalisation.” ¹

WHAT IS PYTHON?

Python is a widely-used programming language. As well as the Python programming language, there is a Python interpreter, which is a software program that reads the Python code and performs its instructions. The Python interpreter, also referred to as the Python Shell, is free to use and the programming language is easy to read, making it suitable for first-time programmers.

Python is notable for the large number of ‘libraries’ of working code that Python programs can reuse. Python programs can be edited and the results tested, quickly and easily. It is regarded as a productive language for professional programmers.

Python is an actively used and evolving computer language. There are two major versions currently in use: Python 2 and Python 3. There were significant changes in version 3 and ECDL Computing learning materials are based on the newer version, **Python 3**.

The Python programming language is promoted and protected by the Python Software Foundation. You can download the relevant versions of the Python interpreter, as well as find a range of resources including tutorials, learning resources and help on their website at www.python.org. Instructions on installing Python are included in the next section.

The following provide a good introduction to the Python programming language.

Resource:	Informal overview of the basics of Python
------------------	---

¹ Selby, Cynthia and Woollard, John (2014) [Refining an understanding of computational thinking](#)

URL:	https://realpython.com/learn/python-first-steps/
Learning Objective:	Programming in Python.
Suggested Use:	Background information for teachers.
Description	This article is a very brief introduction to Python. Most of the syllabus items are lightly touched upon.

Resource:	Comprehensive Python Manual
URL:	https://docs.Python.org/3.5/tutorial/index.html
Learning Objective:	Programming in Python.
Suggested Use:	Background information and reference guide for teachers.
Description	This is a complete tutorial on Python 3. (Please note that lesson 4 onwards proceeds rapidly beyond the learning objectives of ECDL Computing.)

HARDWARE AND SOFTWARE REQUIREMENTS

In order to use the learning materials students will need access to the hardware and software outlined below.

HARDWARE

In lessons 1 to 3 in the learning materials the activities are mainly 'unplugged activities' related to computational thinking. These activities can and should be carried out away from the computer.

In lessons 4 to 15 in the learning materials programming is introduced and these lessons require the use of a computer with Python installed. One computer per student is recommended for these lessons but students can work in pairs or teams depending on available hardware. The materials are based on the use of desktop computers with Windows operating system installed.

SOFTWARE

In order to follow the learning materials some software must be installed on students' computers before they begin.

1. **Python** - The Python interpreter software - the Python shell - is free to download from <http://python.org/> and is available for Windows, OS X and Linux.
2. **Pygame Python Library** - ECDL Computing learning materials use a free Python library called Pygame. This is a library of modules designed for writing games in the Python language. It is needed to draw graphics and to react to the computer mouse from within a user's program. Pygame is used in the learning materials to illustrate some concepts but it isn't covered in the ECDL Computing syllabus. To find out more about Pygame and its possibilities go to <http://www.pygame.org/wiki/GettingStarted>
3. **Boilerplate code** - Students will need to access some 'boilerplate' code to use Pygame effectively. Boilerplate code refers to sections of pre-defined code that need to be included in multiple places with little or no alteration. This can be found in Lesson 13 in the learning materials code folder **StudentCodeFiles**.

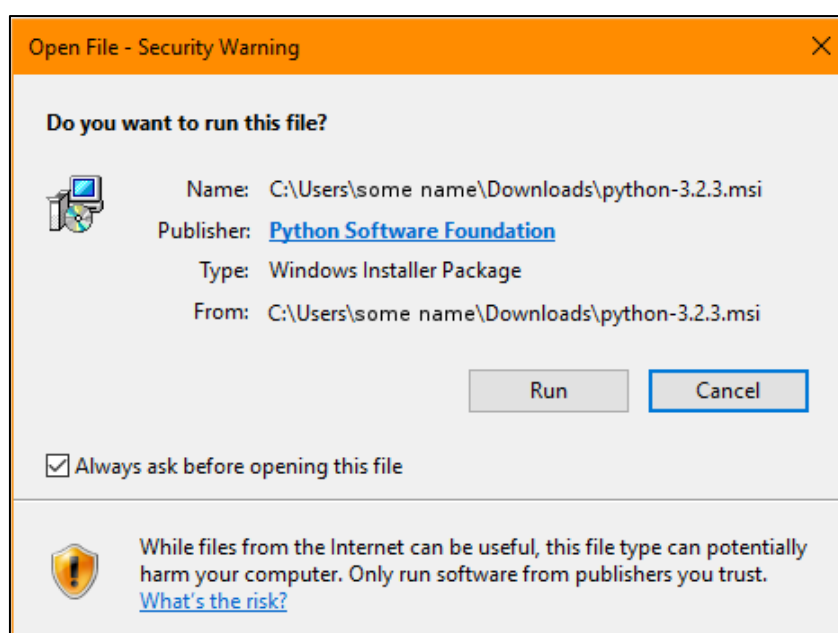
Installation instructions are outlined in the next section.

INSTALLING PYTHON

ECDL Computing learning materials are based on Python 3, or specifically Python 3.2.3. There are more recent versions than 3.2.3 available, but 3.2.3 should be used because it is stable and compatible with Pygame, also used in the learning materials.

To install the 32-bit version of Python 3.2.3 on a Windows desktop machine:

1. Go to <https://www.python.org/downloads/>
2. Locate the Python release version **Python 3.2.3** and click **Download**. (You may have to search by release version.)
3. In the Download section, select the **Windows x86 MSI Installer (3.2.3)** version of Python. (This is the 32-bit version of Python rather than the 64-bit version of Python. You must select the 32-bit version in order for it to run with Pygame, which is compiled for 32-bits only.)
4. When downloaded, double click the file to open it. The following dialog box appears:

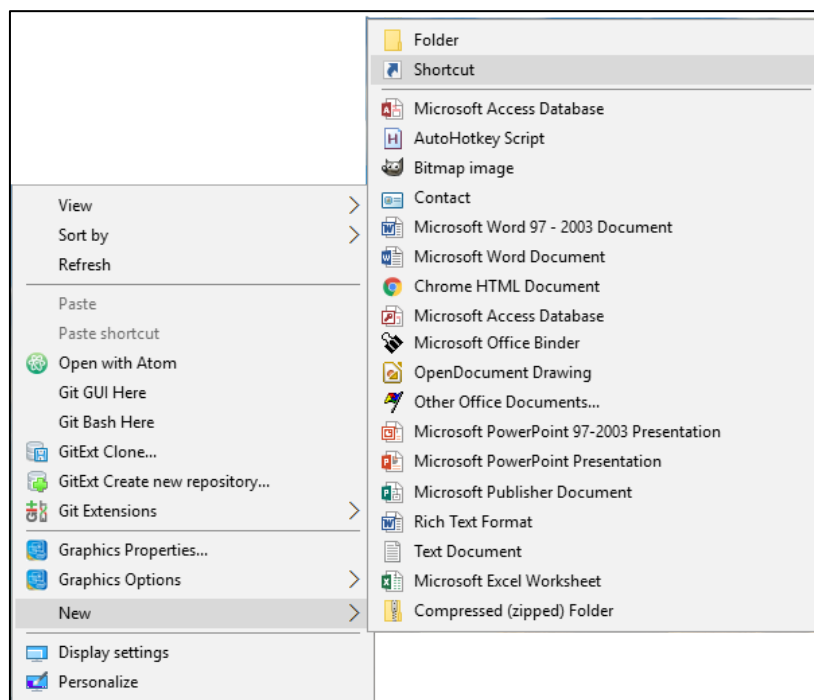


5. Click **Run**.
6. Choose the default options that follow. Take a note of the location where Python is being installed in the **Select Destination Directory** dialog box, for example, **C:\Python32**
7. Click **Finish** to exit the installer.

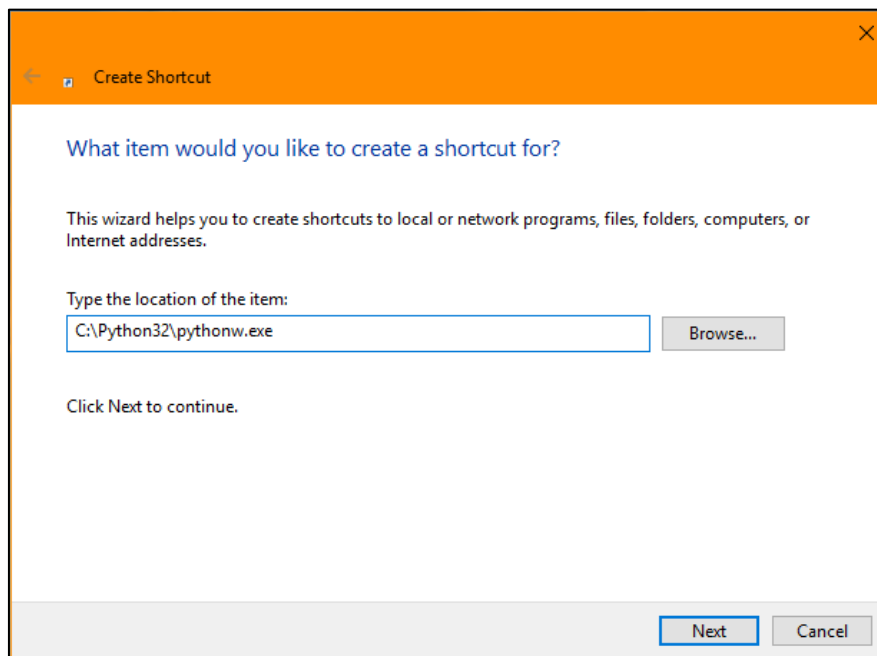
Create a shortcut icon to launch the Python Shell from the desktop:

1. Right-click anywhere on the desktop.

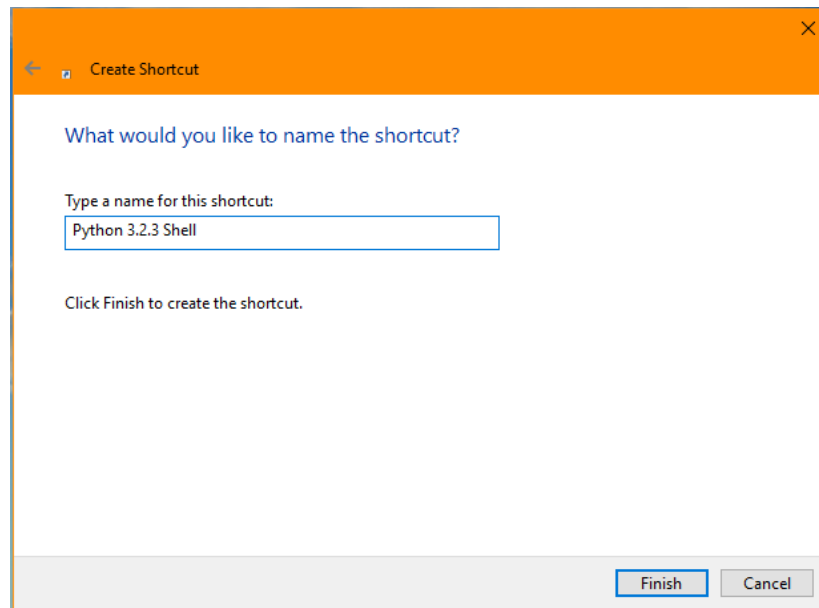
- From the menu that appears navigate to **New > Shortcut**.



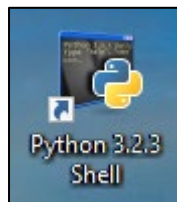
- Click **Shortcut**.
- Type the location or browse to the location of the pythonw executable file (**pythonw.exe**), for example, **C:\Python32\pythonw.exe**.



- Click **Next** to continue.
- To name the shortcut, type **Python 3.2.3 Shell**.



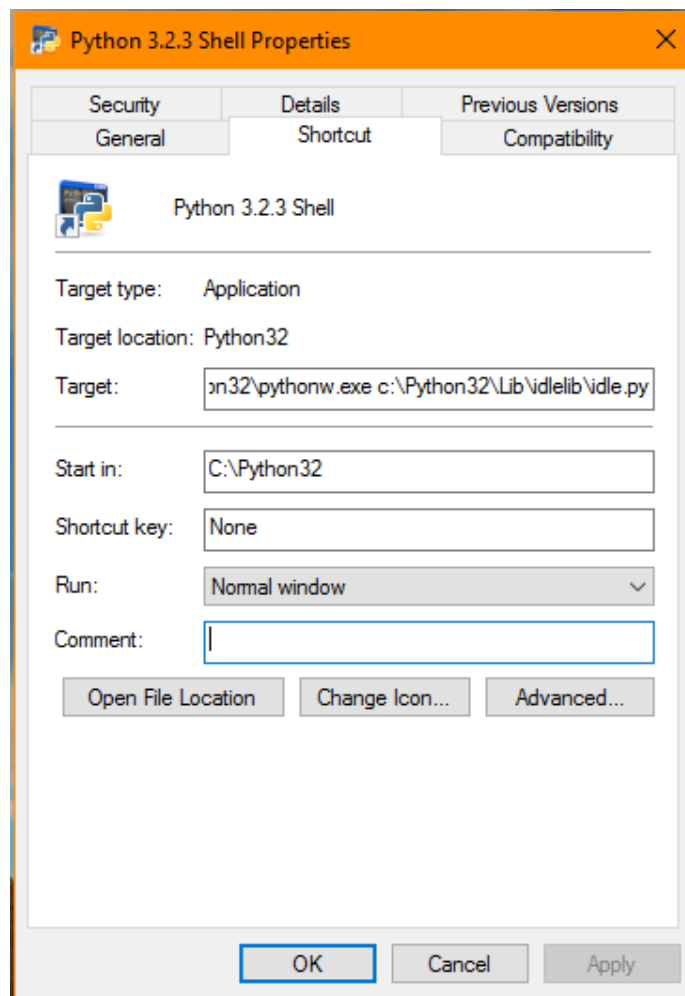
8. Click **Finish**.
9. Right click on the new **Python 3.2.3 Shell** shortcut icon on the desktop.



10. Select **Properties** from the menu.
11. In the dialog box, modify the target to

c:\Python32\pythonw.exe c:\Python32\Lib\idlelib\idle.py

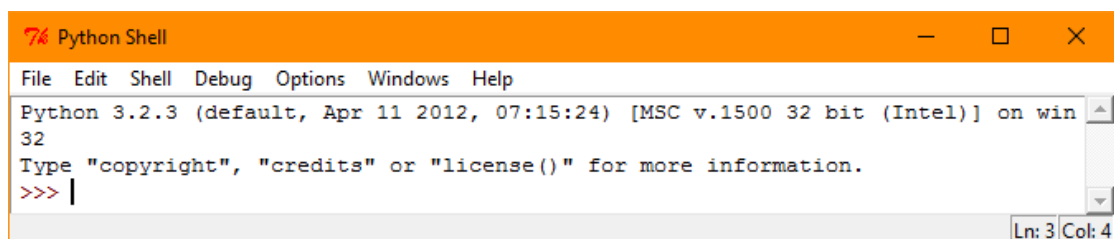
Note: If you installed Python somewhere other than c:\Python32 then change c:\Python32 in the example above to your location.



12. Modify the **Start in** directory to the directory where you would like your students to work.

13. Click **OK**.

14. To check that the Python Shell starts, double click the **Python 3.2.3 Shell** shortcut icon on the desktop to launch Python.



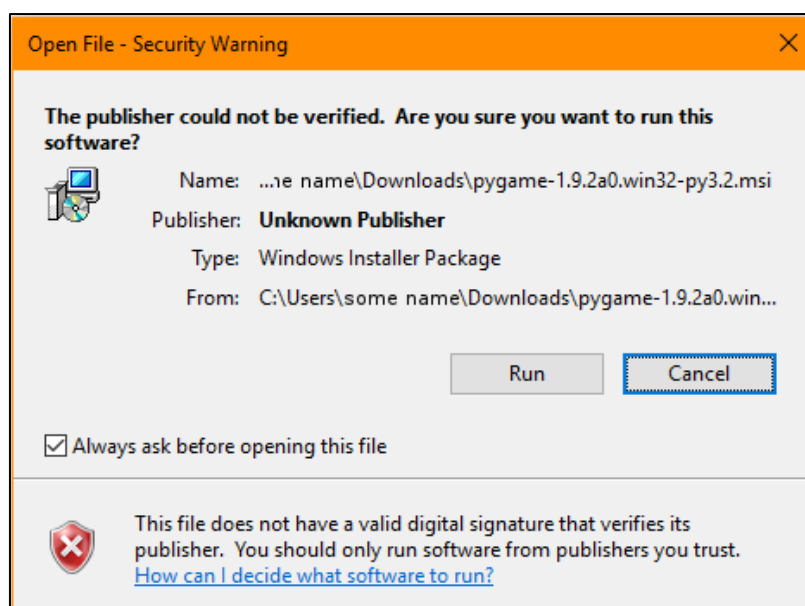
15. Click the **X** in top right hand corner to close the Python shell.

INSTALLING THE PYGAME PYTHON LIBRARY

ECDL Computing learning materials use **Pygame-1.9.2a0.win32-py3.2.msi**.

To Install Pygame:

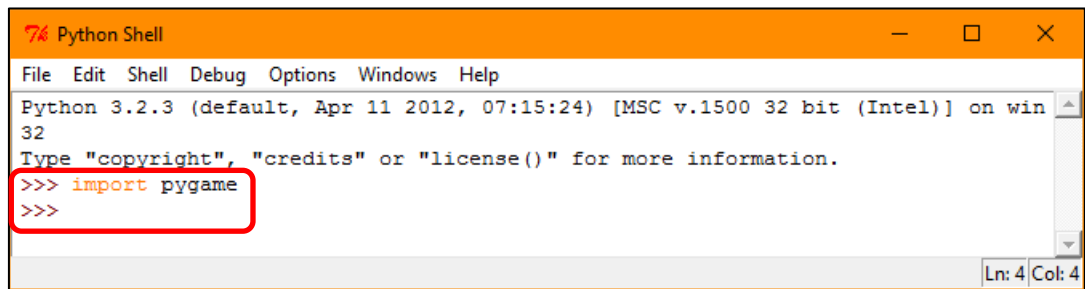
1. Go to <http://www.pygame.org/download.shtml>
2. Under **Windows** find the version of Pygame used in the ECDL learning materials: **pygame-1.9.2a0.win32-py3.2.msi**.
3. Click **pygame-1.9.2a0.win32-py3.2.msi**.
4. Double click the file to open it.



5. Click **Run**.
6. Choose the default options in the installation dialog boxes. **Note:** If given the option, do not install Numpy as it is not needed for these lessons.
7. Click **Finish** to exit the installer.
8. To check that Pygame has been installed successfully, open the Python shell by double clicking the **Python 3.2.3 Shell** shortcut icon on the desktop.



9. In the Python shell window type **import pygame:**



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>>
```

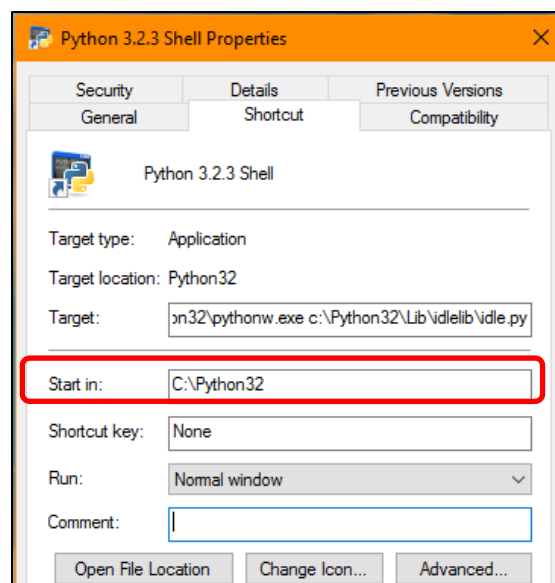
10. Pygame has been installed correctly if the prompt `>>>` appears with no error message, as shown above.

INSTALLING BOILERPLATE CODE

In the learning materials students will use Pygame in Lesson 13 – Libraries. However in order to use the Pygame library, a program needs to ask for the Pygame library, initialise it, define a few colours and do several other similar ‘housekeeping’ things. This requires additional ‘boilerplate code’. The same code will also handle Pygame events.

To install the ‘boilerplate code’:

1. Browse to **Lesson 13** in the learning materials code folder **StudentCodeFiles**.
2. Copy the file **BoilerPlate.py** into the directory where your students will work. (This is the directory you specified in the **Start in** field in the shortcut properties for the Python 3.2.3 Shell.). Students will use this file in lesson 13 – Libraries.



LESSON 1 – THINKING LIKE A PROGRAMMER

Learning objectives:

- Define the term computing
- Define the term computational thinking
- Outline the typical methods used in computational thinking: decomposition, pattern recognition, abstraction, algorithms
- Use problem decomposition to break down data, processes, or a complex problem into smaller parts
- Define the term program
- Understand how algorithms are used in computational thinking
- Identify patterns among small, decomposed problems
- Use abstraction to filter out unnecessary details when analysing a problem

1.1 LESSON OVERVIEW

This lesson is about computational thinking. It's about how a programmer analyses and breaks a problem down into smaller chunks. The intent is to show the strengths and limitations of computational thinking, beyond the confines of working as a programmer.

The lesson includes discussions on problem solving strategies and activities related to designing a washing machine and organising a music festival; both of which benefit from computational thinking. There is also an exercise for the class on an approach to sorting people in their class by order of height.

1.2 ADDITIONAL RESOURCES

Resource:	BBC Bitesize Guide to Computational Thinking
URL:	http://www.bbc.co.uk/education/guides/zp92mp3/revision
Learning Objective:	Explore the concepts of computational thinking
Suggested Use:	Student self-study

This bitesize guide limits itself to pattern recognition, abstraction, decomposition and algorithms in its definition of 'Computational Thinking'.

Resource:	Google's Exploring Computational Thinking
URL:	https://www.google.com/edu/resources/programs/exploring-computational-thinking/index.html#!ct-materials
Learning Objective:	Computational thinking. Introduction to algorithms.
Suggested Use:	Student activity

This material has a strong bias to maths and physics.

The 'Algorithmic Thinking' worksheet from that site provides a nice classroom exercise on formulating and following instructions. The challenge is to give instructions to someone else for putting Lego bricks together to recreate a design. See: https://docs.google.com/document/d/1Qouj-ZxcPvmYehvIvLGnNV0X_4E_9YNyjXEeCOmmBal/edit

When reflecting on the outcomes of this lesson, consider particularly the importance of clear, precise, unambiguous instructions in completing a task successfully.

Resource:	CSUnplugged
URL:	http://csunplugged.org/
Learning Objective:	Algorithms
Suggested Use:	Student activity

The CS Unplugged classroom exercises are generally fun and help to motivate and engage students.

The decoding a picture exercise from the CSUnplugged site helps to communicate the idea that computers encode information as 1’s and 0’s. See: <http://csunplugged.org/image-representation/>

Resource:	Sugar Sugar
URL:	http://www.mathplayground.com/logic_sugarsugar.html
Learning Objective:	Problem solving strategies - Decomposition
Suggested Use:	Student activity

Sugar Sugar is an online problem solving game.

Students work in pairs, moving forward through levels of increasing complexity. Much of the learning happens early on so it is not necessary to finish all levels.

Afterwards, discuss the exercise in class examining what was easy, what was hard and how the students approached the teamwork and problem solving. The learning is likely to differ to the 'analysis - design - implementation' pattern that is described in ECDL Computing Learning Materials lesson 1. Students may instead learn the value of starting over and the value of improvising when things go wrong. This fun activity can lead to a deeper understanding of how a well worked out plan to solve a problem can be created.

1.3 EXERCISES

Resource:	Extensions to Sorting Class by Heights
Learning Objective:	Algorithms, Decomposition.
Suggested Use:	Student activity

This exercise is a more challenging variant of the ‘sort the class in order of height’ challenge from lesson 1 of ECDL Computing Learning Materials.

The task is to arrange the class in a circle so that there is very little difference in height between consecutive people.

- i. The task will likely cause some discussion about how to solve this problem.
- ii. Agreeing on how decisions are made by the group is an unstated part of the task.
- iii. One way to solve the circle problem is to form two rows both separately sorted by height, and then join them together tall-to-tall, short-to-short to make a circle.
- iv. The best algorithms for solving the problem will have very little difference in height between adjacent people everywhere in the circle. The tallest person will have the next two tallest people on either side. The shortest person will have the next two shortest people on either side.

Resource:	What's Easy and What's Hard for a Computer?
Learning Objective:	Strengths and weaknesses of algorithmic design
Suggested Use:	Class discussion

This exercise is part of learning the scope of computing. Computers are very fast at arithmetic, and can do thousands of millions of sums a second. Computers, or more properly robots, can build cars, but they can't yet play football.

- i. Ask why some things are easy and some things more difficult for computers? Repetitive tasks are 'easy'. Tasks where it is hard to define exactly what the individual 'small steps' are, can be extremely hard or even impossible for computers to achieve.
- ii. Computers can predict the weather, but they do it by breaking the world map up into lots of small regions, and computing how the temperature and wind will change; based on physics. If they 'try' to look too far ahead the predictions or simulations become inaccurate.
- iii. Computers can, within limits, translate between languages. They can easily look words up in a computer dictionary but the translated text doesn't always make perfect sense.

The discussion should conclude that computers are best at doing things that can be broken down into well-defined simple steps.

Resource	Is a Mobile Phone a Computer?
Learning Objective:	Define the term computing
Suggested Use:	Class discussion

This exercise elicits thoughts from the class on what are the essential qualities of a computer.

- Is a mouse or physical keyboard necessary for a machine to be a computer? Is a screen necessary?
- It is likely the class will quickly decide that a mobile phone, even a very basic mobile phone, is a kind of computer. So, move on to washing machine, camera, microwave, toaster, light bulb. Are they computers?

Because there aren't sharp boundaries between computers and non-computers this exercise isn't really about the definition and right or wrong answers. It is more about fostering an awareness of what data different devices are working with, and how flexible their control systems are.

- Students might be surprised to learn that any device that plugs into a USB port of a computer or uses Wi-Fi has some kind of processor chip in it. For example, a keyboard has a chip in it that converts the key presses into 1's and 0's that can be sent over the USB link. That saves the keyboard from having one wire to the computer for each key.
- Students might be surprised to learn that the SD cards for storing pictures in a camera has a processor chip in it. That processor is responsible for working out where to store the picture, keeping track of which parts of the SD card memory are in use, which parts are good and which parts aren't.

Resource:	Design a Robot
Learning Objective:	Computational Thinking – Decomposition, Abstraction
Suggested Use:	Student activity

This exercise is primarily about 'decomposition'. To make a robo-chef, robo-footballer, robo-artist or household repair-bot - what component functions would the robot have to have to perform its task?

- i. Split the class into groups of four, and give each team an A2 whiteboard or A2 pad with markers.
- ii. If they are tasked with designing a robo-chef, each team is to decide what kind of chef they are going to design. For example, the robo-chef could be deployed solely to make sandwiches, or pancakes, it could be a specialist in Chinese meals, a salad expert or a barbeque pro.
- iii. Give teams five minutes to decide on:
 - Team Name
 - Spokesperson for the team
 - The Robo-Chef's speciality

- iv. Give the teams ten minutes to decide on the various functions their robot must do. Encourage the teams to:
 - Break the functions down into smaller functions, so ‘apple pie with ice cream’ becomes ‘slice apple pie’, ‘scoop ice cream’, ‘stick wafer into ice cream’.
 - Include ‘sensing’ tasks for things that could go wrong, as well as ‘action’ tasks, e.g. ‘detect burning’, ‘detect cake-is-stuck in cake tin’, ‘detect ice cream is melting’.
 - Find general reusable functions, such as ‘make pie case’, which can be used both for a quiche base and an apple pie base. Or peel(ingredient) which can be used to peel(potato) or peel(apple).
- v. Encourage teams to treat the robot as a real product. Is appearance important, or is it going to be working behind the scenes? What are the additional considerations e.g. cost, reliability, safety, how to instruct the robot, etc.

Quicker version: Teams share their results, giving a brief description of their robot. Remind the teams that they have been finding patterns, and breaking a problem down (decomposing) into smaller ones.

Longer version: Each team gets 20 minutes to prepare a pitch for funding to develop their robot. The aim is to present their robot as a potential commercial success. In the pitch, each team member articulates what they contributed to the project.

Resource:	Impossibly Big Problems
Learning Objective:	Decomposition
Suggested Use:	Class discussion

This is basically an exercise in problem solving; a problem, no matter how large, can be broken down (decomposed) into smaller problems. Computational thinking techniques like decomposition can be applied to problems that are not simply programming challenges.

- i. On the board write out these three big challenges:
 - World Peace
 - Global Warming
 - A Colony on Mars

- ii. Briefly describe what each problem is and get the class involved in talking about why these three different problems matter to them. Choose one of the challenges to work with.
- iii. Elicit ideas that could contribute to solving the problem. Complete solutions are not required, just some ideas that can help. It's a search for approaches that break the problem down by one step.
- iv. Ask particularly for ideas where a program or software could help. Don't accept ideas without a concrete foundation e.g. a program to invent a teleportation device to get to Mars, or a program to make everyone nice to each other so there is no war. A program to design a spacecraft, or a program to improve farming techniques are examples of acceptable ideas for how to start breaking the problem down.
- v. Break down a few suggested 'big ideas' into smaller parts, by asking questions e.g. what part of a spacecraft could a program help with? What aspect of farming could be improved by a computer program?
- vi. Here are some suggestions:
 - World Peace – computers that could translate voice as well as text could improve communication between peoples.
 - Global Warming – programs to analyse household electricity use making it easier to identify where to make efficiencies.
 - Colony on Mars - Can a computer model the inputs and outputs needed for a sustainable self-sufficient colony to be created?

1.4 ANSWERS TO REVIEW QUESTIONS

1. Computing is a set of activities that includes:
 - a. performing of calculations or processing of data.
2. Computational thinking is:
 - b. the process of analysing problems and challenges and identifying possible solutions to help solve them.
3. Which method is not a typical part of computational thinking?
 - b. Apprehension
4. Which of these is a good example of decomposition of a problem?
 - a. Splitting the task of designing a robot into designing the hand, designing the power source, deciding on and designing the sensors.

5. A program is:
 - b. An algorithm expressed in a form that is suitable for a computer
6. Which of these is least likely to be a way that algorithms are used in computational thinking?
 - b. An algorithm may lead to a computer program that employs intuition to derive better solutions.
7. Here are three recipes:

Chocolate cake:

Set oven to 180°C
Butter two 9" cake pans
Mix ingredients with a whisk for one minute
Transfer mix to cake pans
Bake for 30 mins
Allow to cool
Ice and decorate cake.

Gingerbread men:

Cream the ingredients together
Set oven to 190°C
Roll out the dough to about 1/8" thickness, and cut into gingerbread men shapes.
Bake until edges are firm, about 10 minutes.

Blueberry muffins:

Set oven to 185°C
Grease 18 muffin cups
Cream butter and sugar until fluffy
Add other ingredients
Spoon into muffin cups
Sprinkle with topping
Bake for 15 to 20 mins.

Which of these is a pattern common to all three recipes?

- d. Set oven to some temperature.
8. For designing a program for cooking for a robot chef to use, which of the following would be the most relevant abstraction?

- b. What quantities of each ingredient to use.

LESSON 2 – SOFTWARE DEVELOPMENT

Learning objectives:

- Understand the difference between a formal language and a natural language
- Define the term code; distinguish between source code and machine code
- Understand the terms program description, specification
- Recognise typical activities in the creation of a program: analysis, design, programming, testing, enhancement

2.1 LESSON OVERVIEW

This lesson starts with an explanation of the need for precision in computer languages to motivate the distinction between formal and natural languages. It then moves on to the distinction between the languages programmers program in and the 1's and 0's that computers use. It concludes with a description of the steps in software development.

2.2 ADDITIONAL RESOURCES

Resource:	6502 Machine Code
URL:	http://www.e-tradition.net/bytes/6502/6502_instruction_set.html
Learning Objective:	View actual machine code
Suggested Use:	Background information for teachers and students

This resource demonstrates the complexity of machine code as opposed to the illustrative example of machine code used in lesson 2 in the learning materials. The 6502 is a simple processor with approximately 151 different machine code instructions.

A related resource shows the transistors in a 6502 processor turning on and off as the processor obeys individual instructions. See:

<http://visual6502.org/JSSim/index.html>

This visual illustration of inside a processor is worth relating to the idea of decomposition of a problem. There are about 3,500 transistors in this circuit. Designing and verifying such a large circuit depends on being able to decompose it into simpler parts e.g. a part that adds two numbers, a part that fetches the next instruction etc.

Resource:	History of Compilers
URL:	https://en.wikipedia.org/wiki/History_of_compiler_construction
Learning Objective:	Difference between source code and machine code
Suggested Use:	Background information for teachers

The early translation programs were very simple and written in machine code by hand. These simple translation programs made it practical to write and translate larger programs. Incrementally new features could be added to the translation program, using the features of the version that preceded it.

2.3 EXERCISES

Resource:	Specification for a Game
Learning Objective:	Understand the term program specification
Suggested Use:	Student activity

In this exercise students in teams of four create a design specification for a game.

- i. The game can be based on the output of the Impossibly Big Problems exercise from lesson 1, such as a weather based computer game to raise awareness of global warming. Alternatively, the idea can be a completely new one.
- ii. Students will not at this stage be able to practice a full development cycle from analysis through to implementation and testing and program enhancement.
 - 1.1 The game SHALL be a platform jumper game.
 - 1.2 The game SHALL feature a melting snowman.
 - 2.1 There SHALL be 7 levels.
 - 2.2 The game SHALL feature sunglasses, umbrellas, penguins and polar bears.
- iii. Students will be able to practice decomposing the game ‘problem’ into a more detailed description of the game, and then into formal requirements, such as:
 - 1.1 The game SHALL be a platform jumper game.
 - 1.2 The game SHALL feature a melting snowman.
 - 2.1 There SHALL be 7 levels.
 - 2.2 The game SHALL feature sunglasses, umbrellas, penguins and polar bears.
- iv. Encourage students to number and group the requirements logically, so that related requirements are grouped together.
- v. <http://www.projectcartoon.com> - This URL has a variant on a popular cartoon showing the difference between a client’s expectations for software and the end product. The cartoon could be used to reflect on the necessity for well-defined software requirements.

2.4 ANSWERS TO REVIEW QUESTIONS

1. A formal language is:
 - b. A language with defined rules and unambiguous meanings.
2. English, Arabic and Chinese are:
 - c. Natural languages.
3. Machine code is.
 - c. The 1's and 0's that a computer obeys.
4. An algorithm written in the Python computer language is an example of:
 - a. Source code
5. A program specification is:
 - d. A description of what a program should do that is used during designing the program.
6. Match each activity in creating a program, a to e, to their purpose:
 - a) Testing, b) Design, c) Programming, d) Analysis, e) Enhancement

Improving an existing program.	e
Clearly defining the problem.	d
Checking whether the program works correctly.	a
Expressing the algorithm in the chosen computer language.	c
Working out an algorithmic approach to solving a problem.	b

LESSON 3 – ALGORITHMS

Learning objectives:

- Define the programming construct term sequence. Outline the purpose of sequencing when designing algorithms
- Recognise some methods for problem representation like: flowcharts, pseudocode
- Recognise flowchart symbols like: start/stop, process, decision, input/output, connector, arrow
- Outline the sequence of operations represented by a flowchart, pseudocode
- Write an accurate algorithm based on a description using a technique like: flowchart, pseudocode
- Fix errors in an algorithm like: missing program element, incorrect sequence, incorrect decision outcome

3.1 LESSON OVERVIEW

This lesson introduces the idea of a series of simple instructions followed step by step using flowcharts. There are three flowchart examples.

3.2 ADDITIONAL RESOURCES

Resource:	Everyday Machines
URL:	http://www.explainthatstuff.com
Learning Objective:	Learn what ‘computing’ occurs in everyday machines.
Suggested Use:	Background for the 'Flowchart for a machine' exercise.

These specific resources may be useful in the 'Flowchart for a Machine' exercise outlined in the next section:

- <http://www.explainthatstuff.com/digitalcameras.html>
- <http://www.explainthatstuff.com/washingmachine.html>
- <http://www.explainthatstuff.com/microwaveovens.html> (This is less useful than the others)

3.3 EXERCISES

Resource:	Flowchart for Everyday Algorithms
Learning Objective:	Practice with flowchart symbols Understand the decision symbol
Suggested Use:	Development of a student activity on algorithms

First attempts at everyday algorithms tend to represent linear sequences. Most recipes are a step by step sequence, with perhaps a ‘wait for oven to heat up’ being the furthest deviation from the sequential flow. Most instructions for putting furniture together are step by step.

Encourage the students to find ways that their everyday algorithms could contain an ‘if’ scenario, e.g.

- How does the getting ready for school algorithm change if there is a power cut, or there is no cereal for breakfast?
- How does a sandwich recipe change depending on what is available in the fridge?
- How do the furniture assembly instructions change if you’ve made a mistake in assembly?

Resource:	Inputs and Outputs
Learning Objective:	Understand inputs and outputs Understand scope of computing
Suggested Use:	Class discussion

Students identify inputs and outputs on various machines – coffee maker, oven, car, electric fan, gas boiler etc. Consider input and output sounds (e.g. as buttons are pressed, or the sound on completion) and lights (e.g. the busy/finished lights, the light when the fridge door opened).

Resource:	Flowchart for a Machine
Learning Objective:	Practice with flowchart symbols Understand inputs and outputs Understand scope of computing Know what an infinite loop is
Suggested Use:	Student activity

Students work on this exercise in pairs.

- i. Choose one of a washing machine, camera or microwave oven, and discuss what its inputs and outputs are. Relate the inputs and outputs to the input or output box in the flowcharts.
- ii. Discuss what the 'logic' is behind the operation of the device e.g.:
 - On a **camera**, the automatic flash operates if light levels are low. There is a small speaker that can make sounds, such as a 'click'. The photographer can delete an image they have just taken. A timer can be set to take a picture after a certain time. No more photos can be taken if the storage is full.
 - A **washing machine** has a door lock, a motor, valves that control water coming in, a heater, a temperature sensor, a pump, a water level sensor, a timer, a display panel, control knobs/buttons. Whilst hot water is an input to the washing machine drum, the heater (on or off) is an output of the washing machine controller. The heater plus temperature sensor ensure water is at the correct temperature. The valve, pump and water level sensor ensure water fills to the right level in the drum. The washing machine can make a sound when it has finished.
 - A **microwave oven** has a door sensor, a turntable, a light, a microwave generator, a timer. It may have a 'grill' electric element

too. It has a display and buttons. It can make a sound when it has finished.

- iii. Students create a flow chart for part of the operation of the device, using IF, e.g. IF water level high enough, close water inlet valve.
- iv. Students can then translate the flowchart into pseudocode.
- v. Class discussion of the resulting flow charts will likely show (a) some variations in the ways to achieve the same result and (b) that students don't know exactly what rules the machines follow.
- vi. Review the flowcharts:
 - Possibly some flow charts will have bugs?
 - Have the students included all the inputs and outputs that they identified?
 - Which flow chart has the longest sequence of actions without any IFs?
 - Do the flow charts have an exit or end?

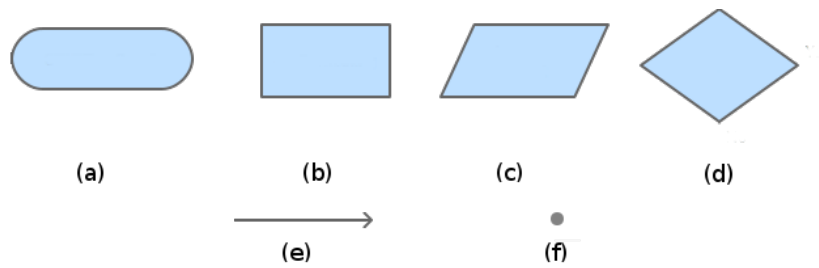
Camera - could pause 'forever' in an infinite loop, waiting for enough light for the exposure, if the camera is in the dark.

Washing machine - Could pause 'forever' attempting to fill if mains water is off, or 'emptying' if the drain is blocked. Or could beep forever, if no one is coming to empty it.

Microwave Oven - Could pause forever waiting for the user to press START.

3.4 ANSWERS TO REVIEW QUESTIONS

1. A sequence is:
 - b. A number of instructions that should be obeyed one after the other.
2. Which of these is not used to represent an algorithm?
 - d. Pseudoscience.
3. Match the following symbols to their names:



Arrow	e
Decision	d
Start or Stop	a
Process	b
Input or Output	c
Connector	f

4. In the following pseudocode:

```

Set oven to 180°C
Mix ingredients
Put cake in oven
Check whether cake is baked and while it isn't
Wait
Take cake out of oven
STOP
    
```

Which instruction is obeyed immediately after 'Mix Ingredients'?

- d. Put cake in oven

5. The following program is supposed to drain water from a washing machine. What error does it have?

```

Turn on drainage pump
Check water level
While there is no water left
    Wait
Turn off drainage pump.
STOP
    
```

- c. Incorrect decision outcome

LESSON 4 – GETTING STARTED

Learning objectives:

- Start and run a program
- Enter code
- Create and save a program
- Open and run a program

4.1 LESSON OVERVIEW

The purpose of this lesson is to allow students to familiarise themselves with the Python shell and gain early success in creating, running and saving a simple program.

Note: You must have Python installed on each student computer before beginning this lesson. Refer to **Installing Python** in this handbook for instructions.

4.2 ADDITIONAL RESOURCES

Resource:	Pygame Examples
URL:	http://www.pygame.org/tags/example
Learning Objective:	Practice opening and running Python programs.
Suggested Use:	Examples of what is possible with Pygame.

This URL provides additional examples using Pygame. The code in the examples themselves goes beyond what's covered in ECDL Computing learning materials however they can be used as practice for opening and running a Python program from the Python shell.

Typically these examples require that you download a compressed 'zip' file, and then extract Python files and images into a folder. The Python program can then be run by navigating to that folder from file->open on the Python shell.

An analog clock is one of the examples that work. It can be downloaded from here: <http://jestermon.weebly.com/pyclock2d.html>

It can be opened and run from the Python shell without modifying it first. Some of the other examples will require modification similar to the change described in the Aliens! exercise below.

4.3 EXERCISES

Resource:	Aliens!
Learning Objective:	Getting started with Python programs
Suggested Use:	Student activity

Open and run this 'Alien Invaders' example that comes with Pygame.

- i. Use the File->Open menu item, and navigate to:

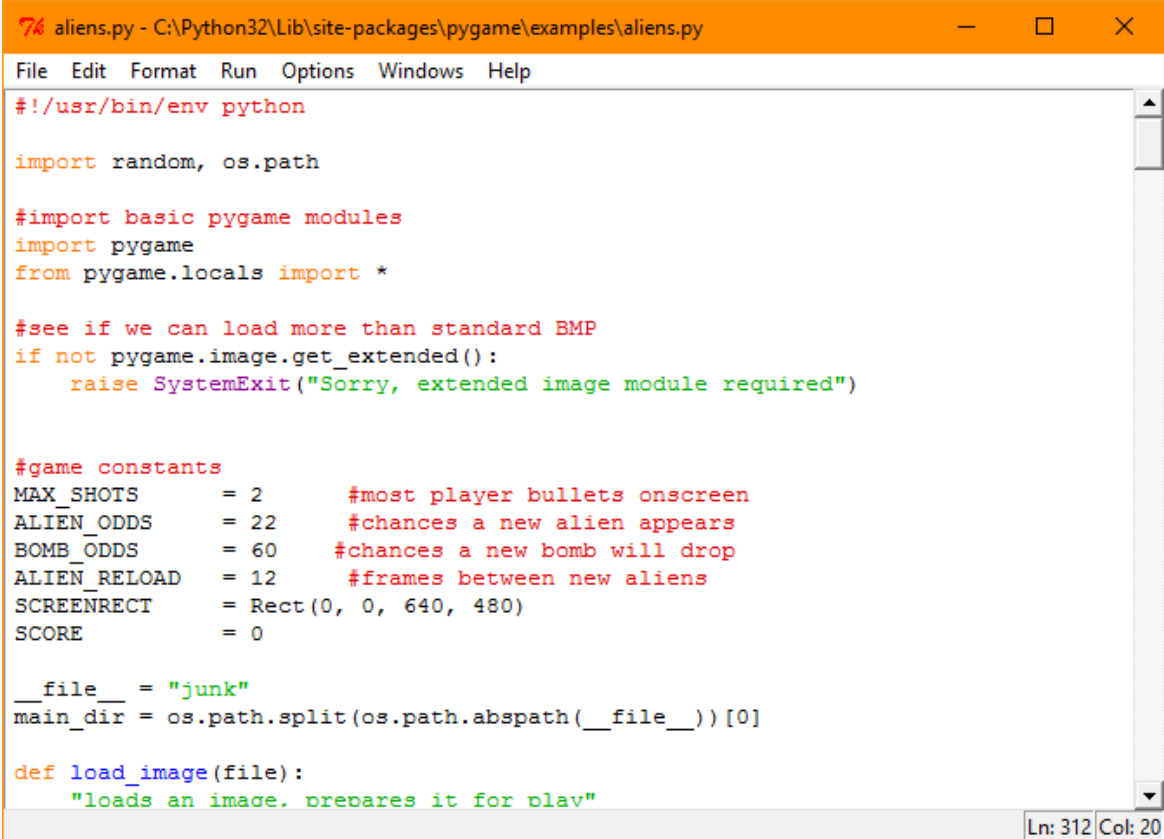
C:\Python32\Lib\site-packages\pygame\examples and choose aliens.py.

- ii. Open and edit the file to add this line:

```
__file__ = "junk"
```

As shown in the screenshot:

(Note: There are two `_` characters, not one, before and after the word 'file'.)



```
74 aliens.py - C:\Python32\Lib\site-packages\pygame\examples\aliens.py
File Edit Format Run Options Windows Help
#!/usr/bin/env python

import random, os.path

#import basic pygame modules
import pygame
from pygame.locals import *

#see if we can load more than standard BMP
if not pygame.image.get_extended():
    raise SystemExit("Sorry, extended image module required")

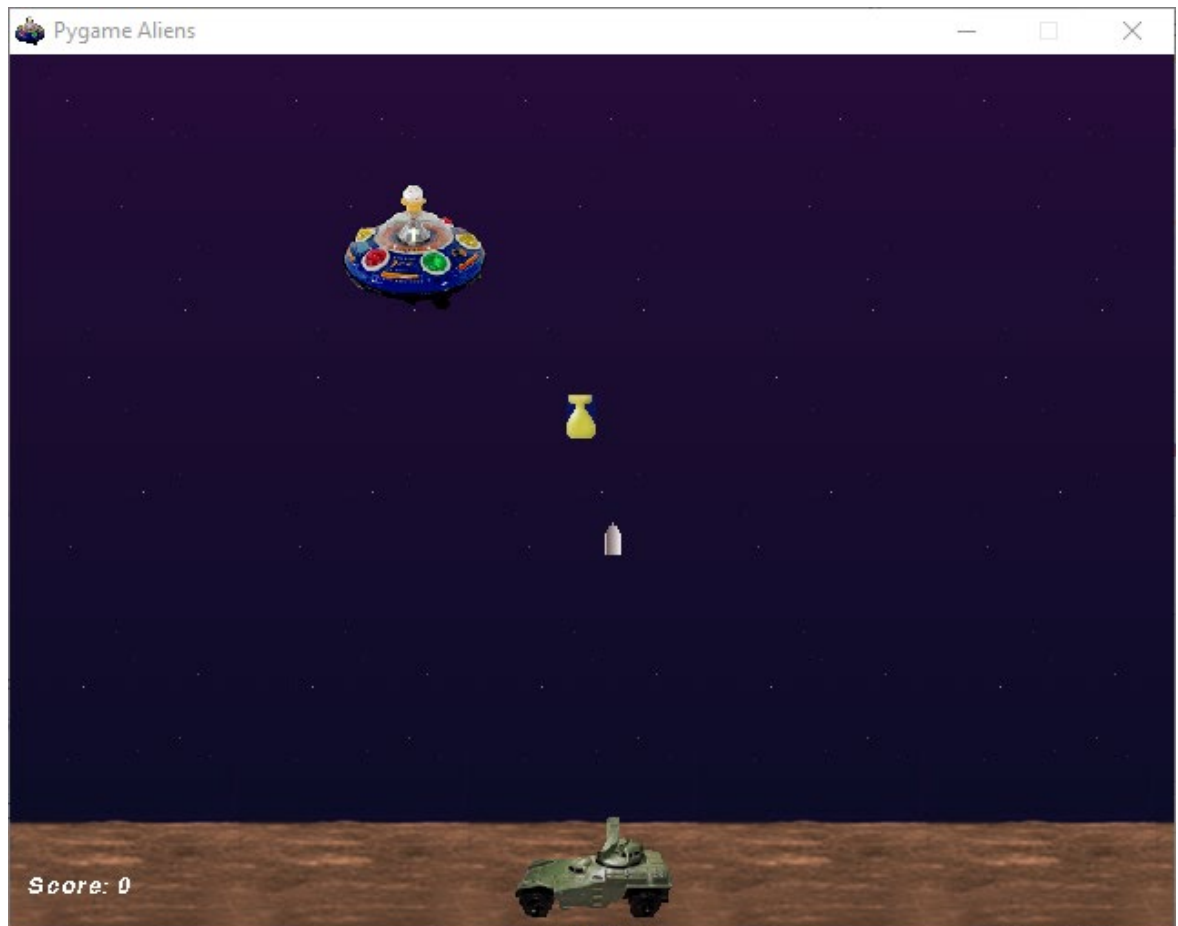
#game constants
MAX_SHOTS      = 2      #most player bullets onscreen
ALIEN_ODDS     = 22     #chances a new alien appears
BOMB_ODDS      = 60     #chances a new bomb will drop
ALIEN_RELOAD   = 12     #frames between new aliens
SCREENRECT     = Rect(0, 0, 640, 480)
SCORE          = 0

__file__ = "junk"
main_dir = os.path.split(os.path.abspath(__file__))[0]

def load_image(file):
    "loads an image, prepares it for play"
```

Ln: 312 Col: 20

- iii. Run the program using F5 to start this game:



Alternatively, you can make the exercise easier for students by modifying and saving the file `aliens.py` before the lesson.

4.4 ANSWERS TO REVIEW QUESTIONS

1. How do you run a program in Python?
- b. Pres F5.

LESSON 5 – PERFORMING CALCULATIONS

Learning objectives:

- Recognise and use arithmetic operators. +, -, * and /
- Know how parentheses affect the evaluation of mathematical expressions
- Understand and apply the precedence of operators in complex expressions
- Understand how to use parenthesis to structure complex expressions

5.1 LESSON OVERVIEW

This lesson uses Python as a calculator. To effectively use calculations in Python students must understand the rules around operator precedence and how Python evaluates expressions. Two data types are introduced: int and float.

5.2 ADDITIONAL RESOURCES

Resource:	Definitive Guide to Python Operator Precedence
URL:	https://docs.python.org/3/reference/expressions.html#operator-precedence
Learning Objective:	Extend knowledge of operator precedence Strengthen the concept of 'formal language'
Suggested Use:	Background knowledge for teachers

This is the full explanation of operator precedence in Python. It has more detail than is required to fulfil the syllabus objectives.

5.3 EXERCISES

Resource:	Four 4's
Topic:	Expressions
Learning Objective:	Using Python as a calculator
Suggested Use:	Student self-study

Find expressions using exactly four x 4's that make the numbers 1, 2, 3, 4, up to 9. Brackets are allowed. For example: 1 is $4/4*4/4$.

- i. If the students want to go beyond 9, they can use square roots too. Square roots are in a Python library (see lesson 13). Students can get the Python math library which contains square roots by typing:

```
import math
```

- ii. Once the math library is imported, square roots can be used in expressions. For example:

```
math.sqrt(4) * 4 * 4 * 4 is 128
```

- iii. Using square roots students can continue from 9 on to 22.

5.4 ANSWERS TO REVIEW QUESTIONS

1. How do you get Python to evaluate an expression?
 - c. Type it in after the >>> prompt.

2. Which of the following expressions is used to multiply two numbers in python?
 - d. $4 * 7$

3. Evaluate the expression $3*4*2+8$ using the Operator Precedence rule and select the appropriate answer below
 - c. 32

LESSON 6 – DATA TYPES AND VARIABLES

Learning objectives:

- Use different data types, character, string, integer, float, Boolean
- Define the programming construct term variable. Outline the purpose of a variable in a program
- Define and initialise a variable
- Assign a value to a variable
- Use data input from a user in a program
- Use data output to a screen in a program

6.1 LESSON OVERVIEW

This lesson examines four basic Python data types: integer, float, string and Boolean. The concept of a variable is introduced. The input () function allows students to further explore working with strings.

6.2 ADDITIONAL RESOURCES

Resource:	Information on variables
URL:	https://www.tutorialspoint.com/python/python_variable_types.htm
Learning Objective:	Deeper understanding of variables
Suggested Use:	Background information for teachers

The ECDL Computing syllabus requires a basic knowledge of Python variables.

The linked URL provides more detail on variables. For example:

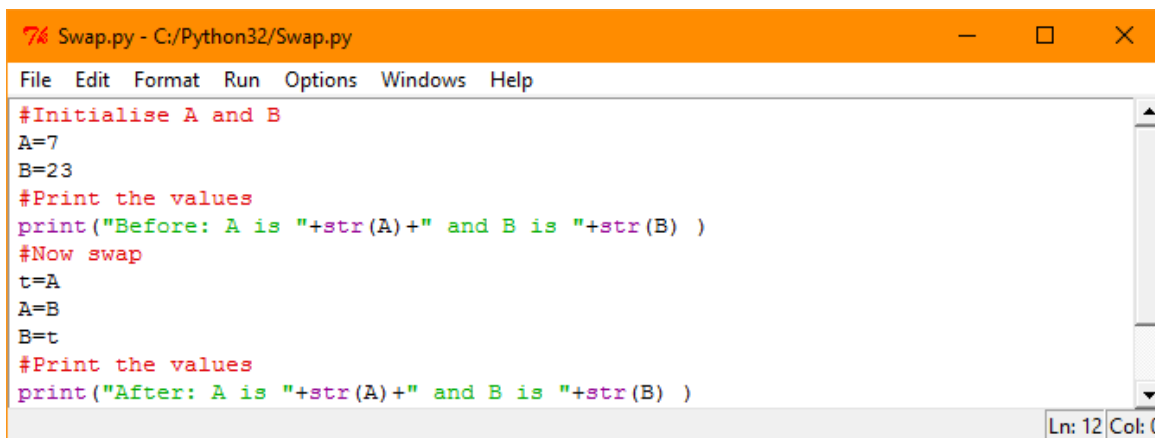
- Multiple assignment, where several variables are given a value at the same time.
- The dictionary data type.
- Removing a variable using 'del'.
- Complex numbers, i.e. the square root of minus one and related numbers.
- The slice operator for looking at part of a list or tuple.

6.3 EXERCISES

Resource:	Swapping Two Variables
Learning Objective:	Using variables
Suggested Use:	Student self-study
Solution File:	Swap.py located in the lesson 6 folder in the TeacherCodeFiles folder

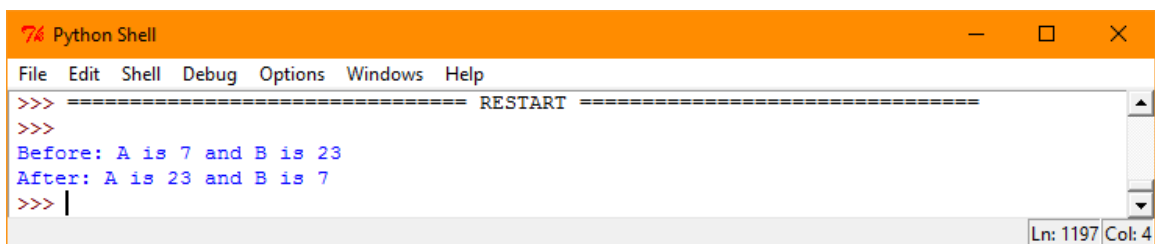
- i. Ask students to imagine that variable A has been initialised to some value and B to some other value.
- ii. Students must write a piece of code that will swap the values of A and B.
- iii. If necessary provide a hint: introduce an additional variable.

Here's a possible solution:



```
76 Swap.py - C:/Python32/Swap.py
File Edit Format Run Options Windows Help
#Initialise A and B
A=7
B=23
#Print the values
print("Before: A is "+str(A)+" and B is "+str(B) )
#Now swap
t=A
A=B
B=t
#Print the values
print("After: A is "+str(A)+" and B is "+str(B) )
Ln: 12 Col: 0
```

When run this gives:



```
76 Python Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
Before: A is 7 and B is 23
After: A is 23 and B is 7
>>> |
Ln: 1197 Col: 4
```

6.4 ANSWERS TO REVIEW QUESTIONS

- Which statement about variables is correct?
 - Sometimes a variable can hold a Boolean value.
- What is the purpose of a variable?
 - To remember some value that will be used again later in the program.
- Which of these Python statements initialises a variable?
 - `x=3.`
- In Python, if a variable named 'length' has the value 200, and the instruction 'length=400' is obeyed, what happens?
 - Python assigns 400 to length and length now has the value 400.
- In Python, the `input()` function:
 - Returns a string.
- In Python, the `output()` function
 - Is used to display a result
- A datatype which has only two values is called

d. Boolean

LESSON 7 – TRUE OR FALSE

Learning objectives:

- Use Boolean logic expressions in a program
- Recognise types of Boolean logic expressions to generate a true or false value like: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT
- Understand the precedence of operators and the order of evaluation in complex expressions

7.1 LESSON OVERVIEW

This lesson progresses from a first look at the Boolean data type in lesson 6 to examining Boolean logic in more detail. It looks at how Boolean expressions are built, comparison operators and Boolean operators. Booleans are combined with variables to produce a guessing game. Finally, parentheses and operator precedence are considered, producing more complex expressions.

7.2 ADDITIONAL RESOURCES

Resource:	Boolean operators in search
URL:	http://library.alliant.edu/screens/boolean.pdf
Learning Objective:	Boolean operators and, or, not
Suggested Use:	Background information for teachers and students

This document shows how AND, OR and NOT can be used in a library search system.

The logical operators define a set of results to present.

The following table may also be helpful:

Boolean Algebra	Sets
A and B	A intersect B
A or B	A union B
not A	Complement of A

7.3 EXERCISES

Resource:	Logic in Search Terms
Learning Objective:	Algorithms (from lesson 1) Formal and informal languages. (from lesson 2) Boolean logic (from lesson 4)
Suggested Use:	Student activity

Logic is useful in online Internet searches such as Google advanced search.

- i. Ask students to look at: https://www.google.com/advanced_search
- ii. Students familiarise themselves with the search entry boxes:
 - Which of the entry boxes is for AND of the search terms?
 - Which of the search boxes is for OR of the search terms?

- Which box is used for 'Not'?
- iii. Students attempt more advanced methods of searching to find examples of ambiguous English. For example, to look for ambiguous English and exclude definitions of the word ambiguous, and to exclude the “Kill la Kill Ambiguous” song use:
- ambiguous english -kill -definition
- iv. Challenge students to perform advanced searches, providing examples such as:
- Pumpkins, but not about Halloween
 - Something no other pair in the class found out about, about snowflakes
- v. Conduct a class discussion on how an Internet search engine attempts to get around the vagaries of natural language. Pay attention to scenarios where search results were not as expected. Consider the challenge in giving clear instructions to a computer. This relates back to the learning about formal languages in lesson 2 onwards.

Resource:	And and Or in Controlling a Washing Machine
Learning Objective:	Reinforce understanding of OR and AND Boolean operators
Suggested Use:	Student activity

Students list parts of a washing machine that can be on or off (outputs).

(A list prepared earlier from lesson 2 activities can be used here.)

- i. The list could include:
- Element to heat the water
 - Motor to turn the drum
 - Pump to pump water out.
 - Valve to allow water in
 - Lock to lock door.
- ii. Talk about the sensors - what the washing machine can measure (inputs):
- Level of water
 - Temperature of water
 - Time

- iii. Ask about things that could go wrong, examples include:
- Heating element ON but no water, then the heating element will burn out.
 - Opening the door when there is water in the drum
- iv. These potential problems lead on to describing the logic of the washing machine's operation. Logic statements may use AND and OR. For example:
- Lock the washing machine door if the drum is spinning **OR** there is water in the drum
 - Turn on the pump to pump out water if the machine is in the spin cycle **AND** there is water in the washing machine drum"

In practice a washing machine's operational logic is more complex e.g. the pump will turn on during rinses. However, AND is still a building block in the logic to control the washing machine pump.

- v. AND and OR may also be relevant in an algorithm to follow for the person who bought the washing machine.
- Contact manufacturer if (Door-seal leaky OR motor-does-not-work) AND (still under warranty)

7.4 ANSWERS TO REVIEW QUESTIONS

1. Which of the following is a python expression to add three numbers together and divide the result by 3:

c. $(4+5+6)/3$

2. The expression ' $3 < x$ and $x \leq 7$ ' is true if x is:

d. 4,5,6 or 7

3. In Python, which expression could give a different result to the other three?

c. $a < b$ or $b < c$

LESSON 8 – AGGREGATE DATA TYPES

Learning objectives:

- Understand aggregate data types
- Use aggregate data in a program like: list and tuple

8.1 LESSON OVERVIEW

The final two data types, list and tuple are introduced in this lesson. The Python ‘list’ aggregate data type fulfils the role of an ‘array’ in other programming languages. Rather than holding a single value, these data types hold multiple items, for example, a list of names. Some basic examples illustrate their use.

8.2 ADDITIONAL RESOURCES

Resource:	Lists v Tuples
URL:	http://stackoverflow.com/questions/626759/whats-the-difference-between-lists-and-tuples
Learning Objective:	Deeper understanding of lists and tuples
Suggested Use:	Background information for teachers and students

Used with care, this URL can also help students understand how to use online resources productively when learning to program.

StackOverflow is an excellent site for professional programmers with answers to many questions, including questions about Python.

In this example, the first answer on the page uses a lot of new terminology. The second and less popular answer has information more relevant to someone starting learning Python. Students will get a better understanding of the difference between lists and tuples from this.

8.3 EXERCISES

Resource:	Reversing a list
Learning Objective:	Working with lists
Suggested Use:	Student self-study
Solution File:	Reverse.py located in the lesson 8 folder in the TeacherCodeFiles folder

- i. Students write a program that makes a list with the colours of the rainbow, red, orange, yellow, green, blue, indigo, violet.
- ii. Can they print the list in reverse order? At this stage in their learning it is unlikely that students can achieve this.
- iii. Students use an Internet search engine to find a solution and try it out.

Here's one possible solution:

LESSON 9 – ENHANCE YOUR CODE

Learning objectives:

- Describe the characteristics of well-structured and documented code like: indentation, appropriate comments, descriptive naming
- Define the programming construct term comment. Outline the purpose of a comment in a program
- Use comments in a program

9.1 LESSON OVERVIEW

This lesson examines best practice in terms of structuring and documenting code with some Python examples for illustration purposes.

9.2 ADDITIONAL RESOURCES

Resource:	Python Style Guide
URL:	http://legacy.python.org/dev/peps/pep-0008/
Learning Objective:	Characteristics of well-structured and documented code
Suggested Use:	Reference guide for teachers

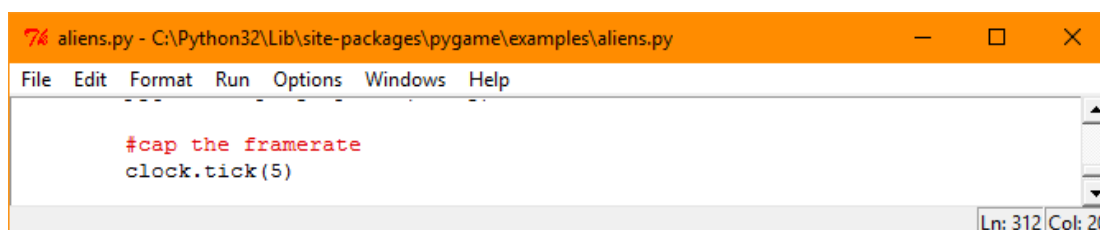
This document gives coding conventions for the Python code. It includes guidance on comments, blocks, indentation and variable names in more detail than is required by the ECDL Computing syllabus.

9.3 EXERCISES

Resource:	Reading code comments
Learning Objective:	Understanding the value of comments in code
Suggested Use:	Student activity

In film and television, the **'frame rate'** is how many frames are displayed per second. Television broadcasts are about 30 frames per second.

- i. Students use the aliens.py game from lesson 4, and find out how to slow it down to make it much easier to play.
- ii. Change the relevant code extract and play the game.
- iii. Discuss how the comments helped in figuring out what to do. The code students are looking for is shown below:



```

76 aliens.py - C:\Python32\Lib\site-packages\pygame\examples\aliens.py
File Edit Format Run Options Windows Help
#cap the framerate
clock.tick(5)
Ln: 312 Col: 20

```

Originally the number 5 was 40, which made the game much faster.

9.4 ANSWERS TO REVIEW QUESTIONS

1. A 'comment' in a computer program is:
 - d. Text to help document the program.

2. You should use a comment:
 - c. To make it easier for someone else to understand your program.

3. Variable names should be:
 - d. Chosen to make your program more easily understood.

4. Python blocks of code are indicated by:
 - d. Indenting the block of code.

LESSON 10 – CONDITIONAL STATEMENTS

Learning objectives:

- Define the programming construct term conditional statement. Outline the purpose of conditional statements in a program
- Define the programming construct term logic test. Outline the purpose of a logic test in a program
- Use Boolean logic expressions in a program
- Use IF...THEN...ELSE conditional statements in a program

10.1 LESSON OVERVIEW

This lesson teaches students how to build logic into their programmes. Students begin to control the flow of their programs based evaluation of certain conditions. Examples using the IF statement are provided. The learning from lesson 9 should be put into practice in all coding activities from this point on.

See the file **NameLen.py** in lesson 10 in the **StudentCodeFiles** folder.

10.2 ADDITIONAL RESOURCES

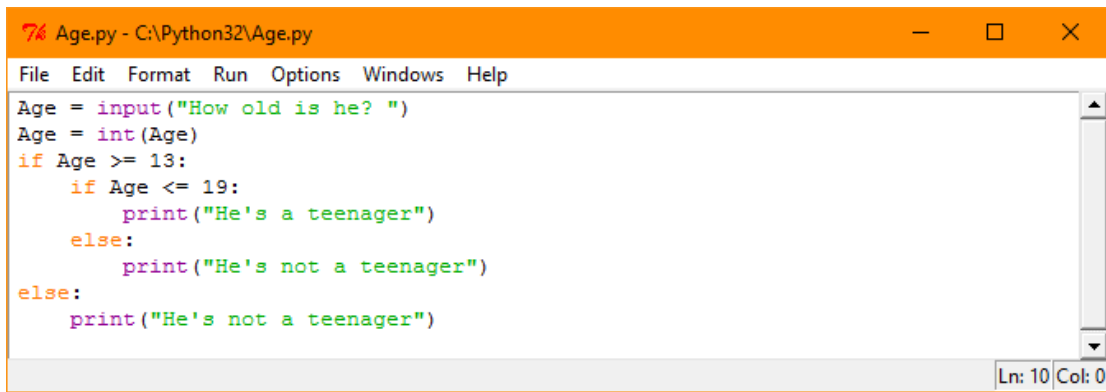
Resource:	Comprehensive Python Manual
URL:	https://docs.Python.org/3.5/tutorial/index.html
Learning Objective:	Deeper understanding of conditional statements.
Suggested Use:	Background information for teachers.

10.3 EXERCISES

Resource:	Multiple IFs
Learning Objective:	AND OR and If
Suggested Use:	Student activity
Solution File:	Age2.py located in the lesson 10 folder in the TeacherCodeFiles folder

- i. Write a program to determine if the user is a teenager. User input is evaluated to see where it is within the age range 13 – 19 inclusive. Use AND and OR to achieve this.
- ii. Challenge students to re-write the same program, this time without using AND or OR.
- iii. In attempting this exercise students should learn that they can use an IF within an IF to achieve the same result as an AND. They may need hints to see that.
- iv. Students in the class may come up with different solutions, so it can be worth comparing solutions.

One possible solution is:



```
Age.py - C:\Python32\Age.py
File Edit Format Run Options Windows Help
Age = input("How old is he? ")
Age = int(Age)
if Age >= 13:
    if Age <= 19:
        print("He's a teenager")
    else:
        print("He's not a teenager")
else:
    print("He's not a teenager")
Ln: 10 Col: 0
```

10.4 ANSWERS TO REVIEW QUESTIONS

1. A Python conditional statement:

- a. Has the keyword 'if' in it

2. In the following code:

```
if password == 'Voldemort' :
    print( 'I guessed your password correctly' )
else :
    print( 'I did not guess your password correctly' )
```

- d. The first or second indented block may be run. Which one runs depends on the value of password.

LESSON 11 – PROCEDURES AND FUNCTIONS

Learning objectives:

- Understand the term parameter. Outline the purpose of parameters in a program
- Understand the term procedure. Outline the purpose of a procedure in a program
- Write and name a procedure in a program
- Understand the term function. Outline the purpose of a function in a program
- Write and name a function in a program

11.1 LESSON OVERVIEW

The benefits of re-using code more than once in a program are realised in this lesson through the exploration of functions and procedures. The arrow head example is used here and developed further in lesson as the code to draw arrows incorporates topics from both lessons. Similarly, the addition exercise provided here is a consolidation of lessons 6 up to lesson 12.

11.2 ADDITIONAL RESOURCES

Resource:	Python Functions Tutorial
URL:	https://www.tutorialspoint.com/python/python_function_s.htm
Learning Objective:	Deeper knowledge of functions
Suggested Use:	Background information for teachers

11.3 EXERCISES

Resource:	Fizz Buzz
URL:	https://blog.codinghorror.com/why-cant-programmers-program/
Learning Objective:	Consolidation of lessons 6-12
Suggested Use:	Student activity
Solution File:	FizzBuzz.py located in the lesson 11 folder in the TeacherCodeFiles folder

This resource suggests an interview question to determine if programmers understand the basics.

'Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".'

The resource claims that many applicants for programming jobs do not understand the basics.

The learning occurring during lessons 6 – 12 contribute to being able to solve the Fizz Buzz problem. By lesson 12, students will have just enough knowledge to write the FizzBuzz program themselves. You can break down the Fizz Buzz program and introduce incrementally during lesson 11 and lesson 12.

Q: Can you write a **logic test** to see if a number is divisible by 3?

A:

```

76 Fizz.py - C:/Python32/Fizz.py
File Edit Format Run Options Windows Help
x = input( "A number: " )
x = int(x)
if( int(x/3)*3 == x ):
    print( "Fizz" )
else:
    print( x )
Ln: 8 Col: 0
    
```

Running this with three different inputs gives the following results:

```

76 Python Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
A number: 7
7
>>> ===== RESTART =====
>>>
A number: 6
Fizz
>>> ===== RESTART =====
>>>
A number: 303
Fizz
>>> |
Ln: 22 Col: 4
    
```

Students will probably need hints to find some way to test for whether a number is exactly divisible by 3. A hint could be "What is `int(35.5)` equal to?". If they don't know, they can try out `int(35.5)` using the Python shell as a calculator.

Once students have the idea of using `int`, they may find other solutions using it.

When students understand loops in this lesson, they are ready for this question:

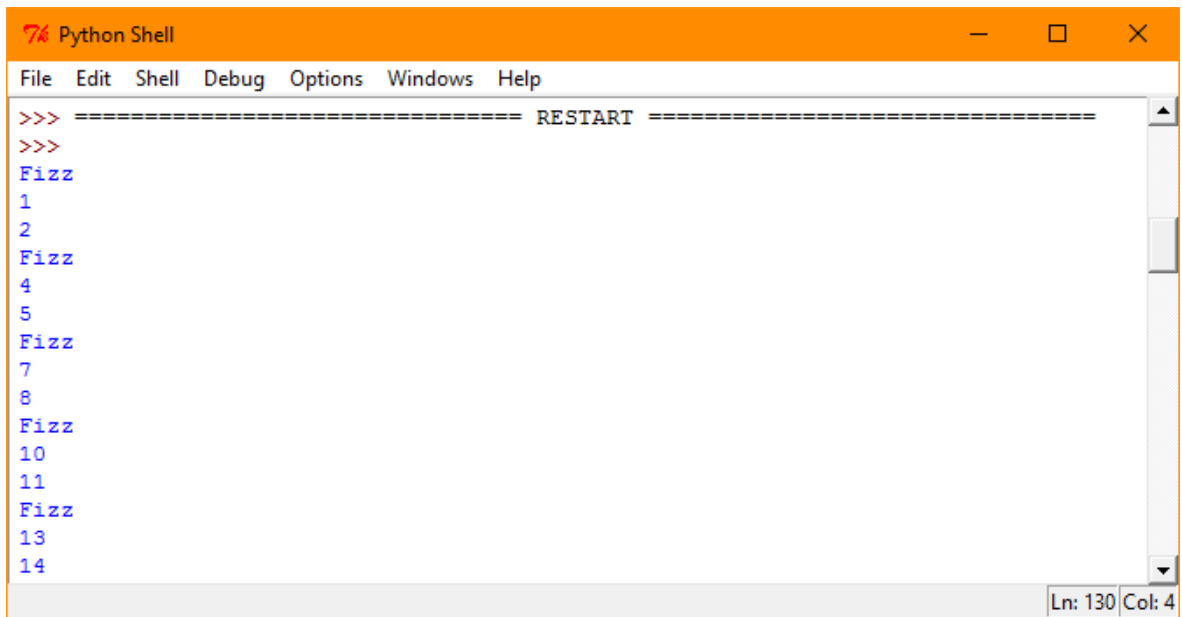
Q: Can you solve the 'Fizz' part of the 'FizzBuzz' problem on its own?

A:

```

76 Fizz.py - C:/Python32/Fizz.py
File Edit Format Run Options Windows Help
for x in range(100):
    if( int(x/3)*3 == x ):
        print( "Fizz" )
    else:
        print( x )
Ln: 1 Col: 0
    
```

Running this gives:



```

Python Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
Fizz
1
2
Fizz
4
5
Fizz
7
8
Fizz
10
11
Fizz
13
14
Ln: 130 Col: 4

```

The program answers the Fizz part of the FizzBuzz question, except it runs from 0 to 99 rather than 1 to 100. Students could fix that using an additional variable, $y=x+1$. (similar to the arrow head example in lesson 11)

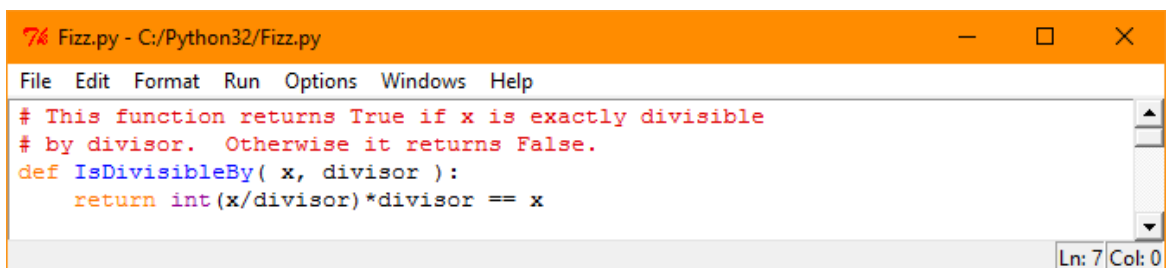
`range()` can have a start and a count, so `range(1,101)` will generate numbers 1 to 100.

Once the students have completed lesson 11 on functions, develop the FizzBuzz challenge further with a question such as this:

Q: Can you put the 'is it divisible by' logic test into a function that gives a Boolean result?

(Remind students to put a comment before the function to enhance their code.)

A:



```

Fizz.py - C:/Python32/Fizz.py
File Edit Format Run Options Windows Help
# This function returns True if x is exactly divisible
# by divisor. Otherwise it returns False.
def IsDivisibleBy( x, divisor ):
    return int(x/divisor)*divisor == x
Ln: 7 Col: 0

```

Finally, at the end of Lesson 11, students can put it all together to get FizzBuzz:

```

74 Fizz.py - C:/Python32/Fizz.py
File Edit Format Run Options Windows Help
# This function returns True if x is exactly divisible
# by divisor.  Otherwise it returns False.
def IsDivisibleBy( x, divisor ):
    return int(x/divisor)*divisor == x

for x in range(1,101):
    if( IsDivisibleBy( x, 3 )):
        if( IsDivisibleBy( x, 5 )):
            print( "FizzBuzz" )
        else:
            print( "Fizz" )
    else:
        if( IsDivisibleBy( x, 5 )):
            print( "Buzz" )
        else:
            print( x )
Ln: 17 Col: 0
    
```

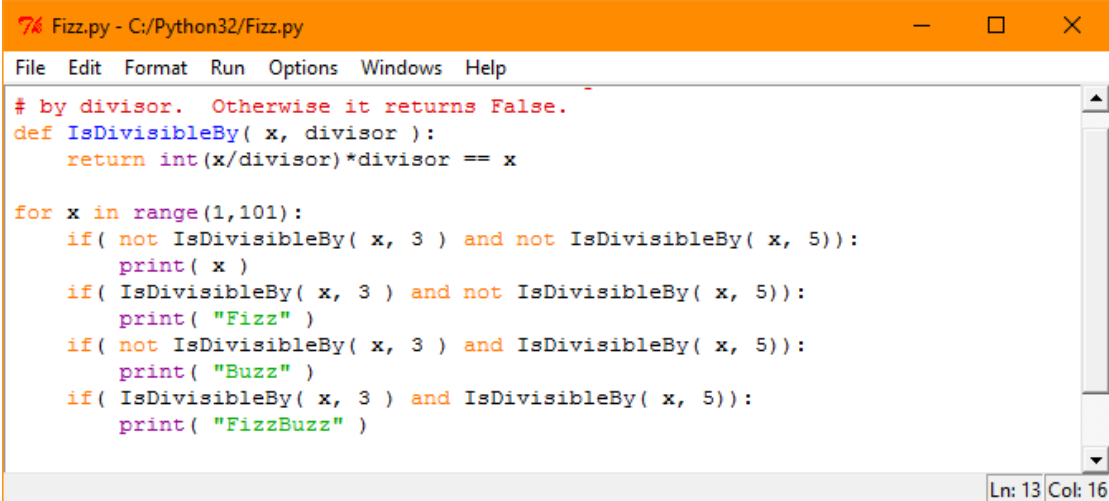
Which when run gives:

```

74 Python Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
Ln: 646 Col: 4
    
```

The use of IF within an IF will be new to the students.

There is another solution that does not use IF within an IF. Instead it uses AND and NOT. It's shown here:

A screenshot of a Python IDE window titled 'Fizz.py - C:/Python32/Fizz.py'. The window contains Python code for a FizzBuzz program. The code defines a function 'IsDivisibleBy' and uses it in a loop to print numbers from 1 to 101, with 'Fizz' for multiples of 3, 'Buzz' for multiples of 5, and 'FizzBuzz' for multiples of both. The status bar at the bottom right shows 'Ln: 13 Col: 16'.

```
7% Fizz.py - C:/Python32/Fizz.py
File Edit Format Run Options Windows Help
# by divisor. Otherwise it returns False.
def IsDivisibleBy( x, divisor ):
    return int(x/divisor)*divisor == x

for x in range(1,101):
    if( not IsDivisibleBy( x, 3 ) and not IsDivisibleBy( x, 5)):
        print( x )
    if( IsDivisibleBy( x, 3 ) and not IsDivisibleBy( x, 5)):
        print( "Fizz" )
    if( not IsDivisibleBy( x, 3 ) and IsDivisibleBy( x, 5)):
        print( "Buzz" )
    if( IsDivisibleBy( x, 3 ) and IsDivisibleBy( x, 5)):
        print( "FizzBuzz" )
Ln: 13 Col: 16
```

This solution isn't as efficient as the previous solution, but still solves the 'FizzBuzz' problem.

11.4 ANSWERS TO REVIEW QUESTIONS

1. A typical function in Python:
 - a. Starts with the keyword `def` and provides some code that returns a value
2. A typical procedure in Python:
 - d. Is like a function, but does not return a value.
3. A parameter is:
 - a. A value which is passed in to a procedure or function for it to use

LESSON 12 – LOOPS

Learning objectives:

- Define the programming construct term loop. Outline the purpose and benefit of looping in a program
- Recognise types of loops used for iteration: for, while, repeat
- Understand the term infinite loop
- Use iteration (looping) in a program like: for, while, repeat

12.1 LESSON OVERVIEW

Students step away from sequential statements in this lesson and explore some basic looping structures. The arrow head example from lesson 11 is further developed here.

See the files **ArrowBig.py** and **ArrowSmall.py** in lesson 12 in the **StudentCodeFiles** folder which relate to the Example: Drawing an Arrowhead in Section 12.4 of the student learning materials.

12.2 ADDITIONAL RESOURCES

Resource:	Advanced looping
URL:	https://wiki.python.org/moin/ForLoop
Learning Objective:	Deeper study of loops
Suggested Use:	Background information for teachers and students

This page contains more information about loops in Python. One of the more useful details is the possibility of creating a loop within a loop. This can be useful, for example, to draw a grid of squares. One loop loops through the rows, and the loop inside this loops through the columns.

12.3 EXERCISES

Resource:	How big can a Python integer be?
Learning Objective:	Infinite loop, Integers
Suggested Use:	Student activity

Ask the students to write a while loop that doubles a number repeatedly.

How big can a Python integer get?

- Python is different from many other computer languages. There isn't a hard fixed limit to the size of an integer in Python. Instead Python can keep doubling a number until it runs out of memory to store the number in.

Resource:	Horizontal Arrow
Learning Objective:	Loops, Procedures, Functions and Parameters
Suggested Use:	Student activity
Solution File:	HorizontalArrow.py located in the lesson 12 folder in the TeacherCodeFiles folder

In Lesson 12 students learned how to draw an arrow head. Can they use the knowledge they have gained to draw a double headed horizontal arrow?

Some helpful tips:

- Use a dot for spaces, until they have the program working, so that they can see where the spaces are being printed.
- Break the problem down into parts (decomposition).
- Use functions for the different parts.
- Start with just a function to draw an arrow head pointing right, and get that part working on its own.

Here's a possible solution:



```

7% HorizontalArrow.py - C:/Python32/HorizontalArrow.py
File Edit Format Run Options Windows Help
def RepeatString( count, string ):
    result = ""
    for i in range( count ):
        result += string
    return result;

def LeftArrow( row, width):
    if( row < width ):
        stars = row+1
    else:
        stars = width*2+1-row
    return RepeatString( width+1-stars, " ") + RepeatString( stars, "*" )

def RightArrow( row, width):
    if( row < width ):
        stars = row+1
    else:
        stars = width*2+1-row
    return RepeatString( stars, "*" ) + RepeatString( width+1 - stars, " " )

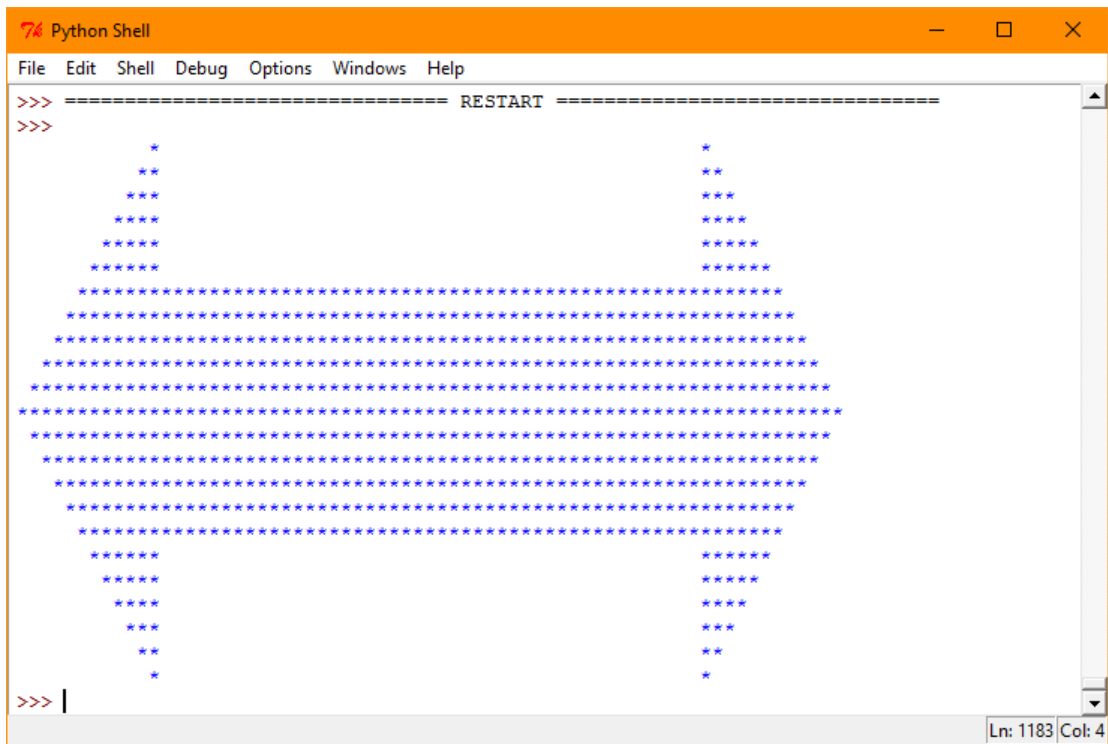
def Shaft( row, height,length ):
    if( (row > height*0.25) and (row < height *0.75 )):
        string = RepeatString( length, "*" )
    else:
        string = RepeatString( length, " " )
    return string;

def Arrow( width, length ):
    for row in range( width*2+1):
        print( LeftArrow( row, width) +
              Shaft( row, width*2, length) +
              RightArrow( row, width) )

Arrow( 11, 45 )
Ln: 35 Col: 0

```

When run this gives:



12.4 ANSWERS TO REVIEW QUESTIONS

1. What is the main purpose of a loop?
 - d. To do some instructions repeatedly.
2. A loop that goes on forever is called?
 - c. An infinite loop.
3. To count through several items, we typically use:
 - d. a for loop
4. To repeat some action several times without counting we typically use:
 - c. a while loop

LESSON 13 – LIBRARIES

Learning objectives:

- Understand the term event. Outline the purpose of an event in a program
- Use event handlers like: mouse click, keyboard input, button click, timer
- Use available generic libraries like: math, random, time

13.1 LESSON OVERVIEW

Students learn about the concept of libraries and use Pygame to achieve some of the ECDL Computing learning objectives.

Boilerplate code serves to illustrate concepts such as events, functions and procedures. Students need to know the definitions of those concepts. They need an outline understanding of the boilerplate code, not an in-depth one.

Note: You will need to install Pygame and the relevant ‘boiler plate’ code before students begin this lesson as outlined in the previous sections **Installing The Pygame Python library** and **Installing Boilerplate Code**.

See the file **Grass.py** in lesson 13 in the **StudentCodeFiles** folder.

13.2 ADDITIONAL RESOURCES

Resource:	Detailed Pygame Library Documentation
URL:	Various at http://www.pygame.org/docs
Learning Objective:	Extend understanding of libraries
Suggested Use:	Background information for the teacher

The Pygame graphics and animation library has many more functions and procedures than used in this course. There is online documentation about what these functions and procedures are. The library contains:

- Functions for drawing shapes:
<http://www.pygame.org/docs/ref/draw.html>
- Functions for making sounds:
<http://www.pygame.org/docs/ref/music.html>
- Functions for reacting to ‘events’ such as doing something when a key on the keyboard is pressed:
<http://www.pygame.org/docs/ref/event.html>

To fulfil the ECDL Computing syllabus objectives students do not need to know what functions are included in Pygame. Students do, however, need to know that Pygame is a library, and that it contains functions that can be reused.

13.3 EXERCISES

Resource:	Python libraries
Learning Objective:	Get a sense of the range of libraries available for Python
Suggested Use:	Student activity

For this exercise students can work in pairs:

- i. Give each student team a letter of the alphabet (excluding Y) and ask them to find a Python library whose name starts with that letter.
- ii. Students find a function in that library and report back to the class on what the library is called and explain what the function within that library is for.

13.4 ANSWERS TO REVIEW QUESTIONS

1. An event is:
 - b. Something that happens that a program should react to.
2. The function 'randint' is found in which library?
 - d. The random library.
3. The string named greeting has the value 'Good'. Which command sets it to have the value 'Good Morning'?
 - b. `greeting = greeting + " Morning"`
4. The tuple named colour has the value (0,0,255). Which command sets it to have the value (255,0,255)?
 - c. `colour = (255 , 0+0 , 255)`
5. Which of the following does not generate events in Python with Pygame?
 - d. Battery low.

LESSON 14 – RECURSION

Learning objectives:

- Understand the term recursion
- Loop
- Procedure
- Function
- Variable
- Parameter

14.1 LESSON OVERVIEW

Recursion is possibly the hardest topic in the course. The main point for students at this level to absorb is that recursion works by a function calling itself.

See the file **Fern.py** in lesson 14 in the **StudentCodeFiles** folder relates to the Example: Recursion for Fractal Fern in Section 14.2. The **Curve.py** in lesson 14 in the **TeacherCodeFiles** folder relates to the Example: Drawing a Curved Line in Section 14.1.

14.2 ADDITIONAL RESOURCES

Resource:	Recursive Drawing
URL:	http://recursivedrawing.com/
Learning Objective:	Recursion
Suggested Use:	Background information for teacher and student

This URL leads into the idea of fractals and recursion in making shapes.

The site has a video that builds up recursive images where a shape contains a copy of itself.

Resource:	Recursion without Drawing
URL:	http://www.programmerinterview.com/index.php/recursion/explanation-of-recursion/
Learning Objective:	Recursion
Suggested Use:	Alternative approaches to teaching recursion.

This resource shows a typical approach to teaching recursion that does not use graphics.

Recursion is explained using ‘factorials’ as an example. The number 7!, which is read as “7 factorial”, for example is $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$. The example URL uses the java language to explain. In Python the definition of n! is

```
def factorial( n ) :
    if n==1:
        return 1
    else:
        return n * factorial(n-1)
```

This approach to teaching recursion has some disadvantages:

- It is a less motivating example than drawing a fractal.
- It is very easy to write an iterative function that evaluates factorials, so the value of recursion is not clear.

14.3 EXERCISES

Resource:	Tower of Hanoi
Topic:	Recursion Pseudocode
Learning Objective:	Understand recursion
Suggested Use:	Student activity

- Students work in pairs to find a description online of the Tower of Hanoi problem.
- Students work to solve the problem.
- Students explain their strategy and more importantly, why their strategy works.

One piece of learning from this is that a problem that has a recursive nature to it can often be solved either by a recursive algorithm or by an iterative one.

Title:	Recursion in Board Games
URL:	http://mediocrechess.blogspot.ie/2006/12/guide-alpha-beta-search.html
Learning Objective:	Recursion – advanced techniques
Suggested Use:	Student self-study

Any students who play board games such as chess and ‘Go’ are already familiar with an example of recursive thinking. Often players will consider what could the game board look like in 6 moves time? This involves thinking about current moves, and then following them by 5 further moves, and so on...

This resource is suitable for chess playing students who are really keen on understanding how computers can use recursive algorithms to play chess.

In pseudocode, a recursive algorithm to print possible board positions could be:

```
print_possible_positions_from( position, number_of_moves_left):
    if( number_of_moves_left == 0 ):
        print position
        return
    for move in valid_moves_from( position ):
```

```
print_possible_positions_from( position + move, number_of_moves_left -1)
```

The structure is the same as other recursions. There is a logic test to stop recursing if looking zero moves ahead. There is code to make one move and then look ahead one fewer moves, by the function calling itself recursively.

A difference between the chess algorithm and the fractal fern algorithm is that instead of the function calling itself four times, it has a loop, and calls itself once for each available move.

Real chess and 'Go' programs use a more complex recursion that takes account of how good or bad the moves seem to be. This reduces the number of positions considered when looking ahead, an action called 'pruning'. The resource URL includes a link to a description of pruning.

14.4 ANSWERS TO REVIEW QUESTIONS

1. A recursive algorithm is one that:
 - d. Breaks a problem into smaller pieces, and then applies the same approach to those smaller pieces.
2. A function that calls itself:
 - b. Could be an example of recursion
3. An infinite loop is:
 - c. A loop that runs forever

LESSON 15 – TESTING AND MODIFICATION

Learning objectives:

- Understand types of errors in a program like: syntax, logic
- Understand the benefits of testing and debugging a program to resolve errors
- Identify and fix a syntax error in a program like: incorrect spelling, missing punctuation
- Identify and fix a logic error in a program like: incorrect Boolean expression, incorrect data type
- Check your program against the requirements of the initial description
- Identify enhancements, improvements to the program that may meet additional, related needs
- Describe the completed program, communicating purpose and value

15.1 LESSON OVERVIEW

The lesson covers the activities in creating fixing and maintaining a computer program. Use this lesson to reinforce concepts covered earlier in the course.

Exploring bugs and how to find them will remind the student of work earlier in the course, for example infinite loops, and that computer languages are formal languages.

Checking a program against a specification and working on enhancements will remind the student of work in the 'Thinking Like a Programmer' lesson.

See the files **Age.py** and **SyntaxError.py** in lesson 15 in the **StudentCodeFiles** folder.

15.2 ADDITIONAL RESOURCES

Resource:	Well-defined Software Specifications
URL:	https://belitsoft.com/software-requirements-specification-helps-protect-it-projects-failure
Learning Objective:	Software specifications
Suggested Use:	Context for teachers and students

This URL has a variant on a popular cartoon (<http://www.projectcartoon.com>) showing the difference between a client's expectations for software and what actually gets built. It then continues with a discussion of common mistakes in writing a software specification. The cartoon could be used to initiate a discussion on the necessity for well-defined software requirements.

15.3 EXERCISES

Resource:	Annotate this
URL:	http://www.annedawson.net/Python3Programs.txt
Learning Objective:	Identify common code errors and best practice in producing readable code
Suggested Use:	Student activity

Simple and more complex Python 3 programs to modify for this activity can be found at this URL. Alternatively perform an Internet search for 'simple python 3 programs'.

- i. Students are given a printed copy of a short program that has a syntax error and a logic error in it.
- ii. Students annotate the program to mark the two errors and classify them as syntax or logic errors.
- iii. Students annotate the program to show features that make it easier or less easy to understand the code e.g.:
 - good/bad comments
 - good/bad choices for variable and function names

15.4 ANSWERS TO REVIEW QUESTIONS

1. Which expression has a syntax error?
 - d. $2+(4*8))$
2. A logic test was intended to test if variable c,d and e are in order with c smallest and e largest. Which of the following has a logic error:
 - d. if (e<d) and (d<c) :
3. Program testing is:
 - d. A formal approach to finding bugs in a program, by running the program with different inputs.
4. Debugging is:
 - c. Work done to track down the cause of a problem and fix it.
5. One of the initial requirement for a particular program is that it ask for and remember your name. Which of the following would you expect to see in the code to help meet that requirement?
 - b. An input() statement
6. A software company wants to make incremental improvements to version 1 of its program for managing recipes. Which of the following is a good way to identify possible enhancements?
 - c. Look for small changes that would make the program useful to more people.
7. When describing the commercial value of a new program, the best way is:
 - b. Describe the benefit to users and back it up with numerical data if you can.

TRAINING PLAN

The following is a suggested schedule for the delivery of ECDL Computing using the supplied resources.

Please note that this is for guidance purposes only and provides an indication of the weighting of various lessons in the learning materials. This schedule should be adapted to suit candidate requirements.

Lesson		Classes
1	THINKING LIKE A PROGRAMMER	2
2	SOFTWARE DEVELOPMENT	2
3	ALGORITHMS	3
4	GETTING STARTED	1
5	PERFORMING CALCULATIONS	1
6	DATA TYPES AND VARIABLES	2
7	TRUE OR FALSE	2
8	AGGREGATE DATA TYPES	1
9	ENHANCE YOUR CODE	1
10	CONDITIONAL STATEMENTS	2
11	PROCEDURES AND FUNCTIONS	1
12	LOOPS	3
13	LIBRARIES	3
14	RECURSION	2
15	TESTING AND MODIFICATION	2
	Test Practice	1
	Test Session	1
	Total	30

ECDL Malta

Flat 2, St John's Flats,
Naxxar Road,
Birkirkara BKR9040
Email: info@ecd.com.mt

Phone: +356 2166 7706
Web: www.ecdl.com.mt