ICDL Foundation

# ICDL COMPUTING

## Teacher Handbook


Provided by:
«Name»

Syllabus 1.0

## ICDL 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 ICDL 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 ICDL 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?

ICDL 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 is 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 – ICDL Computing.

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

# ICDL COMPUTING

# OVERVIEW OF ICDL COMPUTING RESOURCES

To support teachers and learners in the implementation of the ICDL Computing syllabus a range of teaching and learning resources have been created. This implementation of ICDL 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 ICDL 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 ICDL 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 ICDL 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 ICDL 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** | 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.<br><br>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. |

| | |
|---|---|
| **Resource:** | CAS Computational Thinking - A Guide for Teachers |
| **URL:** | https://www.computingatschool.org.uk/<br>You will need to register at the site first, then locate:<br>https://community.computingatschool.org.uk/resources/2324 |
| **Learning Objective:** | Understand the term Computational Thinking |
| **Suggested Use:** | Comprehensive guide for teachers.<br>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/372410/1/372410UnderstdCT.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 ICDL 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 ICDL 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** - ICDL 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 ICDL 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**.

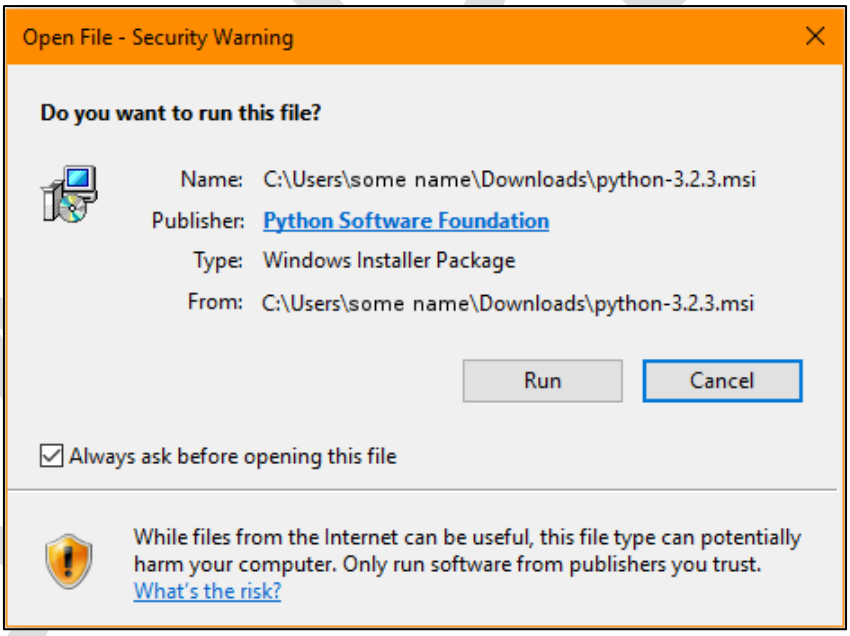
Installation instructions are outlined in the next section.

# INSTALLING PYTHON

ICDL 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:

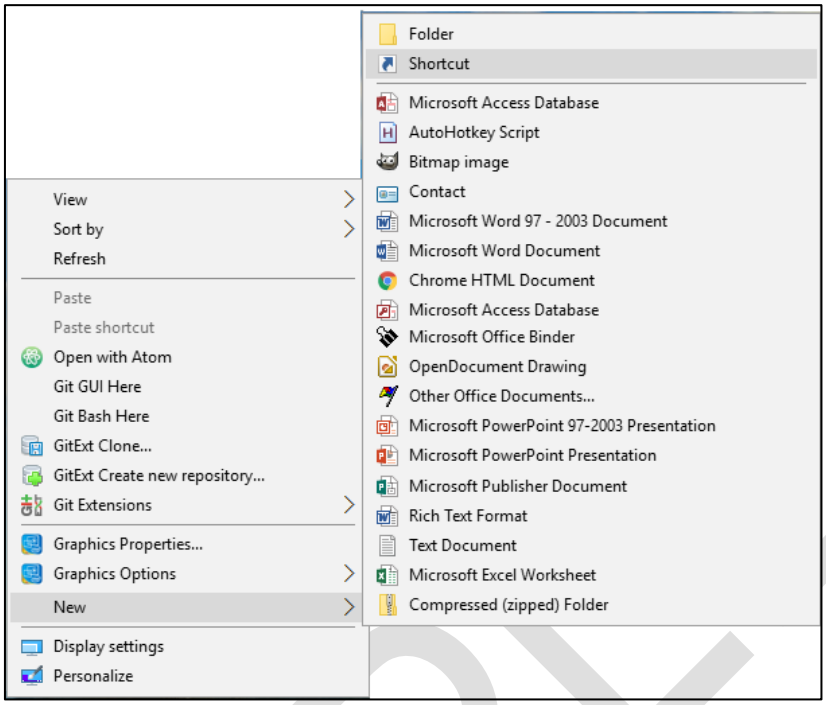



5. Click **Run**.
6. Choose the default options that follow. Take a note of the location where Python is being installed in the **Select Destination Directory** dialog box, for example, **C:\Python32**
7. Click **Finish** to exit the installer.

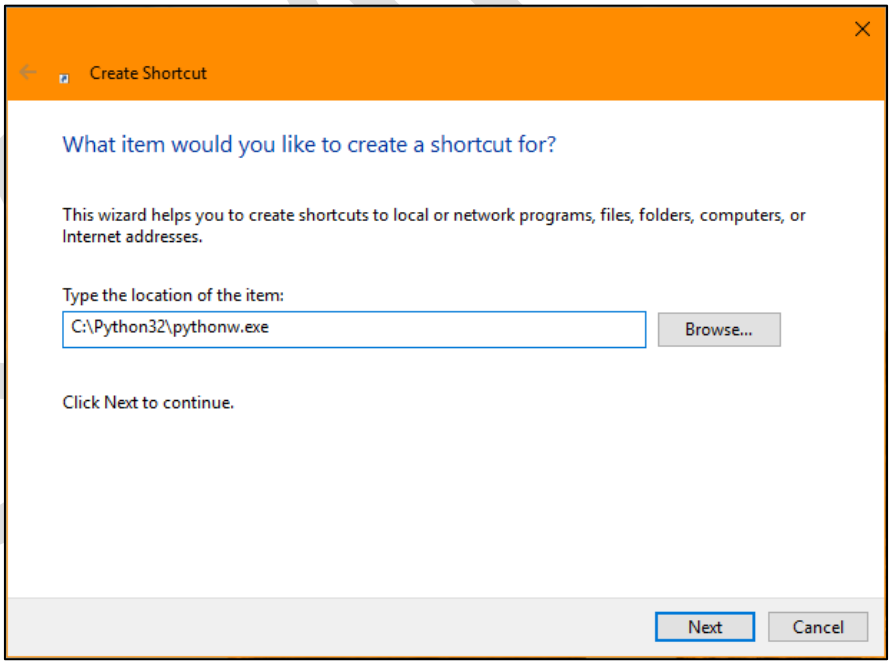
**Create a shortcut icon to launch the Python Shell from the desktop:**

1. Right-click anywhere on the desktop.

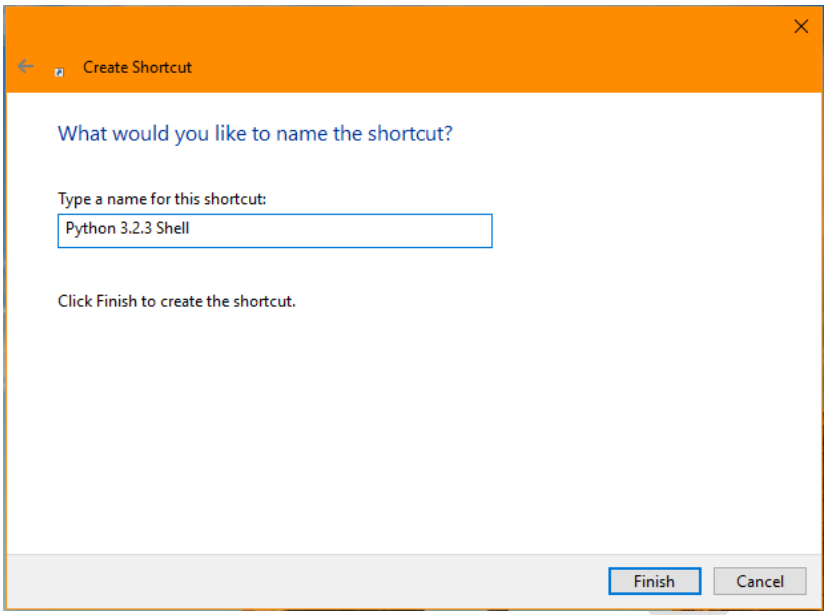
2. From the menu that appears navigate to **New > Shortcut**.




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




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

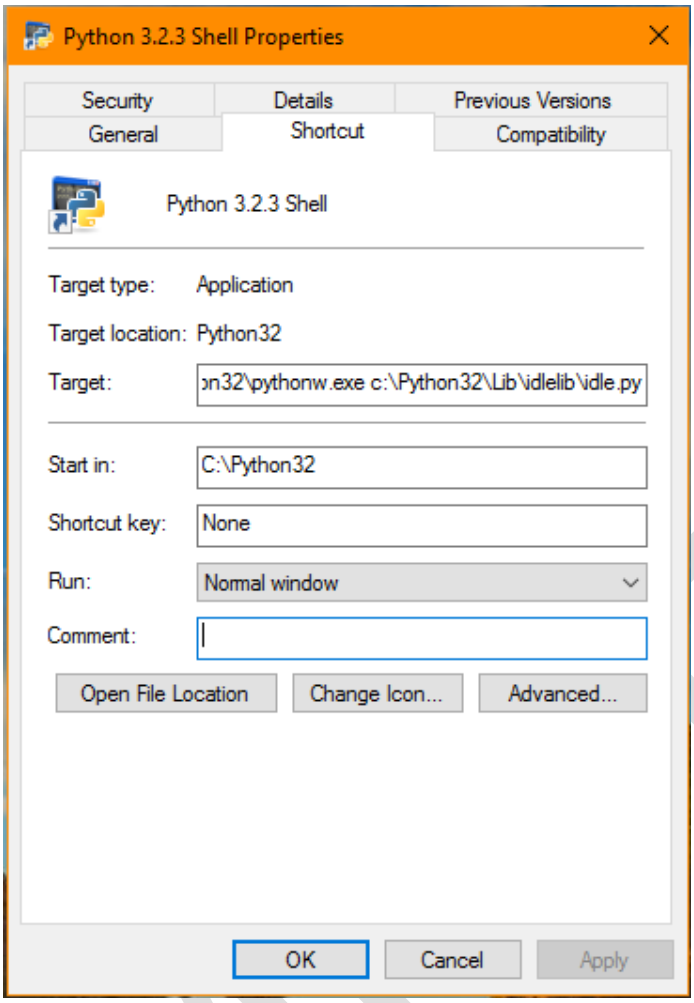



8. Click **Finish.**
9. Right click on the new **Python 3.2.3 Shell** shortcut icon on the desktop.

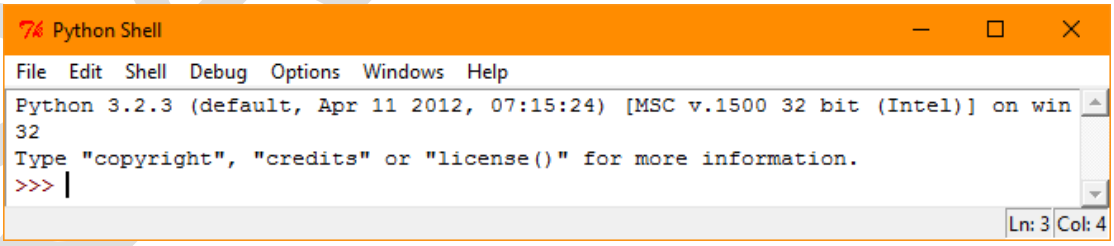

10. Select **Properties** from the menu.
11. In the dialog box, modify the target to

    **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.



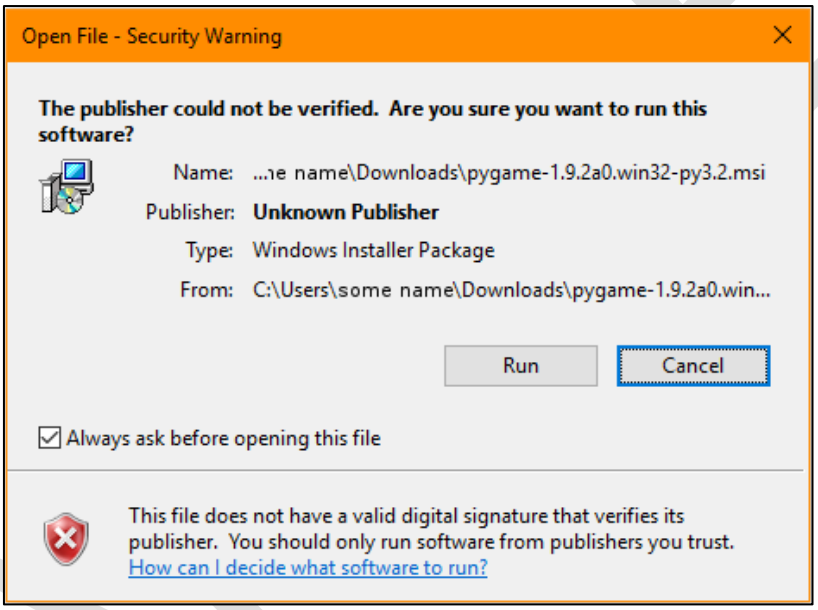

15. Click the **X** in top right hand corner to close the Python shell.

# INSTALLING THE PYGAME PYTHON LIBRARY

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

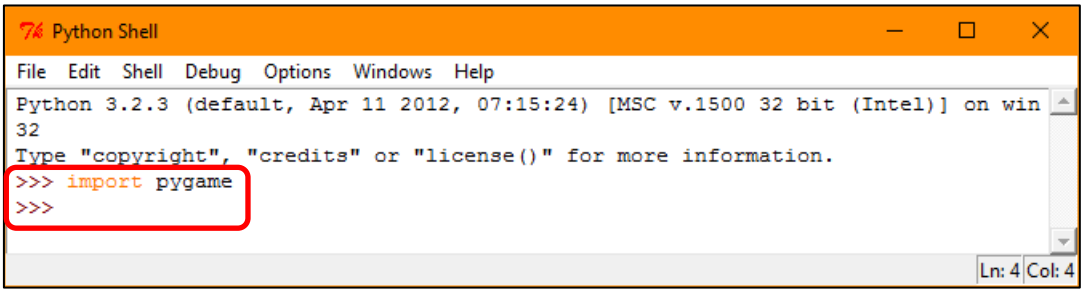
**To Install Pygame:**

1. Go to http://www.pygame.org/download.shtml
2. Under **Windows** find the version of Pygame used in the ICDL 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**:



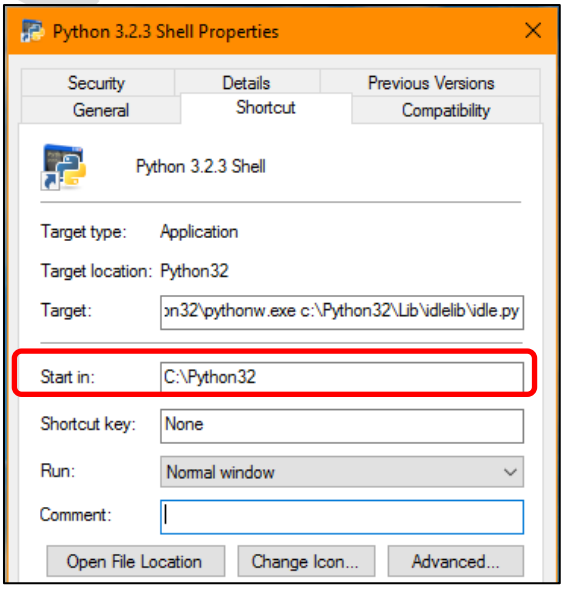

10. Pygame has been installed correctly if the prompt **>>>** appears with no error message, as shown above.

# 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_9YNyjXEeCOmmBaI/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 ICDL 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 ICDL 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 coking 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.

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<br>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<br>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<br>Understand inputs and outputs<br>Understand scope of computing<br>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

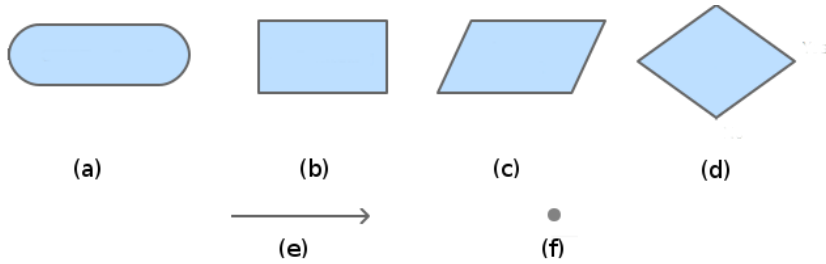
1. A sequence is:

   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 $180^0$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 ICDL 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:

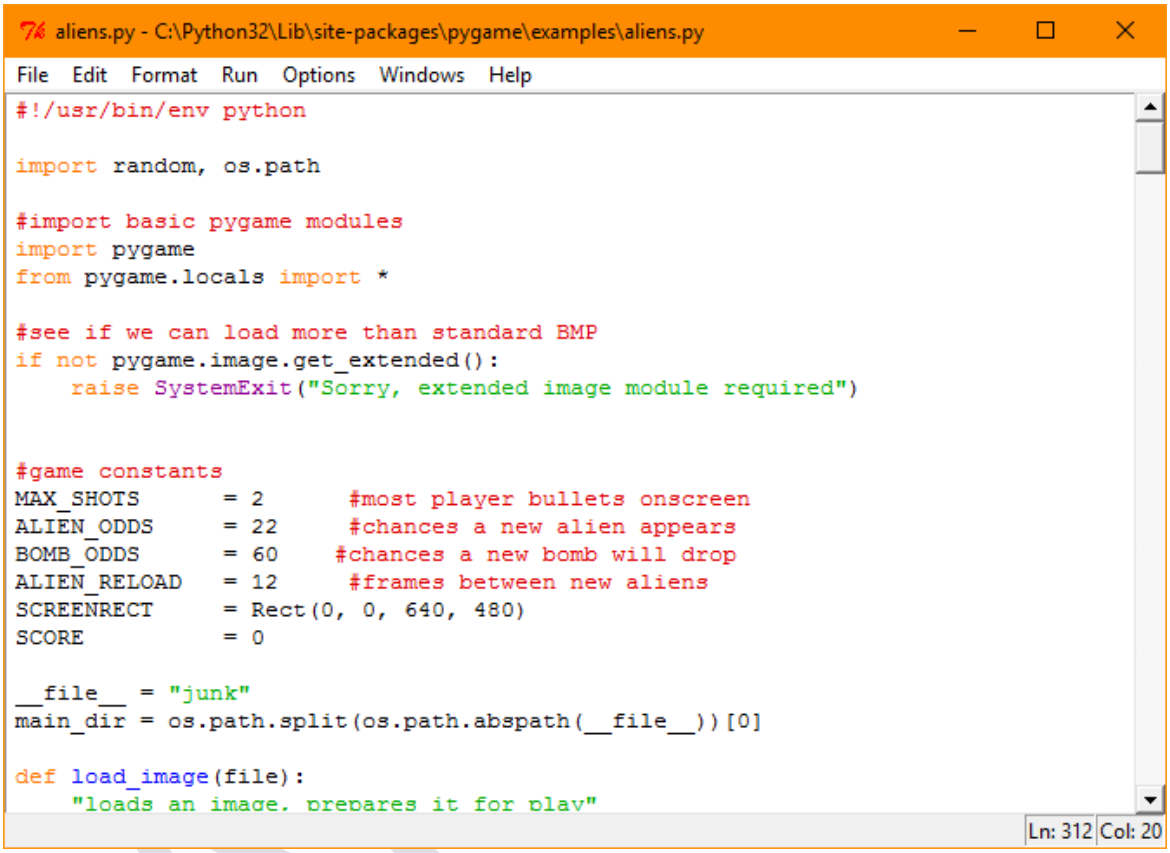
C:\Python32\Lib\site-packages\pygame\examples and choose aliens.py.

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

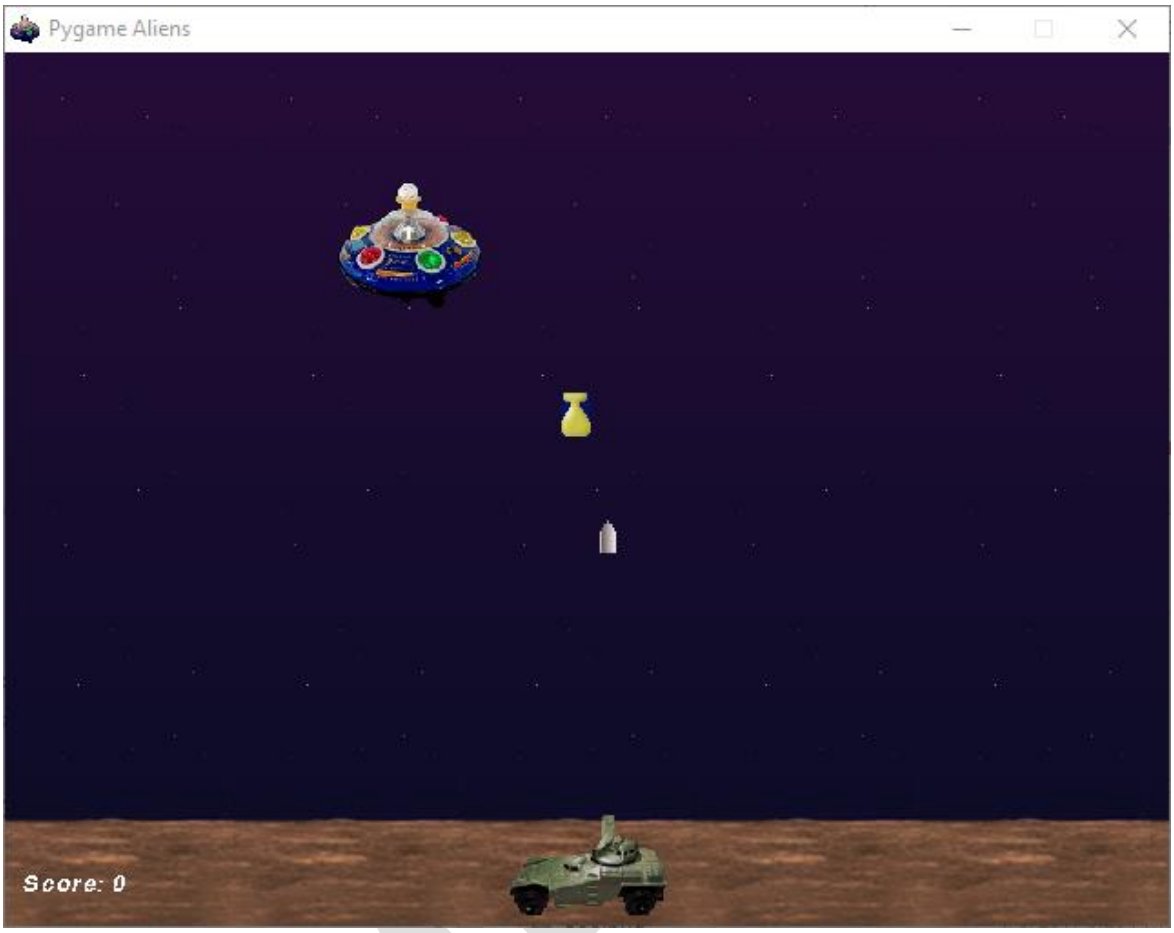
__file__ = "junk"

As shown in the screenshot:

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




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.