Software Engineering Stage 6 (Year 11) – teacher support resource

**The object-oriented paradigm**

# Teacher support resource

**Teacher note: t**his resource has been designed to facilitate the ready conversion into a student booklet by removing the answers within the response windows. Teacher notes can be deleted before distributing to students.

Student name:

Class:

Teacher:

Contents

[Teacher support resource 1](#_Toc150844752)

[Unit overview 5](#_Toc150844753)

[Assessment task overview 7](#_Toc150844754)

[Steps to success 8](#_Toc150844755)

[What is the teacher looking for? 9](#_Toc150844756)

[Glossary 10](#_Toc150844757)

[Understanding OOP 13](#_Toc150844758)

[Apply the key features of an object-oriented programming (OOP) language 14](#_Toc150844759)

[Objects 19](#_Toc150844760)

[Classes 21](#_Toc150844761)

[Objects and classes 25](#_Toc150844762)

[Encapsulation 27](#_Toc150844763)

[Abstraction 28](#_Toc150844764)

[Inheritance 28](#_Toc150844765)

[Generalisation 33](#_Toc150844766)

[Polymorphism 34](#_Toc150844767)

[OOP in C# 39](#_Toc150844768)

[Compare procedural programming with OOP 42](#_Toc150844769)

[Procedural programming 42](#_Toc150844770)

[Object-oriented programming 42](#_Toc150844771)

[Definitions 43](#_Toc150844772)

[Summary 44](#_Toc150844773)

[Historical context 44](#_Toc150844774)

[Imperative and OOP 46](#_Toc150844775)

[Object-oriented programming 48](#_Toc150844776)

[Use data flow diagrams, structure charts and class diagrams to represent a system 50](#_Toc150844777)

[Data flow diagrams 50](#_Toc150844778)

[Structure charts 53](#_Toc150844779)

[Steps to create a structure diagram 55](#_Toc150844780)

[Class diagrams 59](#_Toc150844781)

[Describe the process of design used to develop code in an OOP language 65](#_Toc150844782)

[Task definition 66](#_Toc150844783)

[Top-down and bottom-up 67](#_Toc150844784)

[Facade pattern 68](#_Toc150844785)

[Agility 69](#_Toc150844786)

[Assess the effectiveness of programming code developed to implement an algorithm 70](#_Toc150844787)

[Investigate how OOP languages handle message-passing between objects 72](#_Toc150844788)

[Explain code optimisation in software engineering 77](#_Toc150844789)

[Outline the features of OOP that support collaborative code development 78](#_Toc150844790)

[Consistency 78](#_Toc150844791)

[Code commenting 78](#_Toc150844792)

[Version control 79](#_Toc150844793)

[Feedback 79](#_Toc150844794)

[Programming in OOP 81](#_Toc150844795)

[Create a text-based adventure game 83](#_Toc150844796)

[Part 1 – setting the scene 85](#_Toc150844797)

[Classes and objects 89](#_Toc150844798)

[Using getter and setter methods 92](#_Toc150844799)

[Linking the caves 95](#_Toc150844800)

[Move around the caves 101](#_Toc150844801)

[Part 2 – adding the characters 104](#_Toc150844802)

[Multiple characters 108](#_Toc150844803)

[Fighting Harry the Wumpus 112](#_Toc150844804)

[Create an enemy in the room 115](#_Toc150844805)

[Putting Harry in a cave 117](#_Toc150844806)

[Part 3 – interacting with the game 119](#_Toc150844807)

[If you lose a fight with an enemy, the game should end 121](#_Toc150844808)

[Add an additional way to interact with the characters 123](#_Toc150844809)

[Create a friend subclass 123](#_Toc150844810)

[Part 4 – extending the game 126](#_Toc150844811)

[Creating an Item class 126](#_Toc150844812)

[Add items to the player’s bag 129](#_Toc150844813)

[Make the game last longer 131](#_Toc150844814)

[Design and implement computer programs involving branching, iteration and functions in an OOP language for an identified need or opportunity 134](#_Toc150844815)

[Implement and modify OOP programming code 134](#_Toc150844816)

[Clear and uncluttered mainline 134](#_Toc150844817)

[Use of a clear modular structure 134](#_Toc150844818)

[One logical task per subroutine 135](#_Toc150844819)

[Use of stubs 135](#_Toc150844820)

[Use of control structures and data structures 136](#_Toc150844821)

[Ease of maintenance 136](#_Toc150844822)

[Version control 137](#_Toc150844823)

[Regular backup 137](#_Toc150844824)

[Apply methodologies to test and evaluate code 138](#_Toc150844825)

[Unit, subsystem and system testing 138](#_Toc150844826)

[Black, white and grey box testing 139](#_Toc150844827)

[Quality assurance 140](#_Toc150844828)

[Journal of progress and pitfalls 141](#_Toc150844829)

[Appendix 144](#_Toc150844830)

[Hunt the Wumpus Adventure Game Code 154](#_Toc150844831)

[References 162](#_Toc150844832)

# Unit overview

In this unit students will develop a fundamental understanding of the object-oriented paradigm (OOP). Students will investigate the key concepts of this paradigm. They will differentiate between OOP and other paradigms investigated in the fundamentals of programming focus area. Students will be guided through programming and coding activities that reinforce their understanding.

During Weeks 1 and 2 of the learning sequence, students will be introduced to objects, methods and attributes by identifying their use in Python's Turtle module. They will then use starting code provided in the Appendix of the teacher support resource (TSR) to write their own classes, objects, attributes and methods to draw shapes.

During Weeks 3 and 4 of the learning sequence, students complete a series of workbook activities to consolidate their understanding and learn about modelling tools used in object-oriented programming (OOP).

During Weeks 5 to 8 of the learning sequence, students are guided step-by-step through the creation of a text-based role-playing game (RPG) using key OOP concepts.

During Weeks 9 and 10 of the learning sequence students test, evaluate and submit their projects. Students present their work to the class and are peer assessed via a Q&A session.

This TSR:

* is mapped directly to the content from the object -orientated paradigm (OOP) focus area of the Year 11 Software Engineering syllabus
* is designed to accompany the scope and sequence, program of learning and assessment task for this unit
* assumes no prior knowledge of OOP and introduces the unit through a series of simple coding activities.

The skills required for success in this unit include [computational thinking [PDF 41.6 KB]](https://www.csiro.au/-/media/Digital-Careers/Files/Resources/CTIA-Worksheets/DigitalCareers_CTIAWorksheets_CTdefinitions.pdf), design thinking and systems thinking skills.

**Important assessment note:**

The Year 11 formal school-based assessment program is to reflect the following requirements.

**Three** assessment tasks:

* the minimum weighting for a task is 20%
* the maximum weighting for a task is 40%
* only one task **may** be a written examination.
* one task **must** be based on a project.

Many schools will implement Year 11 examination periods to assist students to determine their progress and choices for Year 12 study.

The remaining 2 tasks would be used to assess student knowledge and skills in the practical application of the content.

The optional sample assessment task maps to the key concepts introduced during the theory in OOP unit and assumes an assessment task for this focus area.

The second sample assessment will accompany the Programming Mechatronics unit where a mechatronic solution is coded using object-oriented programming. This task accompanies the published sample scope and sequence. Teachers will need to need to choose which tasks to assess.

**Alternatives to consider**

Teachers may choose to adapt these assessment tasks to their contexts by:

1. implementing a stand-alone assessment task for the OOP focus area
2. integrating this task with Task 1 from the Programming fundamentals unit
3. adopting the published assessment schedule and use OOP to program a mechatronic solution (enabling a third assessment task for this course to be a final examination)
4. offering projects in one or more of the focus areas as formatively assessed and providing opportunities for group work commensurate with industry practice.

# Assessment task overview

**Type of task:** translate a classic role-playing adventure game (RPG) from a procedural paradigm to an object-oriented programming (OOP) paradigm using Python.

**Outcomes being assessed:**

A student:

* describes methods used to plan, develop and engineer software solutions **SE-11-01**
* explains how structural elements are used to develop programming code **SE-11-02**
* applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
* applies language structures to refine code **SE-11-08**
* manages and documents the development of a software project **SE-11-09**

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/content/n11/fa332a6f82) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

**Suggested weighting:** 35%

Students:

* use Python and an object-oriented approach to create a computer game based around a text-based role-playing adventure that was originally written in an imperative or procedural language
* modify a guided solution based loosely on the famous vintage classic Hunt the Wumpus, including an original scenario, characters, and gameplay
* modify the classes, objects, attributes, and methods of the guided solution to achieve this originality
* provide documentation including a journal and modelling diagrams
* present your solution to the class including a question and answer (Q&A) segment.

This project can be adapted for use as a standalone sample assessment task of the object-oriented paradigm focus area. It has been designed to map to the syllabus content for this focus area. An alternative OOP/Mechatronic assessment task aligns with the published scope and sequence and will be available with published resources.

## Steps to success

Table 1 – assessment preparation schedule

|  |  |
| --- | --- |
| Steps | What I need to do |
| **Identifying and defining**  Investigate why object-oriented programming languages are suited to the development of computer games. | * Design the characters and the environment for your game. * Define these characters as objects. * Identify common characteristics (attributes) that may be inherited from a parent class. * Identify how characters (objects) will interact with each other and their environments through methods. * Read [The Genesis of Wumpus](https://www.atariarchives.org/bcc1/showpage.php?page=247). * Investigate how procedural languages were used. * Establish a quality success criterion for your project. |
| **Research and planning**  Document using project management tools.  Plan the game you will create. | * Apply appropriate project management tools to develop the project by creating: * a structure chart * data dictionary * a class diagram. |
| **Producing and implementing**  Develop your game | * Create the environment (set the scene). * Create the classes and objects. * Create the attributes and methods to move and interact. * Add the characters and their behaviours. * Interact with the characters and environment. * Add items and gameplay. |
| **Testing and evaluating**  Review and improve your game | * Apply methodologies to test and evaluate the code including: * unit, subsystem, and system testing * black, white, and grey box testing * quality assurance. * Analyse and evaluate your solution against the quality success criteria. |

## What is the teacher looking for?

This task will require students to step through activities from the accompanying teacher support resource. This will provide the essential OOP concepts and skills to develop a simple RPG text-based adventure game. Through completing the steps, students begin examining how they will modify the game.

Students research the development of a vintage classic game written using a programming language from the procedural or imperative paradigm and consider how this would be coded in the object-oriented paradigm using Python.

The activities of the teacher support resource culminate in an adventure game loosely based on the vintage classic [Hunt the Wumpus](https://en.wikipedia.org/wiki/Hunt_the_Wumpus).

Students play and analyse the game to identify the control and data structures that could be created using classes, objects, attributes and methods used in interactions.

Students modify the games environment, characters, items and gameplay to create their own version of the project.

## Glossary

Many of the following words will gather more meaning to you as you work through this booklet. Each time you see an unfamiliar word in bold throughout this workbook you can add its definition in the table below in case you need to refer later.

|  |  |
| --- | --- |
| Word | Definition |
| abstraction | Abstraction is the process of simplifying complex systems by breaking them down into smaller, more manageable components. It enables focus upon the essential features and behaviours of an object while ignoring the implementation details that are not immediately relevant |
| agility | Agility is the rapid and adaptive response to change via effective communication between stakeholders. |
| algorithm | A step-by-step procedure required to solve a problem. Algorithms may be presented in many ways, for example written instructions, flow charts or using a computer programming language |
| black box testing | A testing technique where there is no knowledge of the internal workings of an application. A user will provide inputs and observe the outputs in order to determine functionality of the program. |
| branching | When an algorithm makes a choice to do one of 2 (or more things) this is called branching. The most common programming ‘statement’ used to branch is the ‘IF’ statement. |
| class diagrams | Class diagrams provide a visual representation of systems that are implemented using the object-oriented paradigm. They model classes, their attributes and methods, and the relationships between classes. |
| classes | A blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behaviour (member functions or methods). |
| code optimisation | The process of transforming a piece of code to make it more efficient in terms of time or space, without changing its output or side-effects. |
| computational thinking | A process in which a problem is analysed and solved so that a human, machine, or computer can effectively implement the solution. It involves using strategies to organise data logically, break down problems into parts, interpret patterns and design, and implement algorithms to solve problems. |
| control structures | Algorithms are developed using the basic control structures of sequence, selection, and repetition. Writing code incorporates combinations of these control structures. |
| data flow diagrams | A visual representation of the information flow through a process or system. DFDs clarify process or system operations to discover potential problems, improve efficiency, and develop better processes. |
| data structure | A particular way data is arranged so it can be saved in memory and retrieved for later use. |
| data type | A way of classifying data that tells the computer how to use and interpret it. Different data types have different properties, such as size, range, and operations. Examples include string/text, character, integer, floating point or real, date and time and Boolean. |
| encapsulation | Involves bundling together data and methods that operate on the data into a single unit called a class. |
| facade pattern | Used to hide the complexities of a system and provides an interface to the user to access the system. |
| function | A block of code that performs a specific task. It can be called and reused multiple times. Information can be passed to a function and can be sent back. |
| generalisation | A process of extracting shared characteristics from 2 or more objects/processes and combining them into generalising objects/processes. |
| inheritance | Allows new classes to take on the attributes and methods of an existing class. |
| iteration | Is the repetition of a process. |
| object-oriented programming (OOP) | A paradigm based on the concept of ‘objects’ that can contain data and code in the form of procedures. OOP language is a language based on the principles of ‘C’. |
| object | It is a basic unit of Object-Oriented Programming and represent real-life entities. An Object is an instance of a Class. |
| polymorphism | Allows objects of different types to be accessed through the same interface. |
| procedural programming | A method of programming where the program is divided into functions. A program consists of data and procedures (modules) that operate on the data. Data and procedures are treated as separate entities. |
| structure charts | Represent a system by showing the separate subroutines that make up the system and their relationship to each other. |
| stubs | A piece of code used to stand in for some other programming functionality. It may simulate the behaviour of existing code or be a temporary substitute for yet-to-be-developed code. |
| version control | Is the practice of tracking and managing changes to software code. |

**Teacher note:** for students with an EALD background, the glossary can be provided complete so that they have additional time to understand the key terms with bilingual dictionaries. The glossary can be provided to students in their preferred communication mode.

# Understanding OOP

**Teacher note:** the following tutorial series: [Object-oriented programming getting started for Australian teachers](https://sites.google.com/dltv.vic.edu.au/object-oriented) is a detailed investigation of what OOP is and why it is used.

Resources for learning OOP with [Python and/or Javascript are also available.](https://sites.google.com/dltv.vic.edu.au/object-oriented/recommended-resources-for-starting-general-purpose-programming?authuser=0)

The series of video tutorials each conclude with coding exercises which could be used or modified to teach the key principals. These resources could also provide extension work for teachers of the Computing Technology Stage 5: Creating games and simulations focus area.

Coded solutions for each are available on the [Replit](https://replit.com/@OOPWorkshop/1petvariablessol" \l "main.py) site and are readily adaptable for students to Predict, Run, Investigate, Modify and Make using the code samples.

The tutorials cover the following:

[Part 1: Familiar ways for structuring data](https://sites.google.com/dltv.vic.edu.au/object-oriented#h.6to3hw701p5x)

[Part 2: Structuring data better](https://sites.google.com/dltv.vic.edu.au/object-oriented#h.zduabumuvtc)

[Part 3: Truly functional objects](https://sites.google.com/dltv.vic.edu.au/object-oriented#h.7k7ga4lf5e7r)

[Part 4: A complete Object-oriented program](https://sites.google.com/dltv.vic.edu.au/object-oriented#h.ckx1yp3ky7qd)

[Part 5: Object-oriented programming in graphical games](https://sites.google.com/dltv.vic.edu.au/object-oriented#h.jq9mt9bg922t).

The guided tutorial to make an RPG game in the Appendix of this document, which accompanies the assessment task, has been sourced and modified from the Raspberry Pi Foundation under the Creative Commons 4.0 license.

# Apply the key features of an object-oriented programming (OOP) language

**Activity 1:** basic introductory coding: turtle racing with objects.

Students write a program using objects with Python's Turtle module to create a turtle race.

1. Create a new Python program and save it as turtlerace.py.
2. Import the Turtle **class**, which is like a blueprint for making a turtle:

from turtle import Turtle

Warning: do not name your file the same as an import, for example, turtle.py, otherwise the code will try to import itself and will not work.

1. Create an instance of a Turtle **object**.

For example: leonardo = Turtle()

As the name of the Turtle **object** is a variable name, it must start with a letter, and it cannot contain any spaces. You are creating a variable in the same way as you usually do, except that the data type of the variable is not an integer or a string, but a Turtle!

Notice that a capital ‘T’ is used when referring to the Turtle **class**. This is because class names usually start with a capital so that they are easily distinguishable from variable names.

Each Turtle **object** is a different **instance** and will need a different name, so that when you give instructions, you can be specific about which object you are giving the instructions to.

1. Tell your Turtle object what it should look like.

Inside the object are **attributes**, which are pieces of data we can define.

The Turtle object has attributes for colour and shape; you can use the .color and .shape methods to customise those attributes:

leonardo.color('blue')

leonardo.shape('turtle')

1. You can also tell the Turtle object what to do by calling other methods.

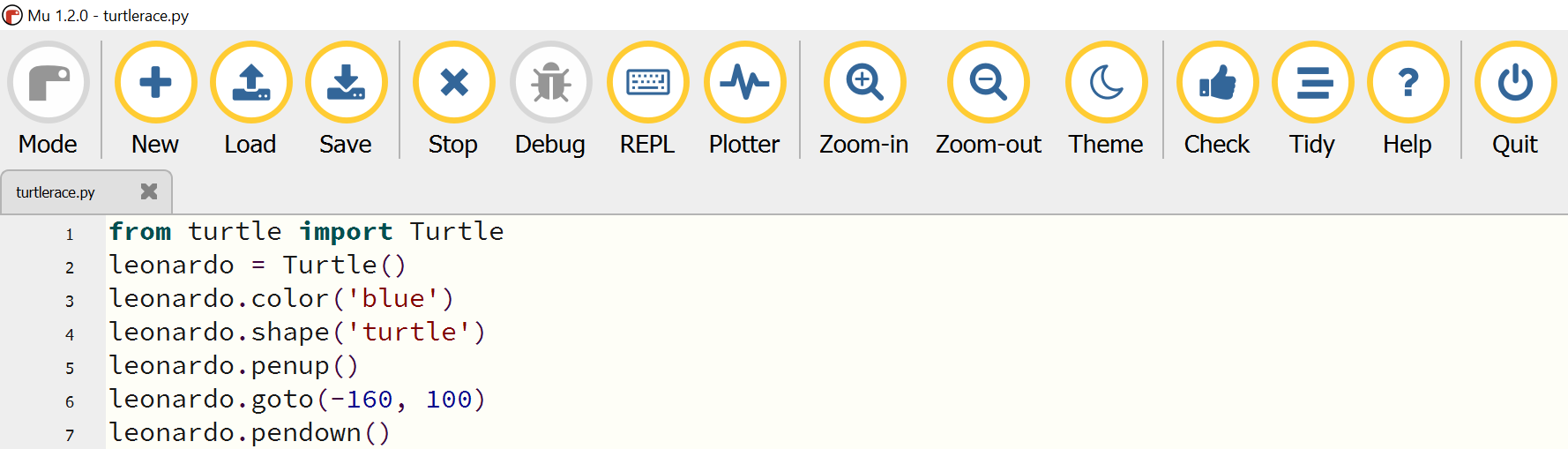
Use the code below to instruct the turtle to stop drawing with penup(), then to move to a location with goto(), and finally to get ready to draw a line with pendown().

Leonardo.penup()

leonardo.goto(-160, 100)

leonardo.pendown()

Figure 1 – turtle race code in Python using Mu



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Teacher note:** use the PRIMM strategy.

P – Predict what will happen

R – Run the code to confirm predictions

I – Investigate what the code does

M – Modify the code

M – Make their own programs

In this activity students are introduced to objects, methods and attributes by identifying their use in Python's Turtle module.

This activity could also be given to the Computing Technology Stage 5 students for the Creating games and simulations focus area. It is provided here since Software Engineering has no prerequisites and students will come to this subject with diverse backgrounds and experience.

1. Create 3 more instances of a Turtle object, each with a different name.

michelangelo = Turtle()

donatello = Turtle()

raphael = Turtle()

If you send all the turtles to the same starting point, they will all be on top of each other.

1. Tell one new turtle to goto(-160, 70), one to goto(-160, 40), and one to goto(-160, 10), for example:

michelangelo.penup()

michelangelo.goto(-160, 70)

michelangelo.pendown()

You can also set a different colour for each turtle if you like, using the .color method as before.

Save and run your code to check that each of your turtle’s positions itself correctly, ready to start the race!

You now need to make the Turtle objects race. Each turtle will move forward by a random number of pixels.

1. After the code to create your 4 Turtle objects, add code, replacing the names (leonardo, michelangelo, donatello, raphel.) with the names of your own Turtle objects:

from random import randint

for movement in range(100):

leonardo.forward(randint(1,5))

michelangelo.forward(randint(1,5))

donatello.forward(randint(1,5))

raphael.forward(randint(1,5))

Just as when you used methods to tell the turtle to penup(), pendown(), and goto(), with this code you are using the forward() method to ask it to move forward a random distance between one and 5 units.

**Note:** the randint function from the random module generates random integers.

Save and run your code and see which turtle wins!

The result should be different each time you run the code.

1. **Note:** depending on the integrated development environment (IDE) you are using your code may execute and then close the window before you have had time to see the output. If so, add this line to the end of your files to prevent this from happening:

input("Press Enter to close")

Experiment with the turtle race program, change some attributes, and create some new Turtle objects.

Visit: <https://projects.raspberrypi.org/en/projects/turtle-race>

Modify your game to include a racetrack and finish line.

**Teacher note:** this activity assists students with understanding that a class is like a blueprint for creating objects, an object has attributes (or data) stored inside it, and methods are used to give it instructions. It is a basic entry point into these concepts and the dot notation. Students will be expected to write their own classes during this course.

**Activity 2:** visit [Object-oriented programming and classes](https://inventwithpython.com/beyond/chapter15.html) and discuss the Real-World Analogy: Filling Out a Form.

[Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645) other suitable analogies and add them in the space below:

|  |
| --- |
| **Sample answer:**  A class is like a blueprint for creating objects, similar to a biscuit cutter: it is a template for all the biscuit objects you make.  You can make as many instances of biscuit objects as you want, and they will all start off from the same template (instantiation).  If you like, you can customise each instance of a biscuit object, perhaps by adding some icing or some sprinkles. But whenever you make biscuits, you use the same biscuit cutter template.  Each object starts off the same, but the attribute values can be changed for each object. |

**Teacher note:** this tutorial will be a useful extension resource: [Object-Oriented Programming (OOP) in Python 3](https://realpython.com/python3-object-oriented-programming/).

# Objects

**Activity 3:** research and respond in the spaces provided.

What is an object?

|  |
| --- |
| **Sample answer:**  Objects are used to model things in code.  An object:   * can represent a physical item, such as a display screen; or a digital unit, such as a bank account or an enemy in a computer game * is a group of data and functions. Such functions are called methods; they are custom functions specifically designed to interact with an object * is created from a class. It is the element created with the properties and methods defined by the class.   One example of an object is a button with the name 'Exit' that closes the program.  Another example from the previous activity was the use of Turtle objects.  Because programmers can define their own objects, anything can be represented using an object. |

**Teacher note: the flipped classroom has students learning in their own time to dedicate class time to hands-on learning and interactive discussion.** For homework, students watch a detailed description of key concepts called [Fundamental Concepts of Object-Oriented Programming (9:15)](https://youtu.be/m_MQYyJpIjg?si=S60zXfZIGu3LHx1X).

In this video the phrase ‘properties’ is used instead of ‘attributes’.

Where may you have seen objects before?

|  |
| --- |
| **Sample answer:**  In Python, everything is an object. String and integer variables, lists, and even functions are individual objects. Students may not have created their own objects or been aware of them though they will have used objects in their Python programs.  This code would create a list of numbers: numbers = [1, 2, 3] numbers and is a list object that contains data and methods. It contains the data 1, 2, 3 and methods which allow you to manipulate this data. The append method is used to add a new item to the end of the list, for example in numbers.append(4), the data within the numbers object would be modified to contain the new number, for example, 1, 2, 3, 4. |

**Teacher note:** for a detailed description teachers could watch [Object Oriented Programming (41:43)](https://youtu.be/-DP1i2ZU9gk?si=1HjxPvglQnfVnHwU).

What is instantiation?

|  |
| --- |
| **Sample answer:**  When a user instantiates something, they are creating an instance of an object.  An instantiated object is given a name and created in memory using the structure described within a class declaration.   In the previous activity the instances of the Turtle objects were named leonardo, michelangelo and so on. |

# Classes

What are classes?

|  |
| --- |
| **Sample answer:**  A class is commonly known as the blueprint or concept. It describes what something is, such as the idea of a button. Classes will exist for different parts of the program.  Many languages have pre-defined classes. All classes define 2 things: attributes and behaviours – or better known as properties and methods. In the previous example the Turtle class (the blueprint for making a turtle) was imported into Python. |

What are attributes?

|  |
| --- |
| **Sample answer:**  An attribute, also known as a property, is a key feature of an object. It may describe any part of an object, such as the height, font, position, or name. In the turtle example the attributes were shape and colour. |

What are methods?

|  |
| --- |
| **Sample answer:**  A method is a procedure associated with a message and an object. The data and behaviour comprise an interface, which specifies how the object may be utilised by any of the various consumers of the object. An example may be that a user can trigger the function associated with a button by clicking on the button. In the turtle example the methods used included penup(), pendown(), goto() and forward(). |

**Activity 4:** Coding

**Teacher note:** this activity introduces classes, objects, attributes and methods to draw shapes. It requires starter code which can be found in the Appendix of this document.

**Step 1**: find the starter code in the Appendix of this document. This code provides the blueprints for paper, triangle, oval, and rectangle objects.

**Step 2**: select all the code and copy into a new Python file\*.

\* Students should note that copying and pasting the code from across multiple pages in this Word document into the Integrated Development and Learning Environment (IDLE) may result in an error when hidden characters are picked up and transferred. To avoid this, students are advised to select and copy code within separate pages before pasting into the IDLE and testing frequently for errors.

**Step 3**: save this new file as shapes.py.

**Step 4**: make a new window in Python.

**Step 5**: import the classes that are needed.

Type: **from shapes import Paper,Rectangle,Oval,Triangle.**

This means import the classes (Paper, Rectangle, Oval and Triangle) from the shapes file that was saved earlier.

Notice that each one of these starts with a capital letter.

**Step 6**: run, the code to see if there are spelling mistakes.

You will be prompted to save the file.

Name the file: **my\_drawing.py** and save.

**Step 7**: if you have any typing mistakes with the class names, you’ll receive an error message. If not it will appear as nothing has happened.

**Step 8**: Drawing

Working in the **my\_drawing.py program:**

The first thing to do is to make a piece of virtual paper to display the rectangle on.

Create an instance of the Paper class.

Type: paper = Paper ()

Note the piece of virtual paper is called ‘paper’, with a small ‘p’ to differentiate it from the class. This is an instance of the Paper class.

Type: rect1 = Rectangle ()

‘rect1’ is an instance of the rectangle class.

**Step 9**: Use a set of methods of the rectangle class to define the attributes of the rectangle.

Type: rect1.set\_color("blue")

This uses the setter method.

**Step 10:** Use the setter method to set the height and width.

Type: rect1.set\_height (100)

rect1.set\_width (200)

**Step 11:** Draw the rectangle on the piece of paper using its draw method.

rect1.draw()

**Step** **12:** Display the piece of paper by taking the instance to calling its display method.

paper.display()

**Step 13**: run the code (you should see a blue rectangle displayed on the paper.

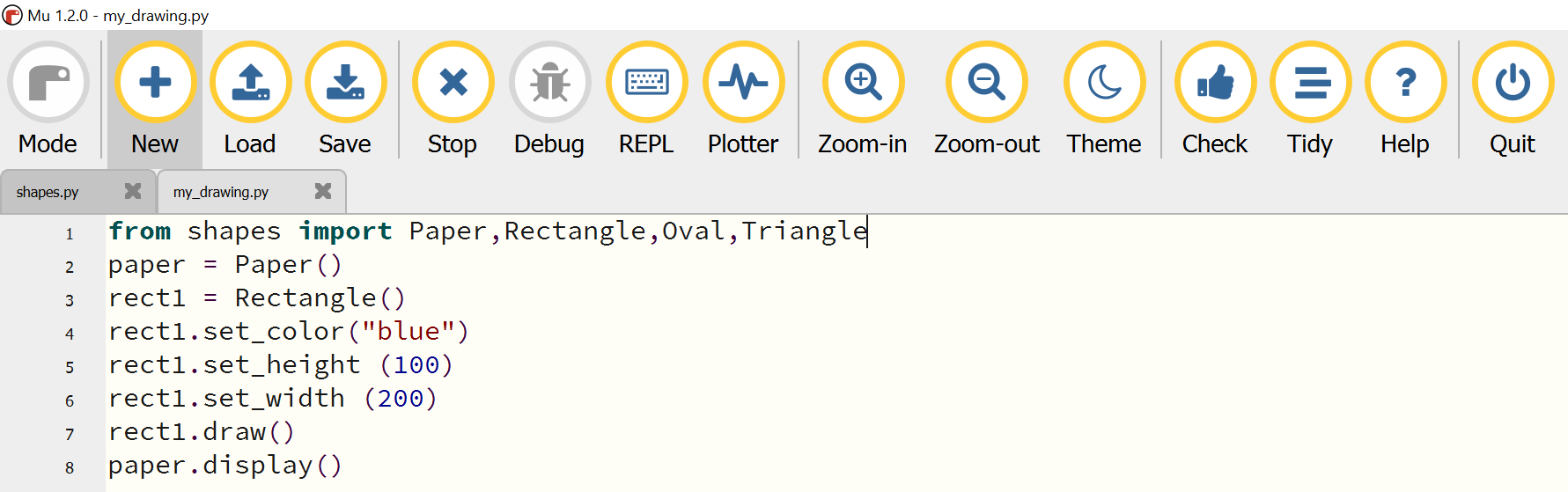
**Step 14**: create a second rectangle that has a different width, height, and colour.

Remember, to create the second rectangle before you display the paper.

**Step 15**: experiment with changing the position of the rectangle using the set-x and the set-y methods

Extension: Create a drawing of a simple house.

Figure 2 – rectangle drawing written in Mu (Python)



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Objects and classes

**Activity 5:** Jigsaw

**Homework:** students watch: [Python OOP Tutorial 1: Classes and Instances (15:23)](https://youtu.be/ZDa-Z5JzLYM?si=SP-_ahqo3nxMhDS9).

Consider object-oriented programming languages representing real life.

In teams choose a real-life object you are familiar with, provide a definition and describe how it could be represented using Objects and Classes. Include its attributes.

|  |
| --- |
| **Sample answers:**  Students have many different pens.  Each pen could be considered an object.  The definition for a pen would be something that contains ink and puts that on paper.  Each pen shares similarities with other pens.  In an object-oriented language, the definition for a type of object is called a class.  The class for a pen object might include an attribute representing colour, and another attribute representing length of pencil. In code, this would be:  Class Pen {  int Length;  string colour;  }  This class outlines all attributes and methods that an object of type pen will have. For example, in code to create a pen write:  Pen myPen1 = New Pen;  Pen myPen2 = New Pen;  In the code example, each line starts off with the class name, pen, then the object’s name myPen1 and myPen2, and then code which states = New Pen.  This is called instantiation. |

**Activity 6:** investigate the following code.

In the space below describe the class Number by referring to each line.

1 Class Number {

2 bool sign;

3 int magnitude;

4 sub Add(m) {

5 magnitude = magnitude + m;

6 if (magnitude < 0) {

7 sign = FALSE;

8 magnitude = magnitude \* (-1);

9 } else {

10 sign = TRUE;

11 }

12 };

13 }

|  |
| --- |
| **Sample answer:**  This code creates a class called Number (line 1).  This class has 2 attributes, called sign and magnitude (lines 2–3).  Sign is a Boolean variable (line 2), and magnitude is an integer (line 3).  The class also defines a method called Add(), which takes one parameter (line 4).  Note that instead of writing BEGIN and END, or IF and ENDIF, curly brackets are used to show a control structure has started or ended and that in this language each line of code ends with a semicolon. |

**Teacher note:** the class Number is just a template. This means that in code, to use these attributes and methods you must write: Number num\_1 = New Number; This is called instantiation. In object-oriented programming languages, a class is first abstractly defined, and an instance of that class is created. This instance is called an object. Abstraction refers to creating a class that defines the methods and attributes objects will have. Instantiation refers to creation of objects, called instances, that have the attributes and methods of that class.

## Encapsulation

**Activity 7:** research and respond in the space provided.

What is encapsulation?

|  |
| --- |
| **Sample answer:**  Encapsulation is used to hide components or values of an object within a class from other objects, which helps to restrict access to the object. This means data inside an object is not modified unexpectedly by external code in a completely different part of the program, written by a different programmer. Also, other programmers need only know what that object’s methods will produce, without needing to know details about the object’s internals in order to use or refer to it in their code. |

**Activity 8:** explain one benefit of using the object-oriented paradigm with respect to software maintenance.

|  |
| --- |
| **Sample answer:**  Due to encapsulation, any objects not affected by changes in requirements do not need to be considered by the programmer.  This allows them to focus specifically on the methods or attributes requiring change. |

## Abstraction

**Activity 9:** research and respond in the space provided.

What is abstraction?

|  |
| --- |
| **Sample answer:**  Abstraction is the process of designing classes, so they are reduced to only their necessary properties and behaviours.  Abstraction can be used with inheritance so that there is a more general class which includes only the characteristics of all people, and then another more specific class.  Abstraction enables the programmer to make use of an object without having to know the detail of its attributes and methods. |

## Inheritance

Students will be familiar with inheritance as something they are passed down from their ancestors or parents. In the context of object-oriented languages, inheritance refers to the ability of a class to access the methods and attributes of a second class, called a parent class. In a real-life example, think of the pens and pencils in your pencil case from the previous examples. Both a pen and a pencil are a method of writing.

This means they might both inherit the parent class WritingObject.

Both pens and pencils have a length (meaning they will share the attribute length) and the ability to write (meaning they both will share the method WriteText()).

However, pens might have a special attribute called penType indicating whether the pen used is a ball-point pen or uses ink, and a pencil might have a special attribute called thickness indicating how thick the pencil’s tip is.

**Activity 10:** [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645#.Y_Kr3SagGrY.link)

Based on the pens and pencils scenario create 3 classes.

Use the space provided to write the parent class.

|  |
| --- |
| **Sample answer:**  01 Class WritingObject () {  02 int length;  03 string Colour;  03 Sub WriteText(string text) {  04 DISPLAY text  05 }  06 }  A **class** called WritingObject (line 01) with the **attribute** length (line 02) and **method** WriteText (line 03).  This is the parent class. Pens and pencils can be created as child classes of this parent class. |

**Activity 11:** use the space provided to write the Pens’ class.

|  |
| --- |
| **Sample answer:**  07 Class Pen () {  08 Inherits WritingObject;  09 bool penType;  10 Sub ExampleUse() {  11 WriteText(“Pen information:”);  12 DISPLAY penType;  13 DISPLAY length;  14 }  15 } |

**Activity 12:** use the space provided to write the pencils’ class.

|  |
| --- |
| **Sample answer:**  16 Class Pencil () {  17 Inherits WritingObject;  18 integer thickness;  19 Sub UseEraser() {  20 }  21 Sub ExampleUse() {  22 WriteText(“Pencil Information:”);  23 DISPLAY thickness;  24 DISPLAY length;  25 }  26 } |

**Teacher note:** answers should show that each child class (pencil and pen) states that they inherit the parent class (line 08, line 17).

This means that they can access methods and attributes defined in the parent class, including the attribute length (line 11, line 22), and method WriteText (line 13, line 24).

The child class Pencil also has an extra method, UseEraser().

**Activity 13:** use the space provided to explain the benefits of inheritance.

|  |
| --- |
| **Sample answer:**  Inheritance has numerous benefits. A parent class can be created with commonly-used functionality. This centralised location for common code reduces complexity and duplication of code, promotes standardised terminology throughout a project, and allows developers to work on separate aspects of a program simultaneously.  If a parent class is commonly used throughout a project, several developers may be tasked with making that class operate as fast and efficiently as possible.  Parent class functionality can then be easily inherited by other classes that expand that functionality. |

**Activity 14:** use the space provided to create pen objects using this class.

|  |
| --- |
| **Sample answer:**  Pen BluePen = New Pen();  BluePen.Colour = Blue;  BluePen.Length = 7;  BluePen.penType = “ball-point”;  Pen RedPen = New Pen ();  RedPen.Colour = Red;  RedPen.Length = 6;  RedPen.penType = “quill”; |

**Activity 15:** use the space provided to create pencil objects using this class.

|  |
| --- |
| **Sample answer:**  Pencil Pencil1 = New Pencil();  Pencil1.Colour = Blue;  Pencil1.Length = 5;  Pencil1.thickness = 14;  Pencil Pencil2 = New Pencil ();  RedPen.Colour = Red;  RedPen.Length = 8;  RedPen.thickness = 15; |

These objects have different values but the same attributes and methods.

If objects of a different class are created, then these will have different attributes and methods.

**Teacher note:** these code samples and questions have been sourced from Dr Thomas Ledger’s *Software Student’s Handbook* with permission.

**Homework:** students watch [Inheritance/Polymorphism in Object Oriented Programming (12:49)](https://youtu.be/C2QfkDcQ5MU?si=EkEQ-Q0fWs9FhDDS).

**Activity 16:** research and respond in the space provided.

What is inheritance?

|  |
| --- |
| **Sample answer:**  Inheritance is a feature of object-oriented programming that allows new classes to reuse, extend, and modify the methods and properties of existing classes. An existing class is sometimes called a parent or superclass. A class that inherits attributes or methods from another class is called the child or subclass. |

## Generalisation

**Activity 17:** Research [generalisation](https://www.coursera.org/lecture/object-oriented-design/1-2-6-generalization-xjMaQ) and respond with examples in the space provided.

What is generalisation?

|  |
| --- |
| **Sample answer:**  In object-oriented programming, generalisation is the technique of extracting the essential characteristics from 2 or more subclasses and then combining them inside a generalised base class.  [For example, in software for banking, the current account, saving account, and credit account as the generalised form of a bank account](https://www.javatpoint.com/uml-class-diagram). |

## Polymorphism

*Polymorphism* refers to subroutines with the same name but different parameters. The word *polymorphism* comes from *poly* meaning ‘multi’, and *morphism* meaning ‘shape’ – so a *polymorph* is something that can have multiple forms.

Two subroutines are said to be *polymorphic* if they:

* exist in the same location with the same name, but
* have different parameters.

For example:

1 Class Process\_Data{

2 Sub Display(int i) {

3 DISPLAY I ;

4 }

5 Sub Display(string i) {

6 DISPLAY I;

7 }

8 }

Here both methods belong to class Process\_Data, and have the same name, Display(). However, the parameters of each subroutine are different types.

One parameter is an integer (line 2) and the other is a string (line 5).

Both methods can be used in code, but which method will run depends on the type of parameter passed.

**Activity 18:** in the space below, write a subroutine based on the class Process\_Data that demonstrates polymorphism.

|  |
| --- |
| **Sample answer:**  1 SUB Demonstration() {  2 Process\_Data example\_object;  2 Int example\_integer = 5;  3 String example\_string = “hello world”;  4 example\_object.Display(example\_integer);  5 example\_object.Display(example\_string);  6 }  Here, the example\_object executes using 2 separate methods of the same name (lines 4–5). |

**Activity 19:** in the space below, explain what would occur if you attempted this in an imperative language?

|  |
| --- |
| **Sample answer:**  In an *imperative* language, this would cause a *type-mismatch error*, because there could only be one subroutine with a parameter of one type. |

**Activity 20:** in the space below respond to this question.

What is polymorphism?

|  |
| --- |
| **Sample answer:**  Polymorphism is when the programmer has the ability to create a method that can be performed at different times, with different sets of data, and at any time during the execution of the program.  Programmers have the ability through polymorphism to process objects from different sub classes.  For example, the button click method is a concept that can be applied repeatedly as needed. |

**Activity 21:** consider this code fragment and answer on the following page.

1 class Operators {

2

3 operator(string str1, string str2) {

4 string s = str1 + str2

5 Return s

6 }

7

8 operator(int a, int b) {

9 int c = a + b

10 Return c

11 }

12 }

13

14 class Main {

15 Operators obj = new Operators()

16 obj.operator(2, 3)

17 obj.operator(“joe”, “now”)

18 }

Using the space below, describe how polymorphism is used in the code provided.

Make reference to line numbers in your response.

|  |
| --- |
| **Sample answer:**  The methods on lines 3 and 8 have the same name, but act differently when given different parameters. Calling the method ‘operator’ on line 16 executes different logic from calling the method ‘operator’ on line 17 as the parameters have different data types. Therefore, the processing of the method ‘operator’ changes based on the parameters provided when the method is called. |

Using the space below, respond to the following scenario:

A new method operator (int n, string str1) in the class Operators returns a single string with str1 repeated n times.

For example, operator(3, “joe”) returns “joejoejoe”.

Write the code for this new method.

|  |
| --- |
| **Sample answer:**  operator (int n, string str1) {  outStr = “”  FOR count = 1 TO n  outStr = outStr + str1  NEXT count  Return outStr  } |

**Activity 22:** match the terms below to the appropriate definition:

object, class, encapsulation, abstraction, attribute, inheritance, instantiation, generalisation, method and polymorphism.

|  |  |
| --- | --- |
| Term | Definition |
| inheritance | Allows classes to reuse the methods and properties of existing classes. |
| encapsulation | The hiding of information that prevents unnecessary code from being shown. |
| instantiation | The creation of an object. |
| polymorphism | Allows objects to use the same method in different ways. |
| attribute | A property assigned to an object. |
| method | An action assigned to an object. |
| class | The blueprint, or description of the object. |
| abstraction | The process of designing a class so only the necessary code is visible. |
| object | Is instantiated from the class. |
| generalisation | Where common properties are formulated as general concepts or claims. |

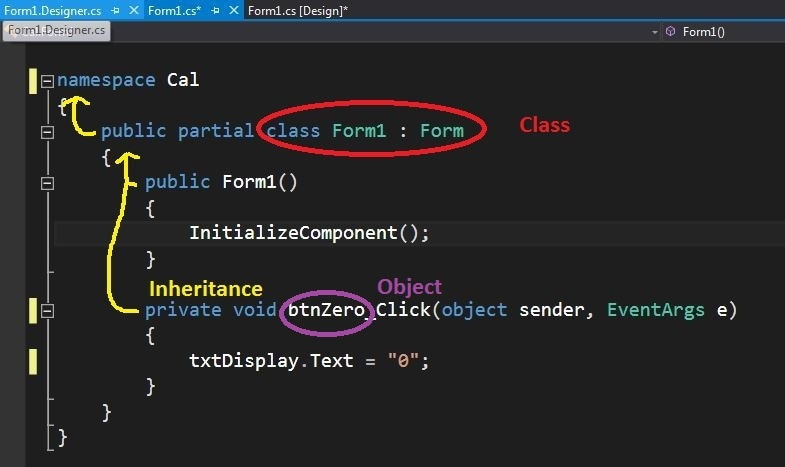
## OOP in C#

Using an object-oriented approach, involves considering each component of the problem as a self-contained unit. Each unit can remember things (data/attributes) and do things (methods/operations).  There are a number of terms in the object-oriented paradigm, and these are defined here with some examples from the C# code and designer interface in Visual Studio.

**Activity 23:** study the following screen shots.

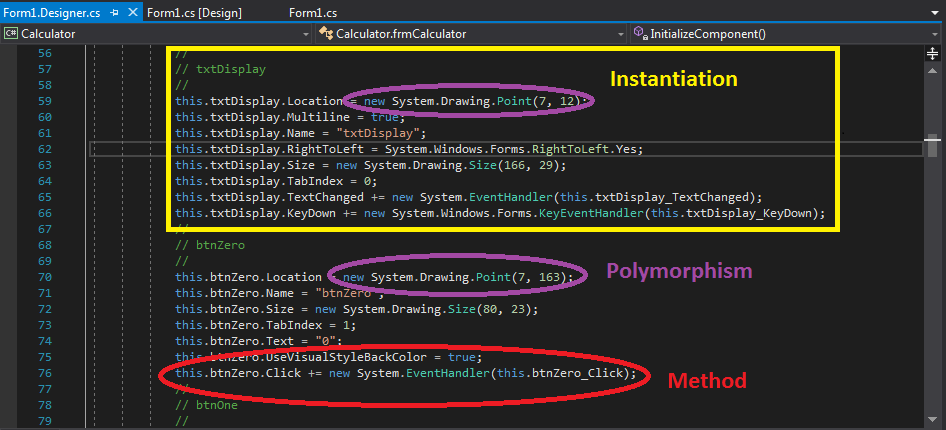
Using [C# OOP (Object-Oriented Programming)](https://www.w3schools.com/cs/cs_oop.php), describe in the space below what the code will do by referring to the coloured annotations.

Figure 3 – visual studio C# interface screen shot



|  |
| --- |
| **Sample answer:**  The namespace keyword is used to declare a scope that contains a set of related objects. Namespace is used to organise code elements and to create globally unique types.  A partial class is a special feature of C#. It provides the ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a **partial** keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.  The class here is the Form circled in red.  An object, in this case a button circled in purple, inherits (in yellow) from that class.  The term public makes itself accessible to all the program's participants from wherever. It provides the fewest restrictions on visibility.  Private: only users of a similar class have access to the secret content. It offers the lowest visibility levels among other access specifiers. |

Figure 4 – visual studio C# interface screen shot



|  |
| --- |
| **Sample answer:**  Instantiation is used to create an object from a class (instantiate) shown here in the yellow box. Polymorphism means ‘many forms’ and occurs when many classes are related to each other by inheritance. Polymorphism uses those methods to perform different tasks. In this example new System.Drawing.Point is used for both display of text and of a button. A method (circled in red) is a block of code which only runs when it is called. Data (known as parameters) can be passed into a method. Methods are used to perform certain actions and are also known as functions.  Why use methods? To reuse code: define the code once, and use it many times. |

**Teacher note:** while Python is confirmed to be a language tested in the digital examination, students should be exposed to and have experience with a number of programming languages.

# Compare procedural programming with OOP

Before discussing the object-oriented paradigm, it is important to develop an understanding of procedural programming.

**Homework:** students watch [Software Development Tutorial – What is object-oriented language? (5:18)](https://www.youtube.com/watch?v=SS-9y0H3Si8)

## Procedural programming

Procedural programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure. Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out. During a program’s execution, any given procedure might be called at any point, including by other procedures or itself. The procedural paradigm is sometimes referred to as the imperative paradigm.

**Languages used in procedural programming**

FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

## Object-oriented programming

Object-oriented programming can be defined as a programming model which is based upon the concept of objects. Objects contain data in the form of attributes and code in the form of methods.

In object-oriented programming, computer programs are designed using the concept of objects that interact with the real world.

Object-oriented programming languages are various, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

**Languages used in object-oriented programming**

Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala, Rust.

## Definitions

**Teacher note:** internet forums will offer a host of interpretations on these definitions.

**Imperative programming**

In the early days of programming in assembly, code would have many GOTOs. In this programming paradigm, the entire program is a single algorithm or has complete functionality and is written linearly, that is, step-by-step.

This is called an imperative paradigm and they were the first human-readable languages developed, and therefore are used in a wide variety of settings.

**Structured and modular programming**

These terms are used interchangeably but there are subtle differences. As higher-level languages ‘evolved’ programming became a hierarchy of functions and many functions at lower levels could be re-used.

**Structured programming**

Describes any programming when functionality is divided into units (block structures) including for loops, while loops, if ... then and so on. These units (functions) can be re-used.

**Modular programming**

Refers to when chunks of code are created to form packages that are general purpose and re-usable. These modules can then be compiled together.

Modular programs which are not structured and structured programs which are not modular are rarely seen. The technical definition is subtly different.

Object-oriented programming is a form of structured programming.

All function based code which is structured code but NOT object oriented is called procedural programming.

## Summary

Put simply structured code where functions (or procedures) dominate over data is called procedural.

Class and object-based representation is called object oriented.

Both are (by definition) modular.

Imperative, procedural and structured are not mutually exclusive properties, they just focus on different aspects of the logic.

## Historical context

Solving different types of problems

Characteristics of the imperative paradigm are not always suited to solving problems.

The imperative paradigm encourages programmers to think of a big problem that can be broken down into smaller parts. Each of the parts must fit into the whole.

**An analogy of a giant clockwork planetarium**

* Inside are thousands of cogs each turning to help the ball move
* Remove a cog and the entire planetarium will at some stage fail.

Like in the imperative paradigm the components are relatively inseparable, and the entire problem must be tackled all at once. Every subroutine and variable added to the software solution must consider every other subroutine and variable developed beforehand.

This may be a quick way to develop code in the short term, though consider a much larger piece of software that for example manages a stock exchange.

This software must take orders from clients, match orders for stock with parties willing to sell it, and transfer electronically the ownership. The software must also store and distribute all relevant information about a company, including in almost real-time the last sale price for stock.

If this solution was to be developed in the imperative paradigm, it would be huge and cumbersome. An error in any component, including in a superficial component such as the component which distributes information, may cause an entire system to fail.

This problem of complexity has been a leading factor in the development of the object-oriented paradigm.

The object-oriented paradigm allows code to be encapsulated in entities called objects. Instead of the sphere filled with cogs, the object-oriented paradigm analogy could be a car.

Each part of the car (battery, steering, propulsion) is a separate component that exchanges fuel, electricity, or data. This means that a failure of a single component (such as the car radio) doesn’t cause the entire car to fail. Consider if a plane was unable to take-off because of a problem with the in-flight entertainment.

**Activity 24:** complete the table below.

|  |  |  |
| --- | --- | --- |
| Paradigm | Description | Example languages |
| Procedural | Structured code where functions (or procedures) dominate over data. | FORTRAN, COBOL, BASIC |
| Object-oriented | Using an object-oriented approach involves considering each component of the problem as a self-contained unit. Each unit can remember things (data/attributes) and do things (methods/operations). | Java, C++, C#, Python, Rust |

**Activity 25:** study the [OOP and non-OOP versions of tic tac toe.](https://autbor.com/compareoop/#diff)

In the space below, note the differences between these 2 versions.

|  |
| --- |
| **Sample answer:**  The main obvious difference is where the game board is set up.  In the non-OOP version functions are called to make the board.  In the OOP version a class Tic-Tac-Toe board is constructed with an instance of the object. |

## Imperative and OOP

Object-oriented code has different syntax, using curly brackets to show control structures, and with lines ending in semi-colons. An example showing this difference, as well as the use of an object is provided in the table below.

|  |  |
| --- | --- |
| Imperative | Object-oriented |
| int N | Number.value = 3; |
| N = 3 | Number.Add(5); |
| AddNumbers(N, 5) |  |

This table shows how a subroutine in the imperative language would be programmed in an object-oriented language.

In this example, an integer called N stores the value 3, and then the integer 5 is added to it. In the object-oriented example, there is an object called Number.

Each attribute or method is shown following the object’s name and a period (full-stop). When the object Number has an attribute called value, it is written as Number.value.

In the object-oriented language, the method Add() is a part of the Number.

This subroutine is shown following the object’s name and a period.

Objects are written like records, but objects are much more powerful.

In the example below, 2 objects are made that are numbers, and then their values are added:

Number1.value = 4;

Number2.value = 5;

Number2.Add(Number1.value);

# Object-oriented programming

**Activity 26:** complete the cloze passage using the words below.

*method*, *attribute*, contained, interact , C++, program

Many programming languages are *object-oriented*, including Java, , and VB.NET. Object-oriented languages focus on making a program from separate components, called *objects*.

Each *object* can \_\_\_\_\_\_\_\_\_\_\_\_ with the other *objects* in the \_\_\_\_\_\_\_\_\_\_\_\_.

Data and the ability to process data is also \_\_\_\_\_\_\_\_\_\_\_\_ within each *object*.

When data is attached to an object it is called an \_\_\_\_\_\_\_\_\_\_\_\_, and when a subroutine is attached to an object, it is called a \_\_\_\_\_\_\_\_\_\_\_\_.

|  |
| --- |
| **Sample answer:**  Many programming languages are object-oriented, including Java, C++, and VB.NET.  Object-oriented languages focus on making a program from separate components, called objects.  Each object can interact with the other objects in the program.  Data and the ability to process data is also contained within each object.  When data is attached to an object it is called an attribute, and when a subroutine is attached to an object, it is called a method. |

**Activity 27:** complete the cloze passage using the words below.

data, duplicated, objects*,* accessed, *encapsulate*, numerous

There are \_\_\_\_\_\_\_\_\_\_\_\_ benefits to this approach.

Objects \_\_\_\_\_\_\_\_\_\_\_\_ data, so any changes to the code within an object will have no effect on the code of other objects.

The code within \_\_\_\_\_\_\_\_\_\_\_\_ can be modified individually and are easily \_\_\_\_\_\_\_\_\_\_\_\_.

Objects are designed to only expose and receive relevant \_\_\_\_\_\_\_\_\_\_\_\_.

The internal processes that alter data are not able to be \_\_\_\_\_\_\_\_\_\_\_\_ by other objects.

|  |
| --- |
| **Sample answer:**  There are numerous benefits to this approach.  Objects encapsulate data, so any changes to the code within an object will have no effect on the code of other objects.  The code within objects can be modified individually and are easily duplicated.  Objects are designed to only expose and receive relevant data.  The internal processes that alter data are not able to be accessed by other objects. |

**Teacher note:** getting students (and especially computing students) to read can be challenging. Cloze passages provide a motivational aspect to the reading process and assist with engagement.

# Use data flow diagrams, structure charts and class diagrams to represent a system

**Activity 28:** in the space below, describe system modelling.

|  |
| --- |
| **Sample answer:**  System modelling is the process of developing abstract models that represent a system.  These abstract models are often in the form of diagrams.  Several diagrams can be used to represent various systems.  These are typically constructed when designing the system by the developer and aid in understanding the nature of the solution to be created. |

## Data flow diagrams

**Activity 29**: in the space below, describe what a data flow diagram is and what it does.

|  |
| --- |
| **Sample answer:**  A data flow diagram is used to represent an information system as a number of processes that together form the single process of a context diagram.  The source of data, its flow between processes and its destination along with data generated by the system is represented. |

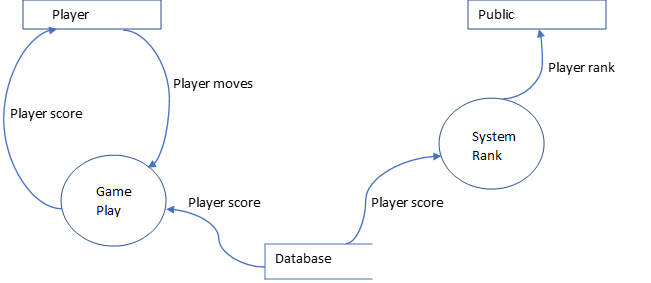
**Activity 30:** using the [Software Engineering Course Specifications](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/teaching-and-learning) complete the following table.

|  |  |  |
| --- | --- | --- |
| Concept | Data flow diagram | Symbol |
| External entities | Up to 3 | Shape |
| Processes | Up to 5 (recommended 2–4) | Shape |
| Data flow | Multi directional | Shape |
| Data storage | Up to 2 | Data storage is represented as a rectangle |

**Note:** never connect one symbol to an identical symbol (i.e. process to process, or entity to entity).A process must somehow change the data going into it.

Study the data flow diagram for an online multiplayer video game.

Figure 5 – data flow diagram for an online multiplayer video game



**Activity 31:** in the space below, describe what a data flow diagram is and what it does.

|  |
| --- |
| **Sample answer:**  In this system the player makes the selection of moves as a normal part of the game.  They are presented with a score based on the normal processes.  The scores are also uploaded to a storage device (likely a cloud server) that then processes the scores, ranks the player and assigns them to the position on the public leader board. |

**Activity 32:** in the space below, draw a data flow diagram for a supermarket barcode reader being operated at a self-serve checkout.

|  |
| --- |
| **Sample answer:**  A data flow diagram depicting processes in a supermarket barcode reader |

## Structure charts

**Activity 33:** in the space below, describe what a structure chart is and what it does.

|  |
| --- |
| **Sample answer:**  Structure charts or diagrams allow the representation of a system broken down into its separate subtasks or processes.  The relationship between each of these processes should also be evident from the diagram.  Rectangles are used to represent tasks, with lines used to show the connections between tasks.  The intended purpose of a structure diagram is to show the overall design of the functions in the program. It provides the programmer with a big picture of the design solution to be coded, for example the functions of the program and how they interact. It is used in the planning and design stage and is intended for the programmer. |

**Activity 34:** in the space below, describe how a structure chart should be read.

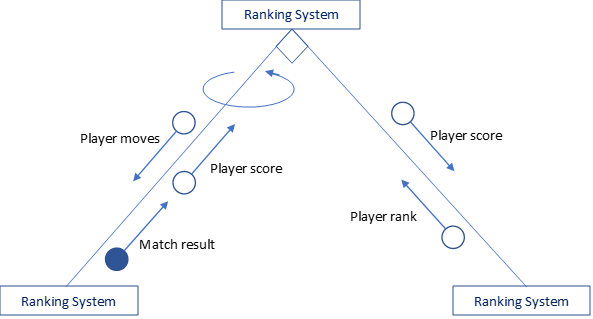
|  |
| --- |
| **Sample answer:**  Structure charts are read from top to bottom, with component subtasks on successively lower levels, and from left to right to show the order of execution of tasks at the same level.  The charts are used to graphically depict a modular design of a program and shows how the program has been partitioned into smaller more manageable modules. |

**Activity 35:** using the [Software Engineering Course Specifications](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/teaching-and-learning) complete the following table.

|  |  |  |
| --- | --- | --- |
| Component | Definition | Diagram |
| Modules | Represent sub programs, functions and processes. | Shape |
| Data movement (variable) | Data movement of a variable between functions. | variable passed between functions |
| Data movement (flag) | Data movement of a Boolean variable, or a control parameter. | A boolean variable (flag) passed between functions |
| Repetition | Execution of a particular task multiple times. | Repetition |
| Decision | An optional execution of a task. | Shape |

**Activity 36:** study the structure chart for the online multiplayer video games ranking system.

Figure 6 – structure chart for the online multiplayer video games ranking system



In the space below describe what the structure chart demonstrates.

|  |
| --- |
| **Sample answer:**  The structure chart demonstrates how 3 modules could work together passing variables and control parameters between each other, in the video game ranking system described above. |

### Steps to create a structure diagram

1. Brainstorm all the functions that need to be coded. This involves:

* analysing the problem to determine what is needed
* use top-down approach to decompose (break) the problem into the sub functions
* indenting can be used to show sub functions.

1. Construct the diagram. This involves:

* drawing a box-rectangle for each function.  Use verbs (for example, a doing word such as get, open, load, add, display, edit, delete, print, select) in each rectangle
* drawing the interconnecting lines between the boxes, showing how the functions are linked to each other
* adding symbols to the diagram
* loops (for example for when a function is called multiple times)
* case (diamond) -(choice of functions)
* passing of parameters (open – variable, closed – control)

1. Provide explanations and further details to explain your diagram.

This could include providing a short description of each function and explanations of other aspects of your design that someone else reading your diagram would need to understand your design.

**Checklist when making a structure chart**

Have you:

1. listed all the ‘sub functions’ for the system?
2. interconnected them correctly to relate the correct relationship between subs?
3. used the correct symbols to relate passing variables, loops, decisions?

**Activity 37:** consider the following algorithm for the Login subroutine.

|  |
| --- |
| **Sample answer:**  BEGIN Login  valid= False  LoadUserNameAndPasswordFile  Get username from User  WHILE username is not equal to 9999 AND valid= False  Get password from User  found = False  FindUserNameAndPassword(username, password, found)  IF found= True THEN  PRINT "Valid details entered"  valid= True  ELSE  PRINT "Please try again"  Get username from User  END IF  END WHILE  END Login |

In the space below, explain the purpose of valid and found in the above algorithm.

|  |
| --- |
| **Sample answer:**  Valid is used as a flag to determine when the username and password has been entered correctly and to stop the program from asking the user to enter their username and password again.  Found is used to determine if the username entered by the user is in the array of records and whether the password entered by the user matches the stored password for that username.  It is initially set to False and then findUserNameAndPassword sets it to True if the username is found and the password matches. |

**Activity 38:** in the space below, draw a structure chart for the Login algorithm above. Ensure you have accounted for the data movement for at least 3 sets of data.

|  |
| --- |
| **Sample answer:**  A structure chart. |

**Activity 39:** When constructing a structure chart, which is the conventional symbol used to show a control parameter? Provide your response in the space below.

|  |
| --- |
| **Sample answer:**  A control parameter is represented by a shaded circle and arrow. |

**Activity 40:** complete the table on the advantages and disadvantages of a structure chart.

|  |  |
| --- | --- |
| Advantages of a structure chart | Disadvantages of a structure chart |
| **Sample answers:**   * Simplified overview of the movement of data between modules * Shows the modularisation of a system * Helps for strategic planning of resources | **Sample answers:**   * Does not show algorithm or demonstrate how a program solves a problem * Can be complicated to follow * Does not distinguish specific data types for variables (int, str, and so on…) |

## Class diagrams

**Activity 41:** in the space below, describe what a class diagram is and what it does.

|  |
| --- |
| **Sample answer:**  The class diagram is the main building block of object-oriented modelling.  It is used for general conceptual modelling of the structure of the application.  The attributes and methods are articulated clearly as well as their relationship to a parent class.  It is represented with boxes that contain 3 compartments:   * name of the class * attributes of the class * methods of the class. |

**Activity 42:** Using the [Software Engineering Course Specifications](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) and continuing the video game battle example, draw a class diagram that describes the various characters and character traits.

Use the space on the following page to complete your diagram.

|  |
| --- |
| **Sample answer:**  Video game class diagram  Class diagram that describes the various characters and character traits in a video game. |

**Activity 43:** Examine the code below constructed in the object-oriented paradigm and answer the questions on the following pages.

|  |
| --- |
| **Sample answer:**  class Point {  private -  point\_no: integer  x\_coordinate: double  y\_coordinate: double  public -  getPoint(point\_no):  return x\_coordinate and y\_coordinate  }  sub-class Circle {  is a Point  private -  circle\_no: integer  radius: double  public -  getArea(circle\_no):  return Math.PI\*radius\*radius  }  sub-class Rectangle {  is a Point  private -  rectangle\_no: integer  height: double  width: double  public -  getArea(rectangle\_no):  return height\*width  } |

**Activity 44:** in the space below, outline a reason why a procedural language would be difficult to apply to the above program.

|  |
| --- |
| **Sample answer:**  The OOP language requires the detail for a choice of 2 variables – either a circle or rectangle. Imperative would need to be strictly structured to ignore one section of code and move onto the other – it would not have the ability to reuse the same fragment of code as easily. The polymorphism in the OOP makes it far easier. |

**Activity 45:** in the space below, identify an example of each of the following within the above code.

|  |
| --- |
| **Sample answer:**  Object: Point  **Attribute**: x-coordinate  **Method:** getArea(circle\_no): |

**Activity 46:** in the space below, justify the use of encapsulation in the code for the Point class.

|  |
| --- |
| **Sample answer:**  The encapsulation means the length, width and diameter of the objects are hidden from the user. Only allowing the points to be created means the distances between them cannot be altered – rather they are calculated by the program. |

**Activity 47:** in the space below, describe how polymorphism is applied to this fragment of code

|  |
| --- |
| **Sample answer:**  The 2 sub-classes are defined to demonstrate polymorphism, where the method getArea is interpreted differently depending on the sub-class in which it is used. |

**Activity 48:** construct a class diagram based on the code above.

|  |
| --- |
| **Sample answer:**  **A class diagram.** |

**Activity 49:** [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645#.YvsdDPklJfI.link)

Create a list of steps and a checklist to follow when creating a class diagram.

|  |
| --- |
| **Sample answer:**   1. **Identify the primary objects of the system.** 2. **Determine how each of the classes or objects are related to one another.** 3. **Create classes for each of the objects.** 4. **Add attributes for those classes.** 5. **Add methods for those classes.** 6. **Enter the diagram name and description.** |

**Teacher note**: students should be encouraged to make class diagrams for all OOP projects they create.

# Describe the process of design used to develop code in an OOP language

Consider the following scenario and answer the questions.

A school library requires a system that allows books to be borrowed and returned.

Books can belong to different categories such as fiction, reference, and non-fiction. Borrowers can be students, staff and parents.

Books can be on loan, available or overdue.

****The system will be developed using the object-oriented programming (OOP).

**Activity 50:** complete the table to identify an example of a class for this system and provide a subclass and an attribute for this class.

|  |  |
| --- | --- |
| Category | Example |
| Class | Sample answer:  Book |
| Subclass | Sample answer:  Fiction |
| Attribute | Sample answer:  Author |

****The system is to be modified so it can notify borrowers of overdue books and calculate fines.

**Activity 51:** in the space below, describe how OOP methods can be used to add the necessary changes.

|  |
| --- |
| **Sample answer:**  OOP methods can be used to update attributes.  Here a method can be used to compare the due date attribute from the Book class to the current date.  When the current date is greater than the due date a fine can be calculated and a flag set.  A different method can be used to set a flag in the connected instance of the Person class indicating that a notification is to be sent out.  It also adds the fine to the balance attribute for that person. |

The system is to be expanded to suggest to borrowers any books that they might like.

**Activity 52:** in the space below, explain how artificial intelligence (AI) can assist in providing this additional functionality.

|  |
| --- |
| **Sample answer:**  The system stores details of every book borrowed by every borrower.  An AI system can track borrowing habits by looking for patterns in this stored data.  For example, when a person borrows a certain book, the system stores details of the books that are borrowed at the same time.  The AI system can find patterns of borrowing using this collected data and therefore recommend similar books. |

## Task definition

A task definition, sometimes referred to as a problem statement, is a short, clear explanation of an issue or challenge that sums up what you want to change. It helps you, team members, and other stakeholders to focus on the problem, why it's important, and who it impacts. A good task definition should create awareness and stimulate creative thinking.

**Activity 53:** in the space below, write a task definition for the school library system.

|  |
| --- |
| **Sample answer:**  **Students should rewrite the scenario with the key features of:** ‘**who, what and how’ identified.** |

## Top-down and bottom-up

In programming, a **top-down approach**, also referred to as stepwise design, is a complex algorithm broken down into smaller units referred to as modules. By breaking down a complex algorithm into smaller units, a top-down approach reduces the complications that usually arise while designing algorithms. The procedural programming languages such as Fortran, COBOL and C follow a top-down approach.

**Bottom-up approach** entails designing an algorithm by starting at the very basic level and building it up towards the complex level.  In the approach the modules are designed independently and are then integrated together to form a complete algorithmic design. Modules are developed, tested, and debugged first, and then the next modules are developed, tested, and debugged prior to integrating them to build a concrete solution. The same process is continued until all modules have been completed. Object-oriented programming languages like C++, Java, C#, Perl and Python follow the bottom-up approach.

**Activity 54:** in the space below justify a top-down or bottom-up approach to developing the school library system.

|  |
| --- |
| **Sample answer:**  **Arguments from either side can be made, although for this scenario students should identify the benefits of a bottom-up approach, creating a solution with modules.** |

## Facade pattern

A Facade pattern hides the complexities of the system and provides an interface to the client from where the client can access the system.

Dividing a system into subsystems helps reduce complexity.

To minimise the communication and dependencies between subsystems a facade object provides a single, simplified interface to the more general facilities of a subsystem.

The facade pattern is used when:

* a simple interface is required to access a complex system
* a system is very complex or difficult to understand
* an entry point is needed to each level of layered software
* the abstractions and implementations of a subsystem are tightly coupled.

Students may research a better understanding of [facade patterns.](https://medium.com/@andreaspoyias/design-patterns-a-quick-guide-to-facade-pattern-16e3d2f1bfb6)

Students should investigate the use of [facade patterns in C#.](https://methodpoet.com/facade-pattern/#:~:text=The%20primary%20intent%20of%20Facade%20is)

**Activity 55:** in the space below, sketch a facade pattern for the school library system.

|  |
| --- |
| **Sample answer:**  Sample sketch of a facade pattern for the school library system.**A facade pattern simplifies the interaction with a complex system by providing a simple and easy to use interface. To use a library system that has many classes and methods for different purposes such as searching, borrowing and returning books, a facade class can be used that wraps these functionalities and only exposes the methods that are needed.** |

## Agility

The agility of a program is described as its ability to adapt to unspecified or changing needs of the client.

****The object-oriented paradigm enhances the programmer’s ability to make these changes without modifying the front-end design of the program.

**Activity 56:** in the space below, suggest how agility could be used for the school library system.

|  |
| --- |
| **Sample answer:**  After consultation with the client, it could be decided (for example) that:   * borrowers can receive alerts for when their books on hold are available or when their checked-out items are due * borrowers manage their reading history by tracking the books they have read and leaving reviews or ratings. |

## Assess the effectiveness of programming code developed to implement an algorithm

**Activity 57:** study the code below.

Answer on the following page.

1 Class Main() {

2 Class\_Player Players (4);

3 FOR (int i; i <= 4; i++) {

4 Players (i) = New Class\_Player;

5 Players (i).Cash = 50000;

6 Players (i).num\_properties = 5;

7 DISPLAY “Net Worth: “ & Players (i).NetWorth();

8 }

9 }

In the space below, describe the purpose of the code above by referring to the line numbers.

|  |
| --- |
| **Sample answer:**  This code shows that objects can be stored in an array.  In this example, 4 objects are created from the abstracted class Class\_Player (line 2). These objects are stored in the array called Players().  Each object is instantiated (line 4).  Each object in the array has an attribute called Cash and attribute called num\_properties (line 5-6), which are set to specific values.  Finally, each object has a method called NetWorth() that is executed (line 7).  The method will do the same operation for each object.  However, it may return a different result, because each object may hold different data.  In object-orientated languages, loops are written as:  FOR (Int i = 0; i<= 4; i++) {  some statements  }  This is in the format FOR (variable name and first value; condition to keep looping; how much to increase the variable by each loop).  Each term is separated by a semicolon.  In the example above, a variable is declared called i with a first value of 0 (Int i = 0).  The loop continues iterating while the condition I <= 4 is true.  Finally, at the end of each iteration, the variable i is increased by a value of 1.  This is shown by the double plus i++, which means the same as i = i + 1.  Other increases, such as i= i + 2, would also be valid. |

**Activity 58:** in the space below, assess the effectiveness of the programming code developed to implement an algorithm.

|  |
| --- |
| **Sample answer:**  The For loop statement sets initial values very efficiently. |

## Investigate how OOP languages handle message-passing between objects

**Activity 59:** describe message-passing in the space provided below.

|  |
| --- |
| **Sample answers:**  Message-passing is the way for objects to communicate within a program.  Objects are self-sufficient units that possess their own properties and methods.  When objects send and receive, they access the data or carry out an action by calling methods. |

**Activity 60:** consider the following scenario and answer the questions in Activities 61–63.

The fragment of code on the following pages are from a theatre booking system.

The system allows tickets to be sold for a variety of events at a number of different theatres.

In the future, the developers are hoping to incorporate changes to the ticketing system, such as allowing a greater variety of discount types.

**Theatre booking system**

1 class TICKET {

2 private –

3 Ticket\_no: integer

4 Event\_id: string

5 Date\_of\_event: date

6 Theatre\_id: string

7 Basic\_cost: float

8 Event\_name: string

9 public –

10 GetEvent\_name()

11 RETURN Event\_name

12 END GetEvent\_name

13 }

14 class TICKETTYPE {

15 is a TICKET

16 private –

17 Type\_of\_ticket: string

18 Discount: real

19 Evidence\_sighted: Boolean ‘REM For example, set to TRUE if driver licence is seen

20 public –

21 GetDiscount()

22 RETURN Discount

23 END GetDiscount

24 GetEvent\_name()

25 RETURN Event\_name, Type\_of\_ticket

26 END GetEvent\_name

27 }

**Activity 61:** in the space below, distinguish between inheritance and polymorphism.

Support your answer with specific examples from this system.

|  |
| --- |
| **Sample answer:**  Inheritance allows the definition of a subclass such as the TICKETTYPE have the same properties as the super class TICKET with additional properties as required, such as discounted tickets.  On the other hand, polymorphism, while still a characteristic of OOP, allows the same method to process data differently, depending on the circumstances.  Although we call the method GetDiscount(), the logic will depend on the specific type of discount available for the ticket being purchased.  Answers could include: GetEvent\_name() occurs in both classes but the logic is different. |

A discount can only be provided if relevant evidence is sighted.

**Activity 62:** in the space below write the code to achieve this include line numbers to show where the code is to be added.

|  |
| --- |
| **Sample answer:**  IF Evidence\_sighted is FALSE THEN  Discount = 0  END IF |

**Activity 63**: in the space below, describe how message-passing may work between objects in the ticketing system

|  |
| --- |
| **Sample answer:**  ‘The components of the ticketing system: interface, database and ticketing processor could be modelled as objects and communicate with each other via message-passing. |

# Explain code optimisation in software engineering

**Activity 64:** research code optimisation and in the space below explain its use in software engineering.

|  |
| --- |
| **Sample answer:**  Code optimisation is the process of transforming a piece of code to make it more efficient, either in terms of time or space, without changing the output. The goal of a programmer will ultimately be to write as little code as possible while still meeting the project requirements.  Some students play ‘code golf’ where they try and accomplish the same task in the fewest lines of code.  Optimisation techniques and strategies to enhance its speed, memory, reliability and compatibility.  These strategies include reducing the number and size of variables, using faster algorithms and data structures, avoiding unnecessary calculations and operations, caching results, parallelising tasks, compressing code, using native code or libraries, and adapting to different platforms.  Careful not to over-optimise your code or sacrifice readability, maintainability or security for performance. |

**Activity 65:** in the space below, describe why optimising code is important?

|  |
| --- |
| **Sample answer:**  Code optimisation is important because it improves the performance, efficiency, and overall quality of the source code.  Optimised code has faster execution speed, utilizes memory efficiently, and gives better performance.  It is an essential step in the software engineering process as it ensures that the final product is free from bugs, runs smoothly, and is able to handle large amounts of data. Although the optimisation process may require additional time and money, the result is a better experience, not just for developers but also for end users. |

## Outline the features of OOP that support collaborative code development

The most important component of collaboration within object-oriented programming is the ability for developers to understand how each other operates.

### Consistency

It is important to develop good practices early on when writing and developing code. By remaining consistent in terms of structure, variable identification, formatting, interface design and use of control structures, a developer can make it easier for another developer to understand their work.

When creating variable names, try to develop clear explicit terms. It should be easy to guess what the purpose of a variable may be based on the name it is given – it should be intrinsic. A common technique is to construct variables using camelCase style. camelCase is written in all lower case but requires the first letter of each subsequent word to be capitalised. For example:

|  |  |
| --- | --- |
| Poor variable names | Good variable names |
| cc | camelCode |
| Mean | schoolMean |
| ACC.ID | accountFirstName |
| R | redDrinkBottle |

### Code commenting

By code commenting, referred to as internal documentation, a developer can provide clear non-executable statements that describe what a particular piece of code is designed to do. This benefits all developers by making the code more maintainable.

### Version control

Throughout the development of a project, it is important to create copies or backups of the entire project. Each iteration of the project should be documented with dates, progress level and goals for the future of the development.

Version control is done for several reasons. It:

* ensures copies are made in the event of an unanticipated catastrophic failure
* keeps programmers accountable for the development progress
* ensures the development is meeting milestones
* allows mistakes to be identified and backtracked without the need to restart development.

### Feedback

****Programmers provide feedback to each other for multiple reasons such as improving code efficiency, creating more ideal solutions, supporting each other and building up respect in the workplace.

**Activity 66:** research, list and describe the features of the OOP paradigm that support collaborative code development.

|  |
| --- |
| **Sample answer:**  **Classes and objects**  A class is a template that consists of data members or variables and functions and defines the attributes and methods for a group of objects. An object is an instance of a class, and each object has its values for the different attributes present in its class. This feature allows multiple developers to work on different parts of the program simultaneously, as they can create their own classes and objects.  **Abstraction**  Abstraction is the process of removing some characteristics from something to reduce it to a smaller set. This allows developers to hide the implementation details of an object from the user, making it easier to use and enabling developers to work on different parts of the program without worrying about how other parts are implemented.  **Inheritance**  Inheritance is a mechanism that allows a new class to be based on an existing class, inheriting all its properties and methods. This feature enables developers to reuse code from existing classes, reducing the amount of code they need to write.  **Polymorphism**  Polymorphism is the ability of an object to take on many forms. In OOP, polymorphism allows developers to write code that can work with objects of different classes, as long as they have a common interface. This feature enables developers to write more generic code that can be reused across different parts of the program.  **Encapsulation**  Encapsulation is the process of hiding an object’s internal state from the outside world, allowing only certain operations to be performed on it. This feature enables developers to work on different parts of the program without worrying about how other parts are implemented.  **Message-passing**  Message-passing is a mechanism that allows objects to communicate with each other by sending messages. This feature enables developers to write more modular code that can be reused across different parts of the program. |

# Programming in OOP

**Teacher note: in this section of the course students will be creating a computer game as outlined in sample Assessment task 2.**

**This will enable students to meet the syllabus content for this focus area and provide an engaging project which could inform work for their major project.**

**This game is loosely based on the historic classic** [Hunt The Wumpus](https://en.wikipedia.org/wiki/Hunt_the_Wumpus) **and provides opportunities to compare procedural and OOP paradigms. This project addresses the syllabus content points for this section and reinforces concepts delivered throughout the unit.**

**Code for this game is included in the Appendix of this document.** Students should note that copying and pasting the code from across multiple pages in this word document into the IDLE may result in an error when hidden characters are picked up and transferred. To avoid this, students are advised to select and copy code within separate pages before pasting into the IDLE and testing frequently for errors.

**Note: this task could also be modified by simplification for delivery to a Computing Technology Stage 5 class studying the Creating games and simulations focus area.**

**It could also be extended to include a graphical user interface and considered for a major project.**

A computer game consists of players controlling characters that move through a fantasy world, collecting items, performing tasks and interacting with other characters.

**Challenge 1:** in the space below, explain why object-oriented programming languages are particularly suited to the development of such a computer game.

|  |
| --- |
| **Sample answer:**  Since the computer game consists of various characters, these characters can be defined as objects.  These characters or objects can be given common characteristics and they may inherit their attributes from a parent class.  These characters or objects can also interact with each other in specified ways through defined methods. |

# Create a text-based adventure game

A text-based adventure game is an interactive invented world described in text.

It can be filled with different spaces, items, obstacles or anything your imagination allows.

The player interacts with the world by typing commands and the game describes the result of the player's commands.

You are to create a text-based adventure game and use methods and attributes to interact with caves in the game.

In this game you will move around by using the commands north, south, east, and west. Each time you move, you enter a new cave and a description is shown.

An example of a cave in your game might be a grotto, described like this:

The grotto. A small cave with ancient graffiti on every wall.

The cavern is north.

The dungeon is west.

**Teacher note:** this adventure game is loosely based on Hunt the Wumpus and set underground in caves. Teachers could (time permitting) allow student choice in designing scenarios from other genres including, for example, historic, science fiction, crime, romance, sports and so on. The caves would then be changed to other spaces (like rooms, islands or planets) and other characters like princes and princesses, astronauts and aliens, gangsters and police or footballers and teams.

**Challenge 2:** **attributes** are the pieces of information stored within an **object**, just like a collection of variables.

In the space below, describe the attributes you think a cave might have.

|  |
| --- |
| **Sample answer:**  Every cave should have a name (for example 'cavern’, ‘grotto’ or 'dungeon'), so will add an attribute called name. |

**Challenge 3:** **methods** are the ways in which you interact with an object.

In the space below, describe the methods you think might be useful for interacting with a cave.

|  |
| --- |
| **Sample answer:**  You need to be able to display the name of a cave when the player enters it, so you might write a get\_cave() method. |

The following code samples were written using [Mu – a simple Python editor.](https://codewith.mu/)

Figure 7 – the coding interface of MU



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Teacher note: students may prefer** [Wing Python](https://wingware.com/downloads)**,** [Visual Studio Code](https://code.visualstudio.com/Download) **with a Python extension or** [Python](https://www.python.org/). **Many online Python editors are available. Teachers will need to determine how much support their students need including whether to provide code samples with discussion around the following worksheet material. To guide students through this project step-by-step will take some time and teachers will need to balance project completion and student motivation with learning key concepts within this syllabus.**

## Part 1 – setting the scene

**Step 1**: open a new Python file. Save it as **cave.py**.

**Step 2**: create your own **class**, which will be a **blueprint** for the caves in your text-based adventure game.

To create a class give it the name: **class Cave:**

Your class is called Cave because it will represent the concept of a cave.

The class name starts with a capital letter.

**Challenge 4**:In the space below explain why this is important?

|  |
| --- |
| **Sample answer:**  This helps to distinguish between class names and variables. |

**Step 3**: create a constructor and define attributes.

Add a constructor to your class.

**Challenge 5**:In the space below explain what a constructor is?

|  |
| --- |
| **Sample answer:**  This is a special method to tell Python how to create an object of this class, and it is always called \_\_init\_\_ with a double underscore on each side of 'init'.  If this special method name is not right constructor will not work! |

****Complete the following instructions in your **cave.py** file.

Indent the cursor by pressing the tab key to tell Python that the code you are about to write is part of the Cave class.

**Step 4**: add the code below to define the constructor method:

def \_\_init\_\_(self):

**Challenge 6: In the space below,** explain what ‘init’ means.

|  |
| --- |
| **Sample answer:**  Here, ‘init' stands for ‘initialise', as a constructor initialises (that is, creates) an object. |

**Step 5**: add attributes for your cave to the constructor.

The cave should have a name (for example, a cavern, a grotto, or a dungeon).

You will need to store a description of the cave to provide some atmosphere — the dungeon could be dark and dusty, for instance.

Add these attributes to your constructor method like this:

def \_\_init\_\_(self):

self.name = None

self.description = None

**Challenge 7: In the space below, explain what** this code means?

|  |
| --- |
| **Sample answers:**  Attributes within the object are referred to in the format self.name\_of\_attribute.  This tells Python that you are referring to a piece of data within the object; self means 'this object'.  Setting the attribute values to None means that they will start off with no value.  Sometimes you will want to allow people to set the values of these parameters when they use your class to create an object. |

**Step 6**: add a parameter to the constructor called cave\_name by altering the existing code like this:

def \_\_init\_\_(self, cave\_name):

This means that when you create an object, you must provide a cave name.

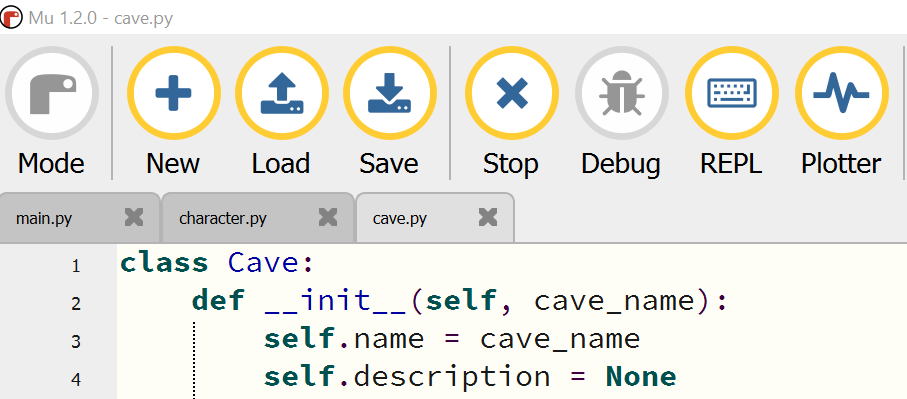
Change the code inside the constructor to tell Python to set the value of the attribute self.name to the cave\_name provided, by altering the existing code like this:

def \_\_init\_\_(self, cave\_name):

self.name = cave\_name

Your Cave class should now look like this:

Figure 8 – class Cave constructor shown in Mu



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 7**: Instantiating your own object

Test your code out by creating an object of the Cave class you have written.

Create a new Python file called **main.py** and save it in the **same folder** as the file **cave.py**.

In the main.py file, add the following line of code:

from cave import Cave

**Challenge 8:In the space below,** explain what this code does.

|  |
| --- |
| **Sample answer:**  This command looks for a file in the same folder called cave.py, and looks inside that file for a class called Cave (note the upper case C); this is the class you just wrote.  The Cave class is then made available for use inside the main.py file. |

(If you did not save your cave code as cave.py, this will not work. In this case, you will need to rename the file.)

**Step 8**: instantiate (create) a Cave object by executing the code in your main.py file.

Add the following code in the next empty line in your main.py file:

**cavern = Cave("cavern")**

This stores the object in a variable called 'cavern' so that you can refer to it later.

Save and run your program.

**Challenge 9:In the space below,** explain why it appears that nothing has happened.

|  |
| --- |
| **Sample answer:**  The Cave object is created, but it can't be seen or interacted with, because there were no methods when the Cave class was created. |

**Teacher note: so far students have:**

* **created a class called Cave and a constructor method with attributes**
  + - * **instantiated an object from the Cave class.**

### Classes and objects

Building a house starts with drawing up the plans.

**Challenge 10:**In the space below, describe how plans for a house are like a class for an object.

|  |
| --- |
| **Sample answer:**  A class is like a blueprint or a design plan for a house. It contains details about the make-up of the house such as the doors, windows, floors, and roof. When the design is built, a house is created. In this analogy, the built house is the object. You can build many houses using the same design, and you can also create many objects from a class.  An object is also called an instance of a class, and the process of creating this is instantiation. |

In the main part of the game, the line below creates a new Cave object with an attribute name set to cavern.

Then the object is assigned to a variable called cavern:

cavern = Cave("cavern")

When the cavern object is instantiated, the Cave class constructor method (\_\_init\_\_) is called.

def \_\_init\_\_(self, cave\_name):

self.name = cave\_name

To fully understand how this works, you need to know how the self-parameter works in the constructor.

The self-parameter automatically receives a reference to the object invoking the method.

In this case, the self refers to the object cavern that was instantiated.

By using self, a method can invoke the object and access the attributes and methods of that object. In the constructor method, the parameter self is used to set the name attribute of the cavern object to the value of the cave\_name parameter.

self.name = cave\_name

**Step 1**: create objects for the other places like a grotto and dungeon so that you have a total of 3 caves in the game.

Open your **main.py** file.

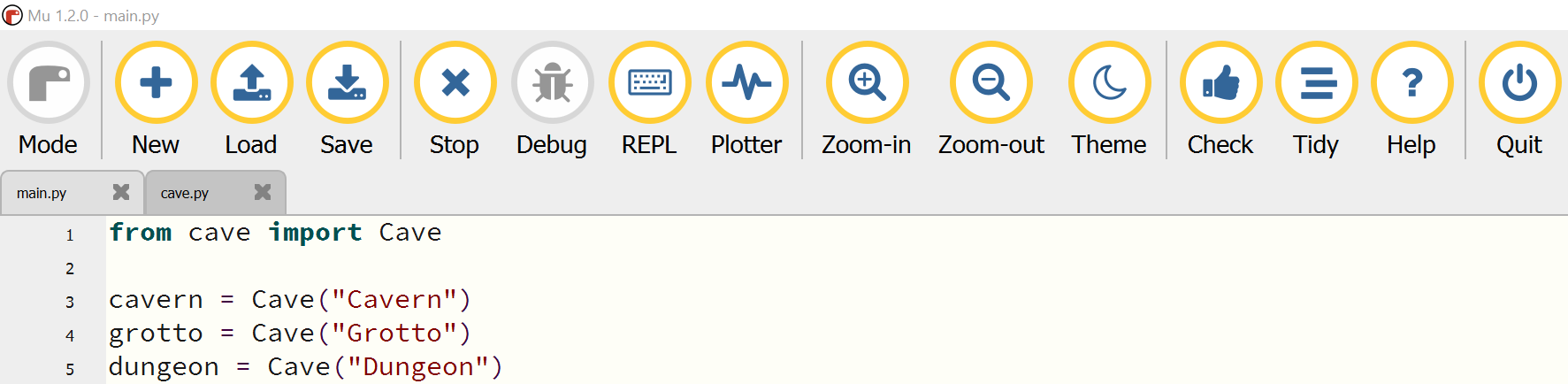
Add the following line of code to create a grotto:

grotto = Cave("grotto")

Add another line of code to create a dungeon:

dungeon = Cave("Dungeon")

Figure 9 – creating cave objects in the main.py file



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

To interact with the cave object, you need to create some getters and setters.

**Challenge 11: In the space below,** explain what getters and setters are.

|  |
| --- |
| **Sample answer:**  These are **methods** that **get** and **set** the values of the object's attributes. |

**Step 2**: add a getter and setter method for the description attribute of your Cave class.

They should be added to the **cave.py** file, after the constructor method.

class Cave:

def \_\_init\_\_(self, cave\_name):

self.name = cave\_name

self.description= None

#Here is a method to get the description of the cave:

def get\_description(self):

return self.description

#Here is a method to set the description of the cave:

def set\_description(self, cave\_description):

 self.description = cave\_description

**Challenge 12:In the space below,** explain how to define methods in a class?

|  |
| --- |
| **Sample answer:**  When defining methods within a class, you first need to specify the self-parameter – the object itself – followed by any data to pass in, just like used in the constructor. |

### Using getter and setter methods

**Step 1**: use the set\_description method just created to give the cavern object a description.

In the **main.py** file add this line of code below the line where the object has been instantiated:

cavern.set\_description("A damp and dirty cave.")

Figure 10 – in the main.py file adding the set method

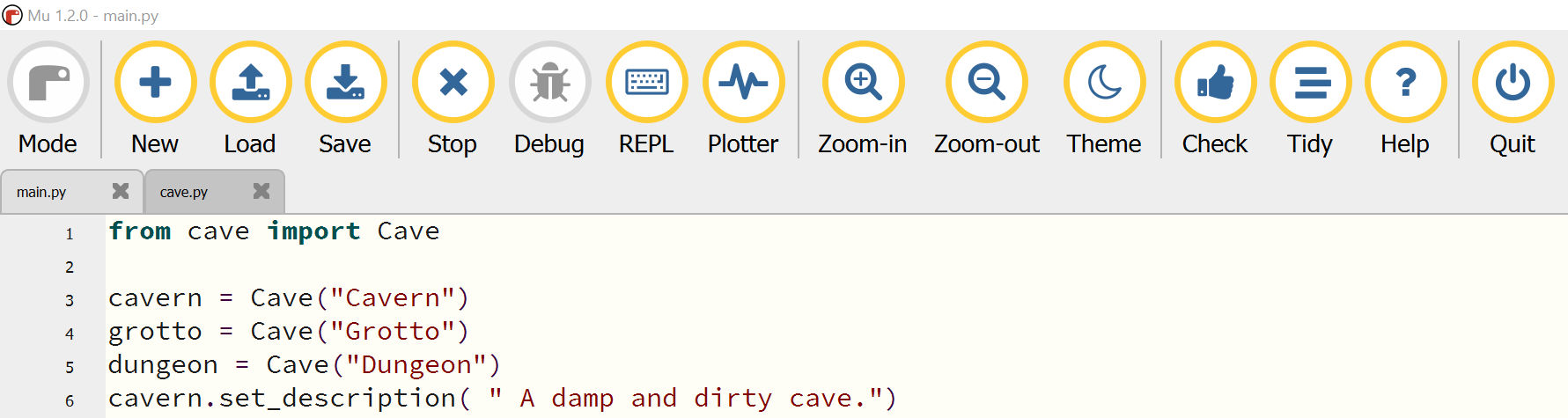
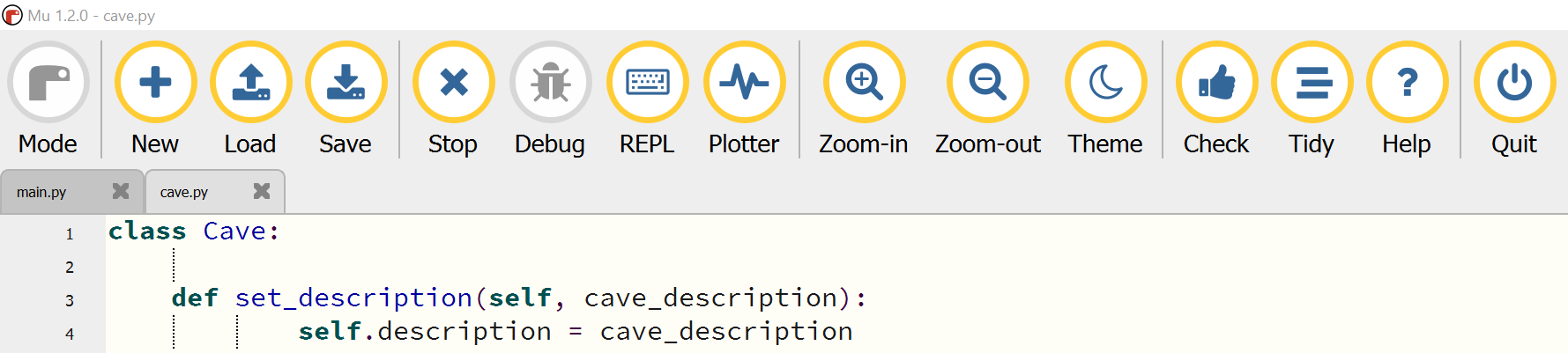


Figure 11 – in the cave.py file where the Cave class has the description set



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Notice that the method you wrote had 2 arguments (self and cave\_description).

However, in the code above only the cave description was provided.

The self-argument is always given first when a method is written, but when using a method, it does not have to give this argument to it. It is automatically passed to the method when using Python.

The get\_description method created earlier can also be used.

**Step 2**: add this line at the bottom of the **main.py file**:

cavern.get\_description()

Save and run your program.

Shouldn’t the description of the cavern cave appear on the screen?

**Challenge 13**:In the space below, explain why nothing appears.

|  |
| --- |
| **Sample answer:**  This is because the get\_description() method returns the description, rather than printing it. This is useful if you want to do something else with the description, such as concatenate it with some other text, but if you want to display the description, you need to use the print command. |

**Step 3**: instead of amending the get\_description() method, use a different approach.

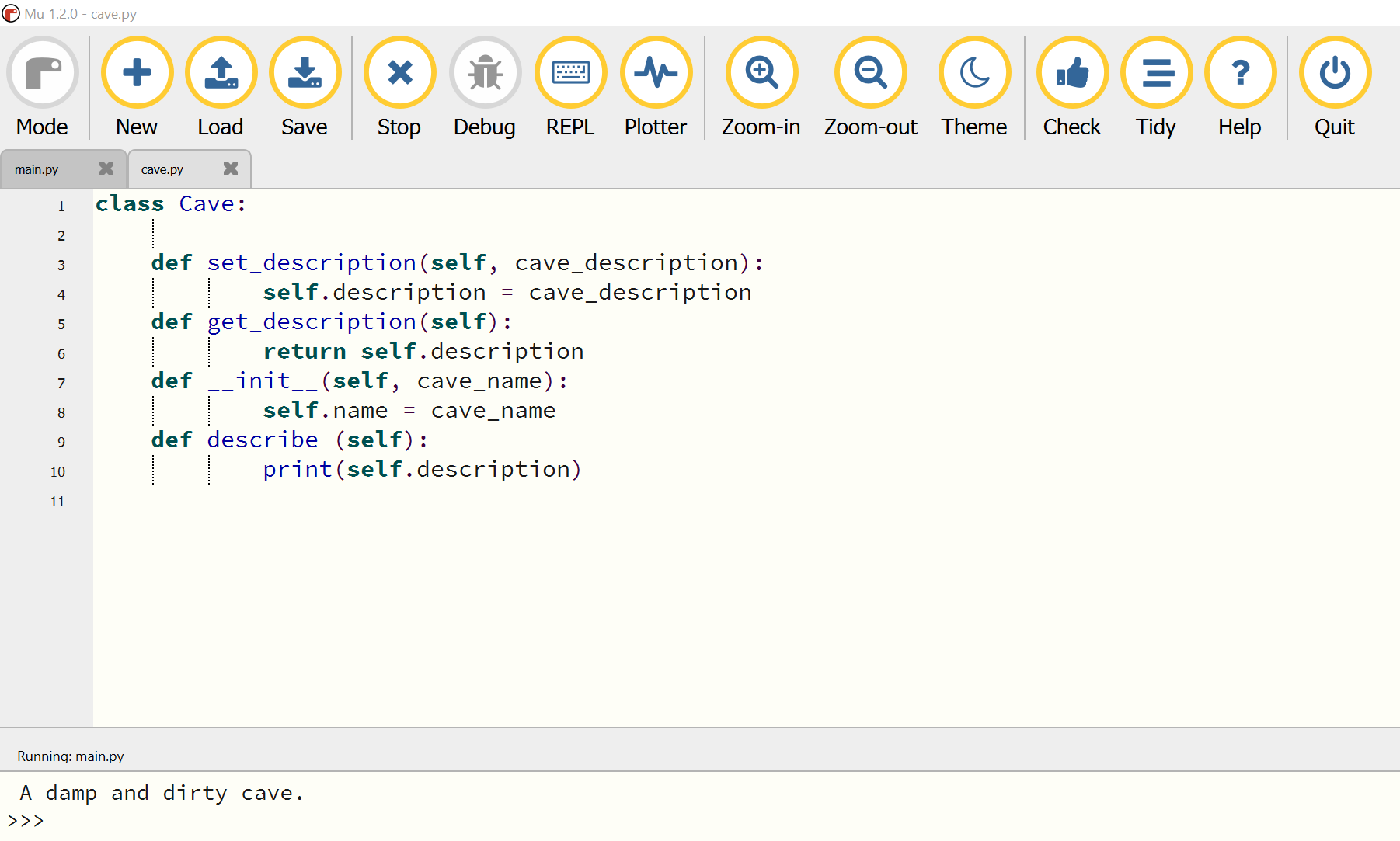
In **cave.py** add an additional method called describe(), which will print out the object's description when it is called.

def describe(self):

print( self.description )

**Step 4**: in main.py, delete the code cavern.get\_description() and add cavern.describe() on the same line, then run it to see the description appear.

Figure 12 – set and get methods that provide descriptions of the cave



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 5**: Creating getters and setters

Add a getter method called get\_name and a setter method called set\_name to your class, to allow someone to get and set the attribute name in the same way.

**Teacher note: Python properties**

**Getters and setters are not strictly needed in Python. The attributes of the object can be accessed directly (for example, you could just print( cavern.description )).**

**Including getters and setters in the class is good programming. This is because they are useful tools for ensuring that people using the code do so in an expected manner. Using getters and setters minimises the risk of unexpected errors.**

**Python also has a more advanced way of specifying getters and setters for attributes using the @property decorator. This is considered more pythonic though this method will not be covered until later in the project. This will simplify and avoid teaching a syntax that is specific to Python.**

### Linking the caves

**Step 1**: in the Cave class, locate the \_\_init\_\_ constructor, and add a new attribute called linked\_caves.

def \_\_init\_\_(self, cave\_name):

...

self.linked\_caves = {}

The linked\_caves = {} code creates an empty **dictionary**; it's empty because at the moment, the cave isn't linked to any other caves.

**Step 2**: add a method to link caves together.

This new method is added below the other methods:

class Cave:

...

def describe(self):

print( self.description )

# Add link\_cave method here

Add the link\_cave method:

def link\_cave(self, cave\_to\_link, direction):

self.linked\_caves[direction] = cave\_to\_link

**Challenge 14**: How many parameters does this method take? Answer in the space below.

|  |
| --- |
| **Sample answer:**  This method takes 3 parameters: the object itself (which you can ignore when you use the method), the cave object to link to, and the relative direction of this object. |

**Challenge 15** :In the space below, sketch a diagram of how the caves will be laid out.

|  |
| --- |
| **Sample answer:**  Compass showing North, and the words cavern, grotto, dungeon. |

**Step 3**: set up the caves in the **main.py** file.

Create the additional caves shown in the diagram above.

Set descriptions for all of your cave objects:

from cave import Cave

cavern = Cave("Cavern")

cavern.set\_description("A dank and dirty cave ")

dungeon = Cave("Dungeon")

dungeon.set\_description("A large cave with a rack")

grotto = Cave("Grotto")

grotto.set\_description("A small cave with ancient graffiti")

**Step 4**: link the caves together.

The dungeon is to the south of the cavern, so link\_cave method on the cavern object in the main.py file:

cavern.link\_cave(dungeon, "south")

grotto.link\_cave(dungeon, "east")

dungeon.link\_cave(grotto, "west")

**Teacher note:** **see inside the dictionary.**

**To see what's inside the dictionary, add this line of code inside the link\_cave method in cave.py to display its contents:**

**print( self.name + " linked caves :" + repr(self.linked\_caves) )**

**class Cave:**

**...**

**def link\_cave(self, cave\_to\_link, direction):**

**self.linked\_caves[direction] = cave\_to\_link**

**# Add the code here (make sure it is indented)**

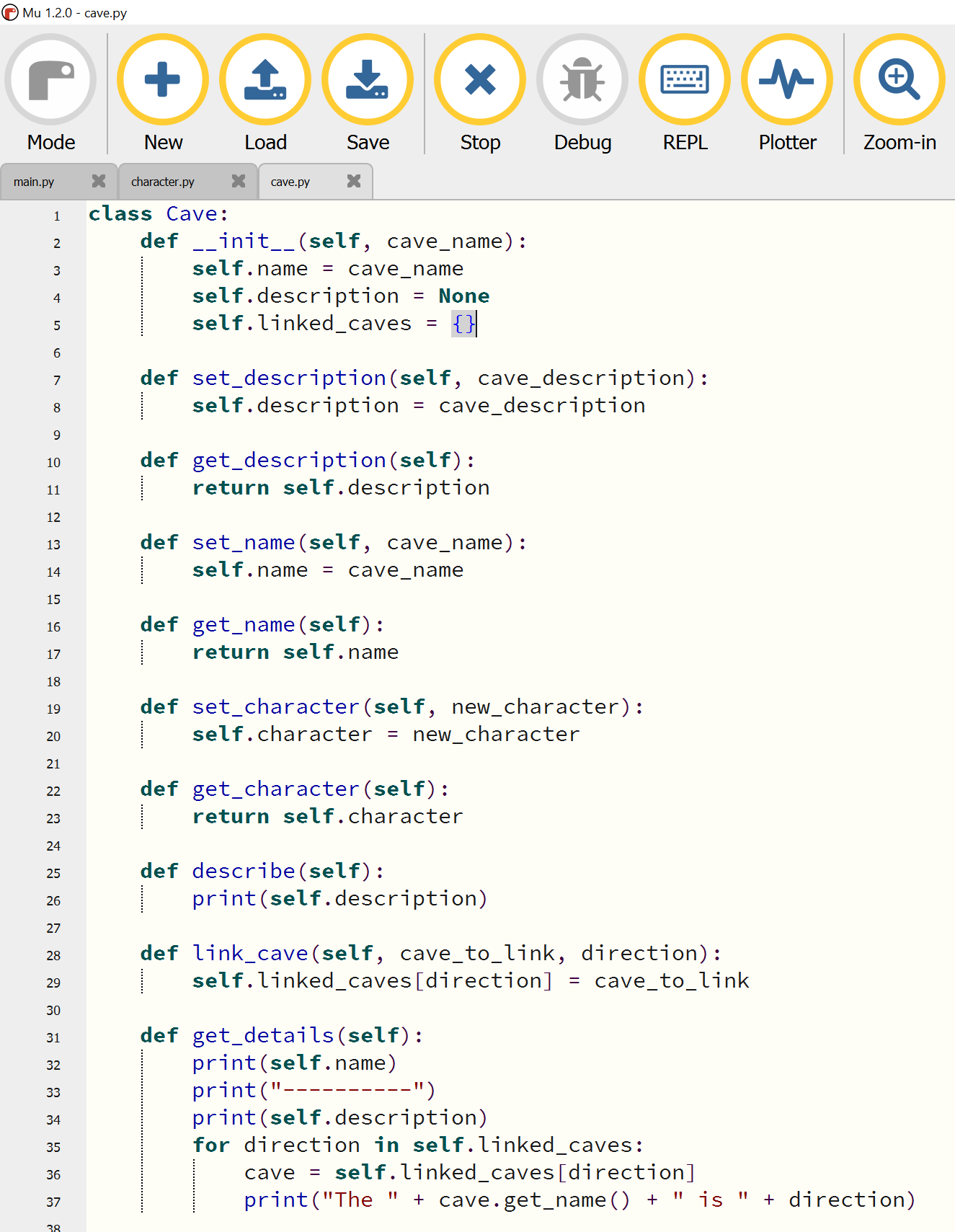
**Run the main.py code and something similar to this, will shows that there is a Cave object in the specified direction:**

**Cavern linked caves :{'south': <cave.Cave object at 0x03A22770>}**

**This code is not necessary for the game just to show how the dictionary gets built up. Once you have seen how it works, comment that line out by putting a # at its start.**

The Cave class should now look similar to the code below, with getter and setter methods for name, and methods to describe the cave and link it to other caves.

Figure 13 – methods to describe the caves and links to other caves



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 5**: create the links between caves.

Use the link\_cave method in main.py to link the caves together in the game so that it matches your sketch above.

**Challenge 16:** In the space below, describe what this snippet code will do.

cavern.link\_cave(dungeon, "south")

|  |
| --- |
| **Sample answer:**  This code links from the cavern to the dungeon. |

There is no link back from the dungeon to the cavern, so the player will be stuck there forever!

**Challenge 17:** In the space below, describe how can this be fixed.

|  |
| --- |
| **Sample answer:**  Create a link in the opposite direction.  dungeon.link\_cave(cavern, "north") |

The cave you are linking from is the object you call the method on, and the cave you are linking to is the object you pass into the method.

**Teacher note: students may design many more caves (or rooms, planets or spaces) and recognise that the original** [Hunt the Wumpus](https://en.wikipedia.org/wiki/Hunt_the_Wumpus) **made a feature of the cave and tunnel structure. This game is simplified to demonstrate the key concepts.**

Displaying the linked caves.

When entering a cave in the game, the game displays a description of that cave to the player, for example:

The dungeon.

A large cave with a rack.

The grotto is west.

The cavern is north.

**Step 6**: add a new method to the Cave class that will display all the caves linked to the current cave object.

The new method is indented, just like all the other methods.

def get\_details(self):

for direction in self.linked\_caves:

cave = self.linked\_caves[direction]

print( "The " + cave.get\_name() + " is " + direction)

**Challenge 18**:In the space below, describe how this method worked with reference to the control structures used:

|  |
| --- |
| **Sample answer:**  This method uses a ‘for’ loop to iterate through the dictionary self.linked\_caves and, for every defined direction, prints out that direction and the name of the cave in that direction. |

**Teacher note: this task addresses the content: design and implement computer programs involving branching, iteration and functions in an OOP language for an identified need or opportunity.**

**Step 7**: Test

In the **main.py** file at the bottom of the script, call this method on the dungeon object, then run the code to see the 2 caves linked to the dungeon.

dungeon.get\_details()

Add the code to the get\_details() method so that it also prints out the name and description of the current cave.

Refer to the current cave as self inside the method.

Check that the get\_details() method works for any cave object by calling it on the cavern and grotto as well.

Figure 14 –get methods for the caves

A close up of words written in Python.



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Teacher note: the get\_details code is for testing and will be deleted later in the project. It addresses content on the use of stubs and applies methodologies to test and evaluate code.**

### Move around the caves

**Step 1**: add a move method to your Cave class.

**Challenge 19**:In the space below, describe how a move method would work?

|  |
| --- |
| **Sample answer:**  The move method should have a parameter for the direction in which the player would like to move. If the direction is one of the directions linked to, the method returns the cave object that is in that direction. If there is no cave in the dictionary in that direction, the method returns self; that is, the player is linked back to the cave they were already in. |

****Go to the **cave.py** file.

Add:

def move(self, direction):

if direction in self.linked\_caves:

return self.linked\_caves[direction]

else:

print("You can't go that way")

return self

Figure 15 – method to move around the caves

A screen shot of Python code in Mu.



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 2**: in the main.py **remove** any code still left from your testing of the get\_details() method.

cavern.get\_details() # remove

dungeon.get\_details() # remove

grotto.get\_details() # remove

**Step 3**: add code at the bottom of the script to create a **loop**, letting the player move between caves.

current\_cave = cavern

while True:

print("\n")

current\_cave.get\_details()

command = input("> ")

current\_cave = current\_cave.move(command)

Save and run your program.

Figure 16 – main.py in Mu with descriptions and link

A screen shot of Python code in Mu


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Type in some directions (for example ‘south’) to move between caves.

Test the directions that won't work, to see whether the game handles them correctly.

If not, modify the code until it does match the intended commands.

## Part 2 – adding the characters

**Challenge 20:** In the space below, sketch a friend and a foe for the adventure game.

|  |
| --- |
|  |

**Step 1**: copy the code below and paste it into a new file named **character.py**

Save this in the folder containing all earlier code.

class Character():

def \_\_init\_\_(self, char\_name, char\_description):

self.name = char\_name

self.description = char\_description

self.conversation = None

# Describe this character

def describe(self):

print( self.name + " is here!" )

print( self.description )

# Set what this character will say when talked to

def set\_conversation(self, conversation):

self.conversation = conversation

# Talk to this character

def talk(self):

if self.conversation is not None:

print("[" + self.name + " says]: " + self.conversation)

else:

print(self.name + " doesn't want to talk to you")

# Fight with this character

def fight(self, combat\_item):

print(self.name + " doesn't want to fight with you")

 return True

**Challenge 21:** Study the code for the Character class and explain what is needed to create a character object.

def \_\_init\_\_(self, char\_name, char\_description):

self.name = char\_name

self.description = char\_description

self.conversation = None

|  |
| --- |
| **Sample answer:**  To create a Character object, the constructor needs 2 parameters: the character’s name and a description of the character. |

**Step 2**: create a new Python file and save it as **character\_test.py,** in the folder with the other code.

from character import Character

harry = Character("Harry", "A smelly Wumpus")

Add this code in the character\_test.py to call the describe method on the object created to show the character's description on the screen.

harry.describe()

Save and run the program.

Figure 17 – testing the Wumpus description

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Harry is here!

A smelly Wumpus

**Step 3**: examine the character class to find the method that sets the conversation attribute.

Add code to the program to call this method and give Harry a line of dialogue.

**The set\_conversation method**

The set\_conversation method below sets the conversation attribute in the Character class:

class Character:

...

def set\_conversation(self, conversation):

self.conversation = conversation

This set\_conversation method expects to pass the message that Harry is going to output:

harry.set\_conversation ("Come closer.I can’t see you!")

Add code to call a method that allows the player to talk to Harry.

Figure 18 – setting the conversation for Harry

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Teacher note: some students will be motivated by the creative writing of dialogue and character design. Teachers may consider how this project could be achieved via group work that requires a number of skill sets.**

### Multiple characters

The game can have multiple characters to interact with. For example, 2 characters: Harry, the smelly Wumpus, and Josephine, the friendly bat.

The player may choose to fight with enemies such as Harry, but not with friendly characters such as Josephine.

Inheritance is one of the key elements of OOP; it allows you to base a new class on an existing one. The new class automatically inherits all the methods and attributes of the existing class, removing the need to include the same methods and attributes in the new class. This game will need a subclass of Character called Enemy. It will use the Character class as a basis, but have more functionality added that is specific to enemies.

**Step 1**: in the existing **character.py** file, create a new Enemy class below the Character class.

class Character:

...

# Fight with this character

def fight(self, combat\_item):

print(self.name + " doesn't want to fight with you")

return True

# Add your code here (first line unindented)

class Enemy(Character):

The name of this new class is Enemy. Putting Character inside the brackets tells Python that the Enemy class will inherit all of the methods from Character.

Character is called the superclass of Enemy, and Enemy is a subclass of Character.

**Step 2**: add code in the next line to create a constructor for an enemy object; this looks identical to the constructor of Character.

def \_\_init\_\_(self, char\_name, char\_description):

Character is the superclass of Enemy, ensure that Enemy inherits from Character.

To do this, call the superclass constructor method inside the constructor of the Enemy class.

**Step 3**: add the code below, in the next line after the Enemy constructor.

super().\_\_init\_\_(char\_name, char\_description)

**Challenge 22:** What does the line of code (above) mean?

|  |
| --- |
| **Sample answer:**  This line of code means 'To make an Enemy, first make a Character object and then customise it.' |

When the \_\_init\_\_() function is created in a subclass, the subclass will no longer inherit the superclass's \_\_init\_\_() function.

The super() function allows you to call functions in the superclass. Calling super().\_\_init\_\_() from the Enemy class will call the constructor of the Character superclass. The Character constructor sets up the attributes for the class.

Calling the previously built methods with super() saves you from needing to rewrite those methods in your subclass.

Save this code.

Using these 2 lines of code prove that the Enemy class has inherited all of the properties of the Character class.

**Step 4**: in the **character\_test.py** program change the code so that all references to the Character class become references to the Enemy class:

from character import Enemy

harry = Enemy("Harry", "A smelly Wumpus")

Run the code and test it.

The description of Harry and talking to him, should appear as before.

An object of a subclass is also considered to be an object of its superclass.

In simple terms, a class that inherits from a <superclass> is called a <subclass>.

So, in this example, an Enemy is a Character.

If a function requires a Character object as a parameter, you can give the function an Enemy object instead and the code will still work.

However, this does not work the other way around. If the function requires an Enemy object, you cannot give it a Character object, because Enemy is a more specialised version of Character.

In this step, modify the Enemy class so that it behaves differently to the Character class. This concept is called polymorphism. The Enemy class is functionally identical to the Character class, so it is a bit redundant.

**Step 5**: add new functionality to customise it.

Add a new attribute to specify the weakness of the enemy character, so the character can be defeated in the game.

An object of the class Character doesn't have an attribute called weakness; this is a customisation to be added to the Enemy class. Each enemy will be vulnerable to an item that can be found in the game.

**Challenge 23:**Provide an example where a character is vulnerable to an item.

|  |
| --- |
| **Sample answer:**  For example, you could make an enemy character called Superman, who would be vulnerable to kryptonite. |

**Step 6**: inside your Enemy class, move to the line below the one calling the superclass constructor.

class Enemy(Character):

def \_\_init\_\_(self, char\_name, char\_description):

super().\_\_init\_\_(char\_name, char\_description)

# Write your code here

Set the value of the weakness attribute to be initialised as None.

self.weakness = None

Add getter and setter methods to Enemy so that you can add a weakness for an enemy.

### Fighting Harry the Wumpus

The Character class has a method called fight(). As you don't want to fight with non-hostile characters, it simply returns a message that the character 'doesn't want to fight with you'.

**Step 1**: Inside the Enemy class, add a new implementation of the fight method to allow you to fight an enemy. This will override the implementation of fight() provided inside Character.

Enter your method like this:

def fight(self, combat\_item):

combat\_item is a string containing the name of an item, for example “cutlass” or “creampie”.

In the game, you want the player to win or lose a fight with a character. Create a new method to fight:

def fight(self, combat\_item):

if combat\_item == self.weakness:

print("You fend " + self.name + " off with the " + combat\_item )

return True

else:

print(self.name + " swallows you, little wimp")

return False

The combat\_item is the item the player is going to fight the enemy with. If it is the enemy's weakness, the method should print out a message saying that the player won the fight, and return True. Otherwise, it will print out a message saying that the player lost the fight, and return False.

**Step 2**: in the **character\_test.py** program, add a weakness for Harry by calling the setter method just written:

harry.set\_weakness("vegemite")

In this game, Harry is vulnerable to vegemite, because he is allergic to it.

Call the fight() method on Harry, choosing an object to fight with:

print("What will you fight with?")

fight\_with = input()

harry.fight(fight\_with)

Save and run the code.

If you choose to fight Harry with a banana, you will see this outcome of the fight:

Harry is here! A smelly Wumpus

[Harry says]: Come closer I can’t see you

What will you fight with?

banana

Harry swallows you, little wimp

Run the code again entering 'vegemite' as the item you will fight with:

Harry is here! A smelly Wumpus

[Harry says]: Come closer I can’t see you

What will you fight with?

vegemite

You fend Harry with the vegemite

**Challenge 24:**In the space below, explain what this step achieved.

|  |
| --- |
| **Sample answer:**  It customised the Enemy class to behave differently to its superclass, Character. |

Figure 19 – using setter methods for Harry the Wumpuses weaknesses

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

### Create an enemy in the room

**Step 1**: add the following line of code at the top of **main.py** to import the Enemy class.

from character import Enemy

Remember that character is the name of the Python file from which the Enemy class is imported.

Below the code that links the rooms together, Harry is recreated.

dungeon.link\_cave(grotto, "west")

dungeon.link\_cave(cavern, "north")

cavern.link\_cave(dungeon, "south")

grotto.link\_cave(dungeon, "east")

# Write the code to create harry here

**Step 2**: include code to set Harry's conversation and weakness attributes.

harry = Enemy("Harry", "A smelly Wumpus")

harry.set\_conversation("Hangry…Hanggrry")

harry.set\_weakness("vegemite")

Figure 20 – code snippet from the main.py where Harry the Wumpus attributes are set

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 3**: situate Harry inside a cave.

Add a new parameter inside the Cave class in **cave.py** so that a cave object knows when it has a character inside it.

Edit the Cave class constructor to add a new field.

class Cave:

def \_\_init\_\_(self, cave\_name):

self.name = cave\_name

self.description = None

self.linked\_caves = {}

# Add the character attribute here

self.character = None

Figure 21 – putting a character attribute inside a cave object

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

A cave object now has a character attribute inside it, set to None. When set, the character attribute will store a Character object. This ability to store one object inside another is called aggregation. In practice a cave may be empty, in which case this attribute will stay as None. The important thing is that a cave can contain a character.

### Putting Harry in a cave

**Step 1** add set\_character and get\_character setter and getter methods to the Cave class to enable putting a character inside a cave.

Using the set\_character method just created, put Harry into a cave by adding this code to **main.py**, immediately below the code where Harry was created.

dungeon.set\_character(harry)

**Challenge 25**:Explain why the cave can contain a character yet Harry is an enemy.

|  |
| --- |
| **Sample answer:**  You are allowed to add an Enemy instead of a Character because of polymorphism: an Enemy is a Character. |

When you enter a cave, if a character is present, the game should let you know.

**Step 2**: in the main game loop, before asking the user for a command, check whether there is an enemy in the room using the getter method.

while True:

print("\n")

current\_cave.get\_details()

# Add your code here

command = input("> ")

current\_cave = current\_cave.move(command)

If there is an inhabitant in the cave, use the describe method that was inherited by Enemy from the Character class to show who is there.

inhabitant = current\_cave.get\_character()

if inhabitant is not None:

inhabitant.describe()

Save and run the program.

Move from the cavern to the dungeon and the description of Harry the Wumpus appear.

Figure 22 – if there is an inhabitant int he cave, describe them

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Part 3 – interacting with the game

The only action the game allows currently is to move between caves, using the basic loop. With a character in the cave, this loop can be expanded to perform different actions depending on the command that is given.

**Teacher note: students should be provided the opportunity to apply their understanding and attempt the following problems:**

**'talk': talk to the inhabitant of this room (if there is one).**

**'fight': ask what item the player would like to fight with, and then fight the inhabitant of this room (if there is one)**.

**If a fight with an enemy is lost, the game ends.**

**Add an additional character to inhabit a different cave in the game.**

**Add more methods in the Enemy class to be able to interact with enemies in other ways. For example, steal from an enemy, bribe an enemy, or send it to sleep.**

**Extend the Character class to create a Friend subclass with custom methods and attributes. For example, hug a friendly character, or offer them a gift.**

**The built-in Python function isinstance() checks whether an object is an instance of a particular class.**

**Step 1**: in the **main.py** add a new command to 'talk' to the inhabitant of the room.

elif command == "talk":

# Talk to the inhabitant - check whether there is one!

if inhabitant is not None:

inhabitant.talk()

**Step 2**: add a command to 'fight', which:

* checks to see if there is a character to fight with and that they are an Enemy, using the isinstance function
* asks the player what they want to use to fight with, before checking to see if they won the fight

elif command == "fight":

if inhabitant is not None and isinstance(inhabitant, Enemy):

# Fight with the inhabitant, if there is one

print("What will you fight with?")

fight\_with = input()

if inhabitant.fight(fight\_with) == True:

# What happens if you win?

print("Bravo,hero you won the fight!")

current\_room.set\_character(None)

else:

print("Scurry home, you lost the fight.")

else:

print("There is no one here to fight with")

### **If you lose a fight with an enemy, the game should end**

The fight method written in the Enemy class returns True if the player survives and False if they do not.

**Step 1**: create a new variable to keep track of whether the player was still alive.

dead = False

**Step 2**: modify the main while loop in the game to only loop while the player is alive.

while dead == False:

If the player loses a fight, the dead variable will be set to True and the game will end.

elif command == "fight":

...

if inhabitant.fight(fight\_with) == True:

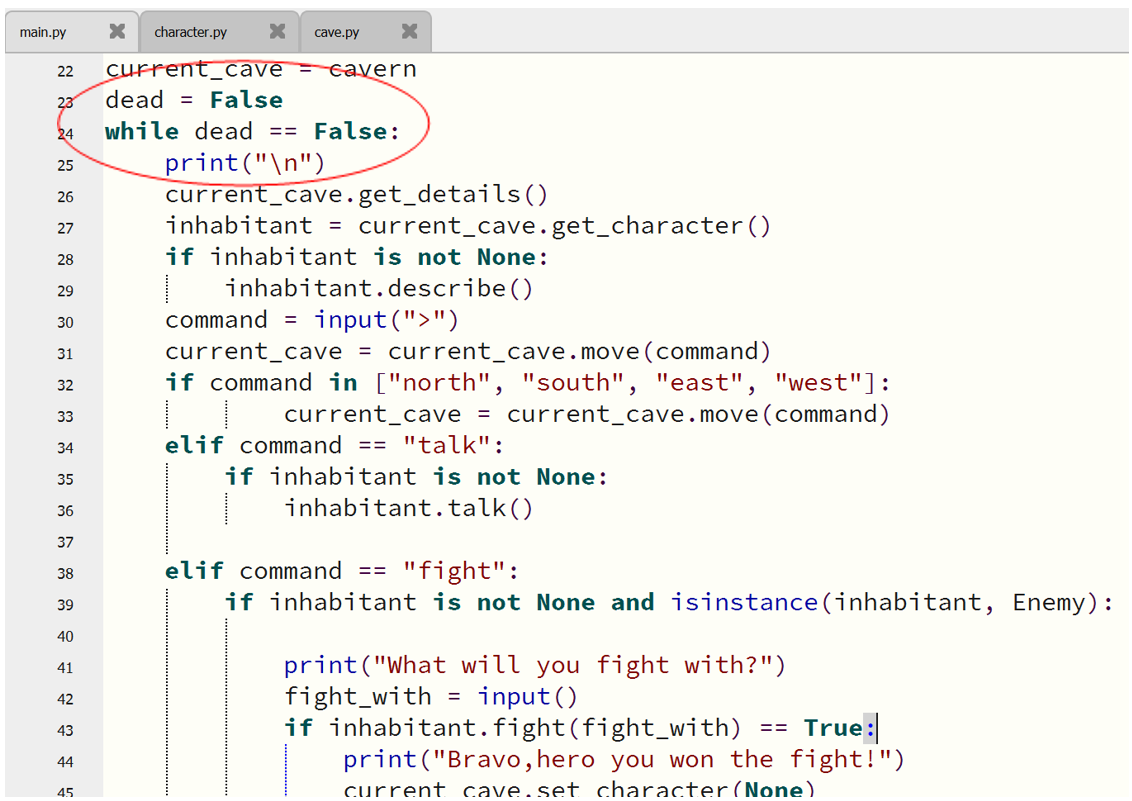
else:

print("Scurry home, you lost the fight.")

print("That's the end of the game")

dead = True

Figure 23 – a variable to track whether the player is alive



[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Challenge 26**:In the space below, explain what the variable dead does.

|  |
| --- |
| **Sample answer:**  Variable “dead” is equal to False which means alive.  The while Loop runs as long as the characters are alive (not dead). |

### **Add an additional way to interact with the characters**

**Step 1**: to add a non-enemy character to your game you will need to import the Character class from the character module, using:

from character import Character

**Step 2**: add a steal method to the Enemy class so the player can take something from them.

class Enemy(Character):

...

def steal(self):

print("You steal from " + self.name)

# How will you decide what this character has to steal?

**Teacher note: depending on the genre and theme of the game students may prefer a method to swap, buy, sell or borrow from the enemy.**

### Create a friend subclass

Extend the Character class.

Create a Friend subclass with some custom methods and attributes.

For example, you could shake hands with a friendly character, pat or offer them a gift.

Use the built-in Python function isinstance() to check whether an object is an instance of a particular class.

**Step 1**: create a new Friend class that inherits from Character so that you can create friendly characters in the game.

class Friend(Character):

def \_\_init\_\_(self, char\_name, char\_description):

super().\_\_init\_\_(char\_name, char\_description)

self.feeling = None

def pat(self):

print(self.name + " pats you back!")

# What other methods could your Friend class have?

Figure 24 – creating a Friend class that inherits from the Character class

A screen shot of Python code in Mu


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 2**: modify the character import to include the Friend class to use it in the game.

from character import Enemy, Friend

**Step 3**: create josephine, a friendly bat, and add them to a cave.

josephine = Friend("Josephine", "A friendly bat")

josephine.set\_conversation("Gidday")

grotto.set\_character(josephine)

Figure 25 – creating a friendly character called Josephine the bat

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Teacher note: some students will be motivated by the creative writing of dialogue and character design and teachers may consider how this project could be achieved via group work.**

**Step 4**: create a new command to pat the inhabitant of a room.

elif command == “pat”:

if inhabitant is not None:

if isinstance(inhabitant, Enemy):

print(“I wouldn’t do that if I were you…")

else:

inhabitant.pat()

else:

print("There is no one here to pat :(")

Figure 26 – new command to interact with an inhabitant of the room shown in Mu

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Part 4 – extending the game

### **Creating an Item class**

The game currently has a Cave class and a Character class (with Enemy sub class).

These have constructors and attributes as well as getter and setter methods.

Create a class to represent an item you might find in the game, such as a torch or a supply of vegemite.

**Step 1**: open a new Python file.

**Step 2**: save it as item.py in the same location as your main.py file.

**Step 3**: create an Item class. The Item class will have the following:

* Attributes for the name and the description of the item
* A constructor method
* Getters and setters for the name and the description of the item
* Any additional attributes and methods you would like to add

**Step 4**: test the Item class by importing item in main.py, creating an Item object, and then calling the methods.

**Step 5**: follow the same process as for the Character class:

* add an attribute to Cave so that it can store an Item,
* create the Item and assign it to the Cave.

**Step 6**: inside the game loop, check whether there is an Item in the cave, and if so, describe it.

**Teacher note**: students should be provided the opportunity to complete this before being issued a solution**.**

**Step 7** add an item attribute to the Cave class in cave.py.

class Cave:

def \_\_init\_\_(self, cave\_name):

...

self.item = None

def get\_item(self):

return self.item

def set\_item(self, item\_name):

self.item = item\_name

**Step 8**: add a describe () method to the Item class to display the item.

def describe(self):

print ("The [" + self.name + "] is here - " + self.description)

**Step 9**: create some items and add them to rooms when setting up the game.

vegemite = Item("vegemite")

vegemite.set\_description("A Wumpuses worst nightmare")

grotto.set\_item(vegemite)

torch = Item("torch")

torch.set\_description("A light for the end of the tunnel")

dungeon.set\_item(torch)

**Step 10**: in the main game loop, get the item from the cave and use the describe method to display it:

while dead == False:

item = current\_cave.get\_item()

if item is not None:

item.describe()

Figure 27 – notice the 4 tabs: main.py, characther.py, cave.py and item.py

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

### **Add items to the player’s bag**

Add a bag to allow the player to store items. This could be a list in the main part of the program, or a separate class: **bag = []**

When a player enters a room that contains an Item, the command 'take' should put the name of the current cave's Item into the bag and also set the Item attribute of the Cave to None.

**Step 1**: create a list to hold the items in a bag: **bag = []**

Figure 28 – the bag list added to the main.py

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 2**: add a new command, take, to the game, which would remove the item from the room and add it to the bag list.

elif command == "take":

if item is not None:

print("You put the " + item.get\_name() + " in your bag")

bag.append(item.get\_name())

current\_room.set\_item(None)

Figure 29 – take command added to the main.py

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Change the game so that when the player chooses an item to use in a fight, the game checks whether the player actually has an item with that name in their bag.

**Step 3**: add an if statement to check that the item is in the bag list before using it to fight with:

elif command == "fight":

# Do I have this item?

if fight\_with in bag:

else:

print("You don't have a " + fight\_with)

Figure 30 – nested If statements indicating conditions when the fight command is entered

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

### **Make the game last longer**

Use a class variable to allow the player to win the game only after they have defeated a specific number of enemies, rather than winning it the instant they defeat a single enemy.

**Step 1**: add an enemies\_to\_defeat class variable to the Enemy class, and increment the value each time an Enemy object was created:

class Enemy(Character):

enemies\_to\_defeat = 0

def \_\_init\_\_(self, char\_name, char\_description):

Enemy.enemies\_to\_defeat = Enemy.enemies\_to\_defeat + 1

**Step 2**: if an enemy was defeated, reduce the value of enemies\_to\_defeat by 1:

class Enemy(Character):

def fight(self, combat\_item):

if combat\_item == self.weakness:

Enemy.enemies\_to\_defeat = Enemy.enemies\_to\_defeat - 1

return True

Figure 31 – the Enemy class including enemies to defeat

A screen shot of Python code in Mu.


[Code with Mu](https://codewith.mu/) by [Nicholas H Tollervey](https://ntoll.org/) is licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/).

**Step 3**: in the main game loop, check to see all the enemies are defeated before ending the game:

elif command == "fight":

if inhabitant.fight(fight\_with) == True:

if Enemy.enemies\_to\_defeat == 0:

print("Congratulations, you have survived another adventure!")

dead = True

When dead is set to True, the while loop exits and the game finishes.

**Teacher note: further extensions on this adventure game could include:**

* + - * **add more caves, tunnels, pits**
      * **add more characters: Friends , Enemies and Bystanders**
      * **add attributes to the Friend and Enemy classes like Health**
      * **use variables to keep track of this health to allow for rounds or bouts**
      * **add methods like random attacks by importing random and using randint functions that can be used to increase or decrease the Health status**
      * **create and hide Easter eggs for the players to discover**
      * **unlock other levels in the game**
      * **add more discussion and dialogue to the character and player interactions**
      * **add more items to the dictionary and have characters interact with these items.**
      * **making the game with graphics by using existing libraries such as** [Pygame Zero](https://github.com/RPi-Distro/pgzero)<https://codewith.mu/en/tutorials/1.0/pgzero>.

This adventure game includes contents from a course free to reuse for non-commercial purposes.

The text from this course was created by the Raspberry Pi Foundation and is licensed under Creative Commons BY-NC-SA 4.0.

This has been copied and adapted under the Creative Commons BY-NC-SA 4.0 license.

# Design and implement computer programs involving branching, iteration and functions in an OOP language for an identified need or opportunity

**Teacher note:** students are directed to identify the branching, iterations and functions used in their adventure game by labelling with comments where they appear throughout the code.

## Implement and modify OOP programming code

Developing a well-structured and elegant solution should reduce the number of errors experienced when coding. Well written, modularised algorithms enhance the coding and debugging process.

## Clear and uncluttered mainline

Mainline approach to complex solutions: the mainline approach means using a clear hierarchical or linear design where each subroutine performs one logical task in a clear and concise fashion.

Solutions should be efficient and of high quality ideally resulting in error free code.

## Use of a clear modular structure

Software projects should be developed using a clear modular approach.

****This refers to the top-down design of the project.

**Activity 67:** in the space below, describe the use of a clear and uncluttered mainline in your adventure game

|  |
| --- |
| **Sample answer:**  The main.py file imports character, caves and item classes from other files. It clearly sets up the game and provides commands to interact and navigate through the game. |

## One logical task per subroutine

A good way of ensuring each subroutine contains only one logical task is to examine the name of the routine.

A short succinct name should be able to describe the processing occurring within the subroutine. The algorithm for the subroutine should fit comfortably on a single page.

****Algorithms that are long and complex are difficult to follow and debug.

**Activity 68:** in the space below, describe the use of one logical task per subroutine in your adventure game.

|  |
| --- |
| **Sample answer:**  An example of this would be when interacting with a character if there is one in the cave or in calling commands to navigate between caves. |

## Use of stubs

**Activity 69:** in the space below, describe a reason a developer may choose to use a stub for testing.

|  |
| --- |
| **Sample answer:**  A stub should be used to check:   * the flow or connection between the modules (sub programs) are working * if a subroutine is causing an error by replacing it with a sub (to check if the error is disappearing, if it does then the error would be in the subroutine) * a section of code, before all sub programs required by that code are finished. |

## Use of control structures and data structures

**Activity 70:** in the space below describe the use of control structures and data structures in your adventure game.

|  |
| --- |
| **Sample answer:**  The game has a main loop control structure that runs while the player is alive and branching (if statements) to determine what happens when commands are run andwhen other characters are met. |

## Ease of maintenance

In large scale organisations, accountability can be ensured by assigning certain employees to modules or subroutines. Coding should be done in such a way that future maintenance programmers can easily update the code as new requirements come to light.

**Activity 71:** in the space below, describe the ease of maintenance in your adventure game.

|  |
| --- |
| **Sample answer:**  The modular use of classes and separate files for objects, methods and attributes makes maintenance easy. For example, when a client or player wants more characters, caves or items this is easy to accomplish. Likewise, if there is a problem in the code this should be easier to code and test with a modular approach. |

## Version control

**Activity 72:** in the space below, describe the use of version control in your adventure game.

|  |
| --- |
| **Sample answer:**  During the creation of the adventure game the code was regularly saved in the same folder as all the other Python files. By saving versions and keeping track of these it is possible to trial different solutions |

## Regular backup

**Activity 73:** in the space below, describe the use of regular backups in your adventure game.

|  |
| --- |
| **Sample answer:**  During the creation of the adventure game the code was regularly backed up in the same folder as all the other Python files. After trialling other possible solutions where errors have occurred, the back-up copy can readily be retrieved. |

# Apply methodologies to test and evaluate code

## Unit, subsystem and system testing

**Activity 74:** describe the key differences between unit, subsystem and system testing. Justify your response with examples.

|  |
| --- |
| **Sample answer:**  In software engineering, there are different levels of testing to ensure the quality of a software product. These levels include unit testing, subsystem testing, and system testing.  **Unit testing**  This level of testing focuses on verifying the functionality of each individual component or module of the software application. It is typically performed by developers and involves testing the components in isolation.  **Subsystem testing**  Subsystem testing is a type of integration testing that verifies the behaviour and interaction between multiple components or modules within a subsystem. It ensures that the subsystem functions are as expected and meet the specified requirements.  **System testing**  System testing is performed to check whether the software or product as a whole meets the specified requirements. It focuses on validating the functionality of the entire system and is usually done by both developers and testers. |

## Black, white and grey box testing

**Activity 75:** describe the key differences between black box and white box testing. Justify your response with examples.

|  |
| --- |
| **Sample answer:**  **White box testing is a technique requiring explicit knowledge of the internal workings of the code being tested. It would be conducted by programmers who are testing for syntax, logic or runtime errors. An example could be a developer testing whether the attributes of an object were changed correctly, or a function modified the value in a variable correctly.**  **Black box testing is performed when the inputs and expected outputs are known. Coding experience is not required as the tester is more interested in proving that the expected output for a series of instructions or test data matches the actual output. An example would be a tester determining if the program produces the correct calculation from a series of test equations.** |

**Activity 76:** In the space below, describe grey box testing.

|  |
| --- |
| **Sample answer:**  Grey box testing is a software testing technique that combines elements of black box testing and white box testing. Black box testing is when the tester does not have access to the internal structure or code of the software, and white box testing is when the tester has full access to the code and can design test cases based on it. Grey box testing is when the tester has partial access or knowledge of the code, such as the algorithms, data structures, or architecture of the software. The tester can use this information to design more effective test cases and identify potential errors or risks in the software design. Grey box testing is useful for testing complex systems that have multiple layers or components, such as web applications. It can also help to improve the overall quality of the software by combining the input of developers and testers. |

**Activity 77:** in the space below, explain the purpose of a flag when testing?

|  |
| --- |
| **Sample answer:**  Flags are used to check that a section of code has been processed or executed. They can be used as part of a solution or as an error detection process. They are normally Boolean variables that change to true (1) if the line of code is executed, for example found = true; |

A user has been given the chance to test a product before it is officially released.

**Activity 78:** in the space below, describe the type of testing that is taking place, and why they would have been given this opportunity.

|  |
| --- |
| **Sample answer:**  The type of testing here is known as Beta testing. The developers believe the product is ready to launch officially but are asking users to test it on a limited basis or with limited functionality with real live test data. |

## Quality assurance

**Testing and evaluating**

Personnel within the software development process perform alpha testing with real data. Beta testing occurs when the product is distributed for use to a limited number of outside users. These users are engaged to report any faults or recommendations back to the software development company. Essentially, we are ensuring the product meets the original objectives and design specifications that were created during the ‘Defining and Understanding the Problem’ stage. All testing should conform to industry practice.

**Activity 79:** complete the table with definitions for Alpha and Beta testing.

|  |  |
| --- | --- |
| Alpha testing | Beta testing |
| **Sample answer:**  Testing of the final solution by personnel within the software development company prior to the product’s release. | **Sample answer:**  Testing of the final solution by a limited number of users outside the software development company using real-world data and conditions. |

## Journal of progress and pitfalls

Use the following pages as a diary to document the development of your project. Make note of skills and knowledge gained, challenges faced and successes demonstrating iterative design and evaluation. This could also be kept as a [Learning Portfolio](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/583#.YxABC73uRm8.link). It should record the progress and pitfalls.

|  |  |
| --- | --- |
| Date | Description |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |
| \_\_/\_\_/\_\_ |  |

# Appendix

The following pages are code samples including starter code to copy into a new Python file.

The first of these provides blueprints for paper, triangle, oval and rectangle objects that can be copied and pasted into Python and saved as: shapes.py

This is used in the exercises DrawingShapes\_Methods and Attributes.

**# This code requires Python 3 and tkinter (which is usually installed by default)**

**# This code will NOT work on trinket.io as the tkinter module is not supported**

**# Raspberry Pi Foundation 2020**

**# CC-BY-SA 4.0**

**try:**

**from tkinter import Tk, Canvas, BOTH**

**except ImportError:**

**raise Exception("tkinter did not import successfully - check you are running Python 3 and that tkinter is available.")**

**import random**

**class Paper():**

**# the tk object which will be used by the shapes**

**tk = None**

**def \_\_init\_\_(self, width=600, height=600):**

**"""**

**Create a Paper object which is required to draw shapes onto.**

**It is only possible to create 1 Paper object.**

**Args:**

**width (int): The width of the display. Defaults to 600.**

**height (int): The height of the display. Defaults to 600.**

**Returns:**

**Paper: A Paper object**

**"""**

**if Paper.tk is not None:**

**raise Exception("Error: Paper has already been created, there can be only one.")**

**try:**

**Paper.tk = Tk()**

**except ValueError:**

**raise Exception("Error: could not instantiate tkinter object")**

**# Set some attributes**

**Paper.tk.title("Drawing shapes")**

**Paper.tk.geometry(str(width)+"x"+str(height))**

**Paper.tk.paper\_width = width**

**Paper.tk.paper\_height = height**

**# Create a tkinter canvas object to draw on**

**Paper.tk.canvas = Canvas(Paper.tk)**

**Paper.tk.canvas.pack(fill=BOTH, expand=1)**

**def display(self):**

**"""**

**Displays the paper**

**"""**

**Paper.tk.mainloop()**

**class Shape():**

**# Constructor for Shape**

**def \_\_init\_\_(self, width=50, height=50, x=None, y=None, color="black"):**

**"""**

**Creates a generic 'shape' which contains properties common to all**

**shapes such as height, width, x y coordinates and colour.**

**Args:**

**width (int): The width of the shape. Defaults to 50.**

**height (int): The height of the shape. Defaults to 50.**

**x (int): The x position of the shape. If None, the x position will be the middle of the screen. Defaults to None.**

**y (int): The y position of the shape. If None, the y position will be the middle of the screen. Defaults to None.**

**color (string): The color of the shape. Defaults to "black"**

**"""**

**if Paper.tk is None:**

**raise Exception("A Paper object has not been created. There is nothing to draw on.")**

**# Set some attributes**

**self.height = height**

**self.width = width**

**self.color = color**

**# Put the shape in the centre if no xy coords were given**

**if x is None:**

**self.x = (Paper.tk.paper\_width/2) - (self.width/2)**

**else:**

**self.x = x**

**if y is None:**

**self.y = (Paper.tk.paper\_height/2) - (self.height/2)**

**else:**

**self.y = y**

**# This is an internal method not meant to be called by users**

**# (It has a \_ before the method name to show this)**

**def \_location(self):**

**"""**

**Internal method used by the class to get the location**

**of the shape. This shouldn't be called by users, hence why its**

**name begins with an underscore.**

**"""**

**x1 = self.x**

**y1 = self.y**

**x2 = self.x + self.width**

**y2 = self.y + self.height**

**return [x1, y1, x2, y2]**

**# Randomly generate what the shape looks like**

**def randomize(self, smallest=20, largest=200):**

**"""**

**Randomly generates width, height, position and colour for a shape. You can specify**

**the smallest and largest random size that will be generated. If not specified, the**

**generated shape will default to a random size between 20 and 200.**

**Args:**

**smallest (int): The smallest the shape can be. Defaults to 20**

**largest (int): The largest the shape can be. Defaults to 200.**

**"""**

**self.width = random.randint(smallest, largest)**

**self.height = random.randint(smallest, largest)**

**self.x = random.randint(0, Paper.tk.paper\_width-self.width)**

**self.y = random.randint(0, Paper.tk.paper\_height-self.height)**

**self.color = random.choice(["red", "yellow", "blue", "green", "gray", "white", "black", "cyan", "pink", "purple"])**

**# Getters and setters for Shape attributes**

**def set\_width(self, width):**

**"""**

**Sets the width of the shape.**

**Args:**

**width (int): The width of the shape**

**"""**

**self.width = width**

**def set\_height(self,height):**

**"""**

**Sets the height of the shape.**

**Args:**

**height (int): The height of the shape.**

**"""**

**self.height = height**

**def set\_x(self, x):**

**"""**

**Sets the x position of the shape**

**Args:**

**x (int): The x position for the shape.**

**"""**

**self.x = x**

**def set\_y(self, y):**

**"""**

**Sets the y position of the shape**

**Args:**

**y (int): The y position for the shape.**

**"""**

**self.y = y**

**def set\_color(self, color):**

**"""**

**Sets the colour of the shape**

**Args:**

**color (string): The color of the shape.**

**"""**

**self.color = color**

**def get\_color(self):**

**"""**

**Returns the colour of the shape**

**Returns:**

**color (string): The color of the shape**

**"""**

**return self.color**

**# Rectangle class is a subclass of Shape**

**class Rectangle(Shape):**

**# This is how to draw a rectangle**

**def draw(self):**

**"""**

**Draws a rectangle on the canvas. The properties of the rectangle**

**can be set using the getter and setter methods in Shape**

**"""**

**x1, y1, x2, y2 = self.\_location()**

**# Draw the rectangle**

**Paper.tk.canvas.create\_rectangle(x1, y1, x2, y2, fill=self.color)**

**class Oval(Shape):**

**def draw(self):**

**"""**

**Draws an oval on the canvas. The properties of the oval**

**can be set using the getter and setter methods in Shape**

**"""**

**x1, y1, x2, y2 = self.\_location()**

**# Draw the oval**

**Paper.tk.canvas.create\_oval(x1, y1, x2, y2, fill=self.color)**

**class Triangle(Shape):**

**# Every constructor parameter has a default setting**

**# e.g. color defaults to "black" but you can override this**

**def \_\_init\_\_(self, x1=0, y1=0, x2=20, y2=0, x3=20, y3=20, color="black"):**

**"""**

**Overrides the Shape constructor because triangles require three**

**coordinate points to be drawn, unlike rectangles and ovals.**

**Args:**

**x1 (int): The x position of the coordinate 1. Defaults to 0.**

**y1 (int): The y position of the coordinate 1. Defaults to 0.**

**x2 (int): The x position of the coordinate 2. Defaults to 20.**

**y2 (int): The y position of the coordinate 2. Defaults to 0.**

**x3 (int): The x position of the coordinate 3. Defaults to 20.**

**y4 (int): The y position of the coordinate 3. Defaults to 20.**

**color (string): The color of the shape. Defaults to "black"**

**"""**

**# call the Shape constructor**

**super().\_\_init\_\_(color=color)**

**# Remove height and width attributes which make no sense for a triangle**

**# (triangles are drawn via 3 xy coordinates)**

**del self.height**

**del self.width**

**# Instead add three coordinate attributes**

**self.x = x1**

**self.y = y1**

**self.x2 = x2**

**self.y2 = y2**

**self.x3 = x3**

**self.y3 = y3**

**def \_location(self):**

**"""**

**Internal method used by the class to get the location**

**of the triangle. This shouldn't be called by users, hence why its**

**name begins with an underscore.**

**"""**

**return [self.x, self.y, self.x2, self.y2, self.x3, self.y3]**

**def draw(self):**

**"""**

**Draws a triangle on the canvas. The properties of the triangle**

**can be set using the getter and setter methods in Shape**

**"""**

**x1, y1, x2, y2, x3, y3 = self.\_location()**

**# Draw a triangle**

**Paper.tk.canvas.create\_polygon(x1, y1, x2, y2, x3, y3, fill=self.color)**

**def randomize(self):**

**"""**

**Randomly chooses the location of all 3 triangle points as well**

**as the colour of the triangle**

**"""**

**# Randomly choose all the points of the triangle**

**self.x = random.randint(0, Paper.tk.paper\_width)**

**self.y = random.randint(0, Paper.tk.paper\_height)**

**self.x2 = random.randint(0, Paper.tk.paper\_width)**

**self.y2 = random.randint(0, Paper.tk.paper\_height)**

**self.x3 = random.randint(0, Paper.tk.paper\_width)**

**self.y3 = random.randint(0, Paper.tk.paper\_height)**

**# Randomly choose a colour of this triangle**

**self.color = random.choice(["red", "yellow", "blue", "green", "gray", "white", "black", "cyan", "pink", "purple"])**

**def set\_width(self, width):**

**"""**

**Sets the width of the shape.**

**Args:**

**width (int): The width of the shape**

**"""**

**self.width = width**

**def set\_height(self,height):**

**"""**

**Sets the height of the shape.**

**Args:**

**height (int): The height of the shape.**

**"""**

**self.height = height**

**# Change the behaviour of set\_width and set\_height methods for a triangle**

**# because triangles are not drawn in the same way**

**def set\_width(self, width):**

**"""**

**Overrides the setter method for width**

**Args:**

**width (int): The width of the shape**

**"""**

**raise Exception("Width cannot be defined for Triangle objects")**

**def set\_height(self, height):**

**"""**

**Overrides the setter method for height**

**Args:**

**height (int): The height of the shape**

**"""**

**raise Exception("Height cannot be defined for Triangle objects")**

**# This if statement means**

**# "if you run this file (rather than importing it), run this demo script"**

**if \_\_name\_\_ == "\_\_main\_\_":**

**my\_drawing = Paper()**

**# Random size and location triangle**

**tri = Triangle()**

**tri.randomize()**

**tri.draw()**

**# Specific size and location rectangle**

**rect = Rectangle(height=40, width=90, x=110, y=20, color="yellow")**

**rect.draw()**

**# Default oval**

**oval = Oval()**

**oval.draw()**

**# Oval with setters**

**oval2 = Oval()**

**oval2.set\_height(200)**

**oval2.set\_width(100)**

**oval2.set\_color("fuchsia")**

**oval2.set\_x(30)**

**oval2.set\_y(90)**

**oval2.draw()**

**my\_drawing.display()**

**from shapes import Paper, Triangle, Rectangle, Oval**

**paper = Paper()**

**rect1 = Rectangle()**

**rect1.set\_width(200)**

**rect1.set\_height(100)**

**rect1.set\_color("blue")**

**rect1.draw()**

**rect2 = Rectangle()**

**rect2.set\_width(300)**

**rect2.set\_height(100)**

**rect2.set\_color("red")**

**rect2.set\_x(100)**

**rect2.set\_y(100)**

**rect2.draw()**

**paper.display()**

Source: [Raspberry Pi Foundation](https://projects.raspberrypi.org/en/projects/turtle-race) is licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/).

# Hunt the Wumpus Adventure Game Code

**main.py**

from cave import Cave

from character import Enemy, Friend

from item import Item

cavern = Cave("Cavern")

cavern.set\_description(" A damp and dirty cave.")

grotto = Cave("Grotto")

grotto.set\_description(" A small cave with ancient graffiti.")

dungeon = Cave("Dungeon")

dungeon.set\_description(" A large cave with a rack")

dungeon.link\_cave(grotto, "west")

dungeon.link\_cave(cavern, "north")

cavern.link\_cave(dungeon, "south")

grotto.link\_cave(dungeon, "east")

harry = Enemy("Harry", "A smelly Wumpus")

harry.set\_conversation("Hangry…Hanggrry")

harry.set\_weakness("vegemite")

dungeon.set\_character(harry)

josephine = Friend("Josephine", "A friendly bat")

josephine.set\_conversation("Gidday.")

grotto.set\_character(josephine)

vegemite = Item("vegemite")

vegemite.set\_description("A Wumpuses worst nightmare")

grotto.set\_item(vegemite)

torch = Item("torch")

torch.set\_description("A light for the end of the tunnel")

dungeon.set\_item(torch)

bag = []

current\_cave = cavern

dead = False

while dead == False:

print("\n")

current\_cave.get\_details()

inhabitant = current\_cave.get\_character()

if inhabitant is not None:

inhabitant.describe()

item = current\_cave.get\_item()

if item is not None:

item.describe()

command = input(">")

current\_cave = current\_cave.move(command)

if command in ["north", "south", "east", "west"]:

current\_cave = current\_cave.move(command)

elif command == "talk":

if inhabitant is not None:

inhabitant.talk()

elif command == "fight":

if inhabitant is not None and isinstance(inhabitant, Enemy):

print("What will you fight with?")

fight\_with = input()

if fight\_with in bag:

if inhabitant.fight(fight\_with) == True:

print("Bravo,hero you won the fight!")

current\_cave.set\_character(None)

if Enemy.enemies\_to\_defeat == 0:

print("Congratulations, you have survived another adventure!")

dead = True

else:

print("Scurry home, you lost the fight.")

print("That's the end of the game")

dead = True

else:

print("You don't have a " + fight\_with)

else:

print("There is no one here to fight with")

elif command == "pat":

if inhabitant is not None:

if isinstance(inhabitant, Enemy):

print("I wouldn't do that if I were you...")

else:

inhabitant.pat()

else:

print("There is no one here to pat :(") # Add code here

elif command == "take":

if item is not None:

print("You put the " + item.get\_name() + " in your bag")

bag.append(item.get\_name())

current\_cave.set\_item(None)

**character.py**

class Character:

def \_\_init\_\_(self, char\_name, char\_description):

self.name = char\_name

self.description = char\_description

self.conversation = None

def describe(self):

print(self.name + " is here!")

print(self.description)

def set\_conversation(self, conversation):

self.conversation = conversation

def talk(self):

if self.conversation is not None:

print("[" + self.name + " says]: " + self.conversation)

else:

print(self.name + " doesn't want to talk to you")

def fight(self, combat\_item):

print(self.name + " doesn't want to fight with you")

return True

class Enemy(Character):

enemies\_to\_defeat = 0

def \_\_init\_\_(self, char\_name, char\_description):

super().\_\_init\_\_(char\_name, char\_description)

self.weakness = None

Enemy.enemies\_to\_defeat = Enemy.enemies\_to\_defeat + 1

def set\_weakness(self, item\_weakness):

self.weakness = item\_weakness

def get\_weakness(self):

return self.weakness

def fight(self, combat\_item):

if combat\_item == self.weakness:

print("You fend " + self.name + " off with the " + combat\_item)

Enemy.enemies\_to\_defeat = Enemy.enemies\_to\_defeat - 1

return True

else:

print(self.name + " swallows you, little wimp")

return False

def steal(self):

print("You steal from " + self.name)

class Friend(Character):

def \_\_init\_\_(self, char\_name, char\_description):

super().\_\_init\_\_(char\_name, char\_description)

self.feeling = None

def pat(self):

print(self.name + " pats you back!")

**cave.py**

class Cave:

def \_\_init\_\_(self, cave\_name):

self.name = cave\_name

self.description = None

self.linked\_caves = {}

self.character = None

self.item = None

def set\_description(self, cave\_description):

self.description = cave\_description

def get\_description(self):

return self.description

def set\_name(self, cave\_name):

self.name = cave\_name

def get\_name(self):

return self.name

def set\_character(self, new\_character):

self.character = new\_character

def get\_character(self):

return self.character

def get\_item(self):

return self.item

def set\_item(self, item\_name):

self.item = item\_name

def describe(self):

print(self.description)

def link\_cave(self, cave\_to\_link, direction):

self.linked\_caves[direction] = cave\_to\_link

def get\_details(self):

print(self.name)

print("----------")

print(self.description)

for direction in self.linked\_caves:

cave = self.linked\_caves[direction]

print("The " + cave.get\_name() + " is " + direction)

def move(self, direction):

if direction in self.linked\_caves:

return self.linked\_caves[direction]

else:

print("You can't go that way")

return self

**item.py**

class Item:

def \_\_init\_\_(self, item\_name):

self.name = item\_name

self.description = None

def get\_name(self):

return self.name

def set\_name(self, item\_name):

self.name = item\_name

def get\_description(self):

return self.description

def set\_description(self, item\_description):

self.description = item\_description

def describe(self):

print(

"The [" + self.name + "] is here - " + self.description

# References

This resource contains NSW Curriculum and syllabus content. The NSW Curriculum is developed by the NSW Education Standards Authority. This content is prepared by NESA for and on behalf of the Crown in right of the State of New South Wales. The material is protected by Crown copyright.

Please refer to the NESA Copyright Disclaimer for more information <https://educationstandards.nsw.edu.au/wps/portal/nesa/mini-footer/copyright>.

NESA holds the only official and up-to-date versions of the NSW Curriculum and syllabus documents. Please visit the NSW Education Standards Authority (NESA) website <https://educationstandards.nsw.edu.au/> and the NSW Curriculum website <https://curriculum.nsw.edu.au/home>.

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

Amos D (2023) Real Python [*Object-Oriented Programming (OOP) in Python 3*](https://realpython.com/python3-object-oriented-programming/)*,* Real Python website, accessed 6 September 2023.

Computer Science (2 November 2020) ['Fundamental Concepts of Object Oriented Programming' [video]](https://www.youtube.com/watch?v=m_MQYyJpIjg), *Computer Science*, YouTube, accessed 10 October 2023.

Coursera (2023) ‘[Object-Oriented Design](https://www.coursera.org/lecture/object-oriented-design/1-2-6-generalization-xjMaQ)’ *1.2.6 – Generalization*, Coursera website, accessed 9 October 2023.

Digital Learning and Teaching Victoria (2021) [*Object-oriented programming: getting started for Australian teachers*](https://sites.google.com/dltv.vic.edu.au/object-oriented), Digital Learning and Teaching Victoria website, accessed 9October 2023.

GitHub,Inc (2023) ['Pygame Zero'](https://github.com/RPi-Distro/pgzero), *pgzero*, GitHub website, accessed 6 September 2023.

Invent With Python (n.d.) [*Object-Oriented Programming and Classes*](https://inventwithpython.com/beyond/chapter15.html), *Chapter 15*, Invent with Python website, accessed 6 September 2023.

Ledger T (2013) [*Software Student’s Handbook*](https://www.lulu.com/shop/thomas-ledger/software-students-handbook/paperback/product-21156347.html?page=1&pageSize=4)[ebook], Thomas S Ledger, lulu.com.

Microsoft (2023) [*Visual Studio Code*](https://code.visualstudio.com/Download), Microsoft website, accessed 6 September 2023.

MIT OpenCourseWare (16 February 2017) ['8. Object Oriented Programming' [video]](https://www.youtube.com/watch?v=-DP1i2ZU9gk), *MIT OpenCourseWare*, YouTube, accessed 10 October 2023.

OOP workshop (12 May 2021) [Replit site](https://replit.com/@OOPWorkshop/1petvariablessol#main.py) [website], accessed 6 September 2023.

Poylas A (28 December 2018) [‘Design Patterns — A quick guide to Facade pattern](https://medium.com/@andreaspoyias/design-patterns-a-quick-guide-to-facade-pattern-16e3d2f1bfb6)’, *Medium*, accessed 6 September 2023.

Python Software Foundation (2001–2023) [*Python*](https://www.python.org/) [website], accessed 6 September 2023.

Raspberry Pi Foundation (n.d.) [Turtle Race!](https://projects.raspberrypi.org/en/projects/turtle-race) [website], accessed 1 November 2023.

Schafer C (21 June 2016) ['Python OOP Tutorial 1: Classes and Instances' [video]](https://youtu.be/ZDa-Z5JzLYM?si=UPX6p5o3ICXyMG7p), *Corey Schafer*, YouTube, accessed 10 October 2023.

Sweigart A [Tic-Tac-Toe Non-OOP and Tic-Tac-Toe OOP](https://autbor.com/compareoop/#diff) [website], accessed 6 September 2023.

Tollervey NH (2023) [Code with Mu: a simple Python editor for beginner programmers](https://codewith.mu/) [website], accessed 6 September 2023.

Wikipedia contributors (2022) [*Hunt the Wumpus*](https://en.wikipedia.org/wiki/Hunt_the_Wumpus), Wikipedia website, accessed 6 September 2023.

Wingware (1999–2023) [Wing Python IDE](https://wingware.com/downloads) [website], accessed 6 September 2023.

W3Schools (1999–2023) [What Is OOP?](https://www.w3schools.com/cs/cs_oop.php) [website], accessed 6 September 2023.

Ying K (22 September 2020) ['Inheritance/Polymorphism in Object Oriented Programming | Python for Beginners | Code with Kylie #10' [video]](https://youtu.be/C2QfkDcQ5MU?si=k6_v0tIc50WXJWKA), *Kylie Ying*, YouTube, accessed 10 October 2023.

Yob G (n.d.) [*Hunt the Wumpus*](https://www.atariarchives.org/bcc1/showpage.php?page=247)*,* atariarchives.org, accessed 9 October2023.

**© State of New South Wales (Department of Education), 2023**

The copyright material published in this resource is subject to the *Copyright Act 1968* (Cth) and is owned by the NSW Department of Education or, where indicated, by a party other than the NSW Department of Education (third-party material).

Copyright material available in this resource and owned by the NSW Department of Education is licensed under a [Creative Commons Attribution 4.0 International (CC BY 4.0) license](https://creativecommons.org/licenses/by/4.0/).

[](https://creativecommons.org/licenses/by/4.0/)

This license allows you to share and adapt the material for any purpose, even commercially.

Attribution should be given to © State of New South Wales (Department of Education), 2023.

Material in this resource not available under a Creative Commons license:

* the NSW Department of Education logo, other logos and trademark-protected material
* material owned by a third party that has been reproduced with permission. You will need to obtain permission from the third party to reuse its material.

**Links to third-party material and websites**

Please note that the provided (reading/viewing material/list/links/texts) are a suggestion only and implies no endorsement, by the New South Wales Department of Education, of any author, publisher, or book title. School principals and teachers are best placed to assess the suitability of resources that would complement the curriculum and reflect the needs and interests of their students.

If you use the links provided in this document to access a third-party's website, you acknowledge that the terms of use, including licence terms set out on the third-party's website apply to the use which may be made of the materials on that third-party website or where permitted by the *Copyright Act 1968* (Cth). The department accepts no responsibility for content on third-party websites.