LEARNER WORKBOOK

HOW TO USE COMPLEX PROGRAMMING TECHNIQUES

Python 3.9

Level 3, Credits 6

Student name	

AS91906v1a (DT3.7)

OVERVIEW

This workbook covers the following NZQA Achievement Standard:

AS91906v1 (DT3.7): Use complex programming techniques to develop a computer program

Learning Activities

Throughout this workbook, you are going to complete learning activities that will help you learn how to use complex programming techniques to develop a computer program.

Merit and Excellence

Look out for these boxes below, which provide information about additional resources for Merit and Excellence grades. When you see one of these boxes, go to the relevant section of the **Merit and Excellence Resources** file in your **Workbook Resources** folder.

Merit

Excellence

Practice Assessment

A practice assessment is given in the Appendix. It is strongly recommended that you do the practice assessment before completing the actual assessment. It provides very important practice and preparation for the actual assessment.

Assessment

This workbook will help you prepare for and complete assessment requirements.

ASSESSMENT LINK

Whenever you see this assessment link in your Learner Workbook, it will tell you what part of the actual assessment is covered by the section.

CONTENTS

SECTION 1
REVIEW OF PROGRAMMING TECHNIQUES

You are here

- **Review of programming techniques**
- Using Pygame (Part 1)
- Using Pygame (Part 2)

WHAT YOU WILL LEARN

- Variables
- Data types and data type conversions
- Doing calculations
- Control structures
- Working with lists and dictionaries
- Functions
- Testing and debugging
- Setting out program code clearly
- Commenting

In this section, you will complete the following learning activity.

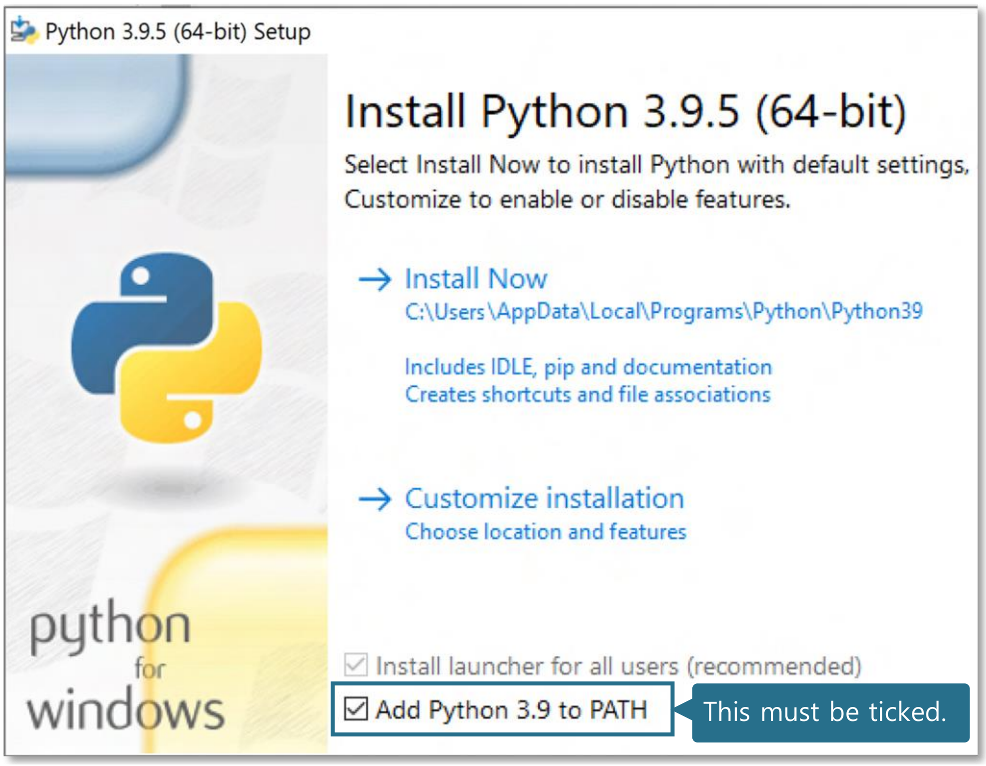
Learning Activity	
1.1	Review of programming techniques

BEFORE YOU GO FURTHER

Make sure you have Python 3.9.5 installed on your computer. You can download it as follows.

- On a Windows computer, go to the **Microsoft Store** and search for Python 3.9, or download from here: https://www.python.org/ftp/python/3.9.5/python-3.9.5-amd64.exe.

VERY IMPORTANT: If you do **not** install it via the Microsoft Store, make sure you do the following when installing.


Python 3.9.5 (64-bit) Setup

Install Python 3.9.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

This must be ticked.

- For all operating systems, you can download Python from here: https://www.python.org/downloads/.

For Windows computers, if you have difficulty downloading it from the Microsoft Store or the web, you can use the file provided in the **Workbook Resources** folder (**python-3.9.5-amd64**).

1.1 Introduction

In this section, we will briefly review the programming techniques you should already know. This includes the following.

- How to work with variables
- Data types and data type conversions
- Sequence, selection, and iteration control structures
- Working with lists
- Testing and debugging
- Setting out program code clearly
- Commenting

If you are confident with the above programming techniques in Python, complete **Learning Activity 1.1** at the end of this section. You can use the information that follows in sections 1.2 to 1.9 as reference where necessary.

1.2 Variables

A variable is created as follows:

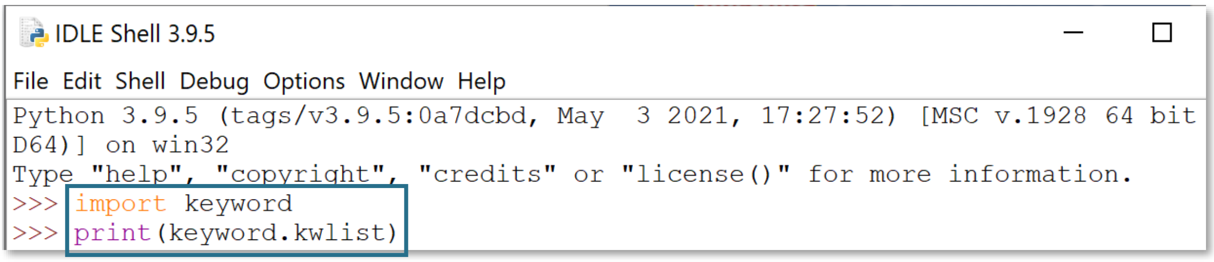
```
my_name = "Drew"
```

Variable naming

You can give a variable almost any identifier (name) you like. However, do **not** use the following.

- Space(s) or special characters (eg #, @, /). **Note**: If you want a variable that has two words (eg my name), put an underscore (_), not a space (eg player_speed).
- A number at the **start** of the variable name (eg `1friend`)
- Python keywords (eg `class`, `True`)
- Built-in function names (eg `print`, `input`)

Note: For a full list of Python keywords open IDLE and enter as shown below:



```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> print(keyword.kwlist)
```

For a full list of Python built-in functions (including usage information), go to:

- https://www.w3schools.com/python/python_ref_functions.asp or
- https://www.programiz.com/python-programming/methods/built-in

Case sensitive

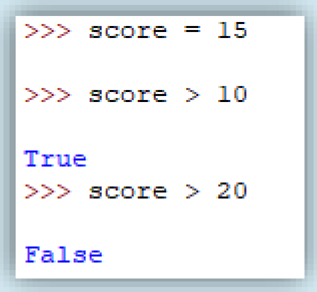
Python is case sensitive, so the following will be three different variables:

```
MyName
myName
myname
```

1.3 Data types and data type conversions

The table below summarises some common Python data types.

Data type	Use
Integer	• For whole numbers (ie no decimal points) • Can be positive or negative *Examples:* `-4, 4, -12, 12`
Float	• For decimals • Can be positive or negative *Examples:* `-4.5, 4.5, -12.93, 12.93`
String	• Single letter, number, punctuation, control character, or words • Must be surrounded by quotes. You can use single (`''`) OR double (`""`) quotes – but use the same type throughout your entire file *Examples:* **Letter**: `'k'` or `"k"` **Number**: `'5'` or `"5"`; `'5.35'` or `"5.35"`. **Note**: Even though these look like numbers, Python will treat them as **text** (because they are in quotes). **Punctuation**: `'!'` or `"!"` **Control character**: `'\n'` or `"\n"` – for a new line **Words (sequence of characters)**: `'Hi there'` or `"Hi there"`

Boolean	• Can be one of two values – True or False
	Example
	
	`>>> score = 15`
	`>>> score > 10`
	`True`
	`>>> score > 20`
	`False`

Continues next page

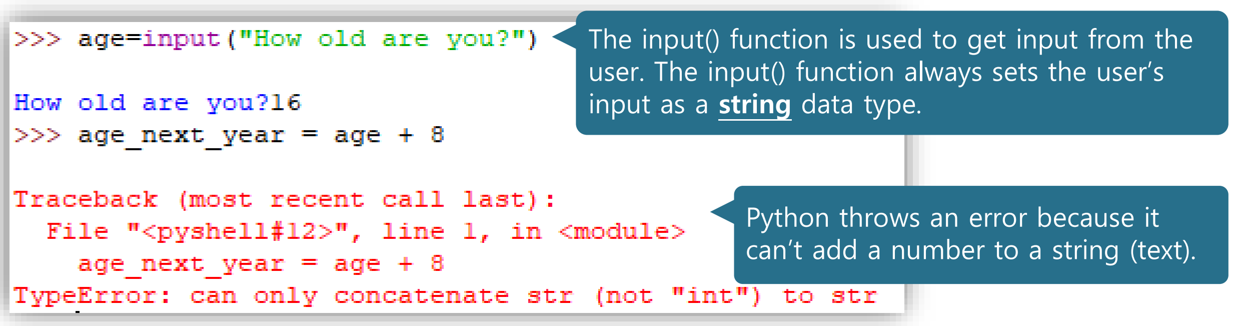
Data type	Use

Lists	• To store many different values in order
	Examples:
	`friends = ["Hanna", "Felix", "Nikola", "Cole"]`
	`primes = [3, 5, 7, 11, 13]`
	`mixed = ["Super Ninja", 63, 14, "Super Spy"]`

Dictionaries	• A dictionary consists of pairs (called **items**): a key and its value.
	• Unlike lists, dictionary items are not in a fixed order, so you can't use an index to access them.
	Example:
	`hero_0 = {`
	`"name": "Materdon",`
	`"power": "flight",`
	`"strength": 5`
	`}`
	• The name of the above dictionary is `hero_0`.
	• Dictionary items are surrounded with curly brackets `{ }`.
	• Each dictionary item has a key and a value, separated by a colon (`:`).
	key value
	`"name": "Materdon"`

Changing data types

There are times when you need to change from one data type to another. Look at the example below.



```
>>> age=input("How old are you?")

How old are you?16
>>> age_next_year = age + 8

Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    age_next_year = age + 8
TypeError: can only concatenate str (not "int") to str
```

The input() function is used to get input from the user. The input() function always sets the user's input as a **string** data type.

Python throws an error because it can't add a number to a string (text).

The table below summarises the built-in functions used to do data type conversions.

Convert ...	To ...	Use
String or Float	Integer	`int()`
String or Integer	Float	`float()`
Integer, Float, List	String	`str()`

1.4 Calculations

The following are mathematical operators used for calculations in Python.

Mathematical operators

	Name and Use	Example
+	**Addition** - Adds values	`cat = 24 + 6` **Result**: `30`
-	**Subtraction** - Subtracts values	`score = 15 - 1` **Result**: `14`

+=	**Increment** - Increases a value	`level = 6`
		`level += 1`
		Result: `7`

-=	**Decrement** - Decreases a value	`oranges = 9`
		`oranges -= 6`
		Result: `3`

*	**Multiplication** - Multiplies values	`apples = 5 * 2` **Result**: `10`
/	**Division** - Divides values. Result includes decimals.	`pizza = 13 / 8` **Result**: `1.625`
//	**Floor Division** - Divides values with result given as an integer. (Values after decimal point are cut.)	`pizza = 13 // 8` **Result**: `1`
%	**Modulo** - Divides two values and returns remainder. In the example, 13 divided by 8 is 1 remainder 5.	`pizza = 13 % 8` **Result**: `5`

1.5 Control structures: Sequence, selection, and iteration

Sequence

Sequence refers to steps of the code in order. For example:

```
player = "Flynn"
easygui.msgbox("Your name is", player)
```

Step 1

Step 2

Selection (condition)

A **selection** structure is used when a decision needs to be made, or a condition needs to be met.

Example:

```
if age < 12:
      print("You are not old enough.")
else:
   print ("You're the right age.")
```

Comparison operators

Selection structures use **comparison operators**. The table below summarises common comparison operators.

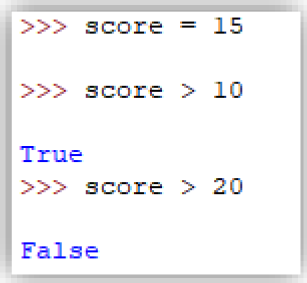
Operator	Use	Example

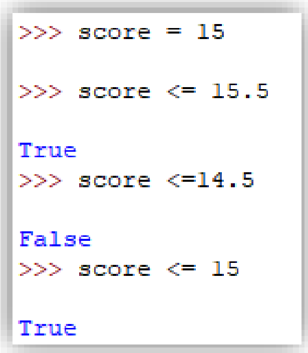
==	Checks if two values are **equal**	`>>> score = 15`
		`>>> score == 10`
		`False`
		`>>> score == 15`
		`True`

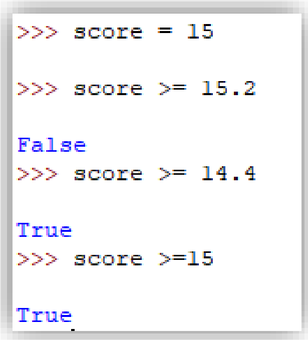
<	Checks if the first value is **smaller** than the second	`>>> score = 15`
		`>>> score < 10`
		`False`
		`>>> score < 20`
		`True`

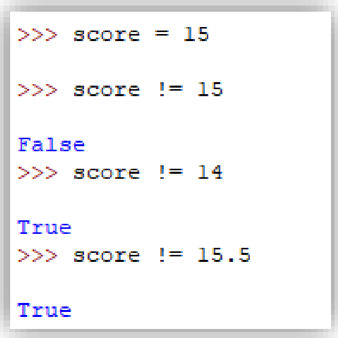
Continues next page

Operator	Use	Example

>	Checks if the first value is **bigger** than the second	 `>>> score = 15`
		`>>> score > 10`
		`True`
		`>>> score > 20`
		`False`

<=	Checks if the first value is **smaller** than or **equal** to the second	 `>>> score = 15`
		`>>> score <= 15.5`
		`True`
		`>>> score <=14.5`
		`False`
		`>>> score <= 15`
		`True`

>=	Checks if the first value is **bigger** than or **equal** to the second	 `>>> score = 15`
		`>>> score >= 15.2`
		`False`
		`>>> score >= 14.4`
		`True`
		`>>> score >=15`
		`True`

!=	Checks if the first value is **not equal** to the second	 `>>> score = 15`
		`>>> score != 15`
		`False`
		`>>> score != 14`
		`True`
		`>>> score != 15.5`
		`True`

Note: In each case, the check is done and then a Boolean value (True or False) is returned.

Selection structure types

The table that follows summarises selection structures in Python.

Example and Use

if …

```
1 player_age = int(input("How old are you?"))
2 if player_age <= 18:
3     print("Note: This game is intended for people over 18.")
4 print("Welcome to The Phoenix Odyssey!")
```

Checks if a condition is met (ie if it is *True*) [**Line 2**]
If it is, then the indented line of code is executed [**Line 3**]
If not (ie if it is *False*), then the indented code is skipped and …
the program moves to the next non-indented line of code [**Line 4**]

if …
else

```
1 player_age = int(input("How old are you?"))
2 if player_age <= 18:
3     print("Game is only suitable for people over 18.")
4 else:
5   print("Yes! You're old enough to play this game!")
6 print("The Phoenix Odyssey is an awesome game!")
```

Checks if a condition is met (ie if it is *True*) [**Line 2**]
If it is, then the indented line of code is executed [**Line 3**]
If not (ie if it is *False*), then the indented code is skipped and …
The program moves to the `else` statement [**Line 4**], and gives the output in **Line 5**.
The program moves to the next non-indented line of code [**Line 6**]

If …
else
if …
else

```
1 player_age = int(input("How old are you?"))
2 if player_age > 18:
3     print("Welcome to The Phoenix Odyssey!")
4 elif player_age >= 14:
5     print("Ask a parent to watch you play this game.")
6 else:
7     print("You're not old enough to play this game.")
```

Checks if a condition is met (ie if it is *True*) [**Line 2**]
If it is, then the indented line of code is executed [**Line 3**]
If not (ie if it is *False*), then the indented code is skipped and …
The program moves to the `elif` statement [**Line 4**]. `elif` combines an `else` and an `if` statement into one. It checks if the condition is true.
- If it is, the indented line of code is executed [**Line 5]**

If it isn't the program moves to the `else` statement [**Line 6**] and outputs the message in **Line 7**.

Using `if` to check if a variable exists

An `if` statement can also be used to check if a variable contains a value.

Example:

```python
superhero = ""
if superhero:
    print("Yes")
else:
    print("No")
```

In this example, `No` will be output because the `superhero` variable is empty.

Compound expressions

The table below summarises the logical operators that can be used to test more than one condition at a time.

Operator	Use	Example
AND	Checks if **both** conditions are met	

```python
dog = input("Dog's name: ")
cat = input("Cat's name: ")
if dog == "Dax" and cat == "Lulu":
    print("Those are both my pets.")
else:
    print("Those aren't both my pets!")
```

Example inputs and expected outputs

Test (enter)	Output expected
correct dog name (Dax); incorrect cat name (eg Max)	Those aren't both my pets!
incorrect dog name (eg Butch); correct cat name (Lulu)	As above
both names incorrect	As above
both names correct	Those are both my pets!

Continues next page

Operator	Use	Example

OR	Checks if **either** condition is met	The code is the same as the example above – except for the parts highlighted in yellow.
		`dog = input("Dog's name: ")`
		`cat = input("Cat's name: ")`
		`if dog == "Dax" or cat == "Lulu":`
		`    print("At least one is my pet.")`
		`else:`
		`    print("Neither is my pet!")`
		Example inputs and expected outputs
		Test (enter) / **Output expected**
		1. correct dog name (Dax); incorrect cat name / At least one is my pet.
		2. incorrect dog name; correct cat name (Lulu) / As above
		3. both names incorrect / Neither is my pet!
		4. both names correct / At least one is my pet.

Logical operators can also be combined – as in the example below.

```
gender = input("Gender? Enter M or F: ")
age = int(input("Age? Enter in years: "))

if gender == "F" and age > 10 or gender == "M" and age > 12:
        print("You are in Group A.")
else:
    print ("You are in Group B.")
```

The line of code the arrow is pointing to checks to see whether the person is a female over 10 **or** a male over 12. If they:

- are – they go to Group A
- are not – they go to Group B.

See the next page for example inputs and expected outputs.

Example inputs and expected outputs

Test (enter)	Output expected (Group)
1. gender: **M** age: **10**	B
2. gender: **M** age: **12**	B
3. gender: **M** age: **14**	A
4. gender: **F** age: **10**	B
5. gender: **F** age: **12**	A
6. gender: **F** age: **14**	A

Note: For complex statements combining numerous logical operators, it can be helpful (for readability) to include brackets. For example:

```
if (gender == "F" and age > 10) or (gender == "M" and
age > 12):
        print("You are in Group A.")
else:
    print ("You are in Group B.")
```

Iteration – while and for loops

Iteration structures (loops) are used for code that needs to be repeated a certain number of times. Two types of Python loops are the `while` and `for` loops.

While loops

Example:

```
username = ""
while username != "Lulu":
    username = input("Enter username:")
print("Welcome back, Lulu")
```

The user will continue to be prompted to enter their username until they enter the username 'Lulu'.

`for` loops

Example 1:

Code	Result

```
for x in range(0,5):
    print(x)
```
	```
	0
	1
	2
	3
	4
	```

Example 2 (iterating over a string):

Code	Result

```
for i in "cat":
    print(i)
```
	```
	c
	a
	t
	```

Example 3 (iterating over a list):

Code	Result

```
for i in ["Hanna", "Felix", "Nikola"]:
    print(i)
```
	```
	Hanna
	Felix
	Nikola
	```

Note: In each of the examples above, the range of the sequence (string or list) to be iterated over could be specified using a variable. For example, the two code blocks below would produce the same result.

```python
for x in range(1,5):
    print(x)
```

```python
num1 = 1
num2 = 5
for x in range(num1,num2):
    print(x)
```

break and continue in loops

`break` and **`continue`** statements are used to alter the normal flow of a loop.

The **`break`** statement breaks (ends) the loop containing it.

`break` example:

Code	Result

`for letter in "wonderful":`
`    if letter == "d":`
`        break`
`     print(letter)`
`print("All done")`
	`w`
	`o`
	`n`
	`All done`

The continue statement restarts the loop. In the example below, once the `d` is reached, the rest of the code inside the loop is skipped and the loop starts again.

`continue` example:

Code	Result

`for letter in "wonderful":`
`    if letter == "d":`
`        continue`
`     print(letter)`
`print("All done")`
	`w`
	`o`
	`n`
	`e`
	`r`
	`f`
	`u`
	`l`
	`All done`

1.6 Working with lists

The table below summarises some key ways in which to work with lists in Python.

Access elements in a list

The code below would be used to output the element at index position `1` of the **`words`** list (ie the second element on the list – `"cat"`).

Note: Index numbering starts from 0.

```
words = ["bat", "cat", "hat", "mat"]
print(words[1])
```

Getting the index of a list element

Use the **`index()`** method to search for an element in a list and return its index.

Syntax: **`list.index(element)`**

Example:

```
>>> names = ["Sally", "Mark", "Tama"]
>>> print(names.index("Mark"))
1
```

Converting strings to lists

Use either the `list()` function or `split()`method.

list() function

Use: To create a list from each element in a string.

Example:

```
>>> word = "fantastic"
>>> letters = list(word)
>>> print(letters)
['f', 'a', 'n', 't', 'a', 's', 't', 'i', 'c']
```

split() method

Use: Breaks up a string at a specified separator and returns a list of strings.

Syntax: **`string.split(separator)`** [If no separator is specified, any white space is treated as the separator.]

Continues next page

Examples:

```
>>> fruit = "Apples Bananas Mandarins Feijoas"
>>> fruit_list = fruit.split()
>>> print(fruit_list)
['Apples', 'Bananas', 'Mandarins', 'Feijoas']
```

No separator specified. The spaces between the words are treated as the separator.

```
fruit = "Apples;Bananas;Mandarins;Feijoas"
>>> fruit_list = fruit.split(";")
>>> print(fruit_list)
['Apples', 'Bananas', 'Mandarins', 'Feijoas']
```

`";"` specified as separator. **Note**: It has to be in `""` because it is a string.

Removing elements from a list

Use the `remove()` method to remove the first matching element specified.

Example:

```
>>> fruit = ["Apples", "Bananas", "Mandarins"]
>>> fruit.remove("Bananas")
>>> print(fruit)
['Apples', 'Mandarins']
```

Converting list elements to a string

Use either the **`join()`** or **`strip()`** method.

join() method

Use: Joins each string element in a list into a single string.

Syntax: **`"".join(list)`**

The part before the `.` indicates what should be used to separate the list elements in the string. So, for example:

- **`""`** will mean there will be nothing between elements
- **`","`** will mean the elements will be separated by a comma (**,**)

Continues next page

Examples:

```
>>> letters = ["H", "e", "l", "l", "o"]
>>> word = "".join(letters)
>>> print(word)
Hello
```

`""` means there will be no separator between the elements

```
>>> words = ["Today", "is", "Tuesday"]
>>> sentence = " ".join(words)
>>> print(sentence)
Today is Tuesday
```

`" "` means there will be single space between the elements (note that there is a space between the `""`)

strip() method

Use: Strips specified elements from the beginning and end of a string.

Syntax: `string.strip(" ")`

The part inside the brackets is what you want stripped from the string.

Example:

```
>>> numbers = [5, 9, 13, 17]
>>> numbers_string = str(numbers)
>>> print(numbers_string)
[5, 9, 13, 17]
>>> output = numbers_string.strip("[]")
>>> print(output)
5, 9, 13, 17
```

`str()` method used to convert the list to a string

Now we need to get rid of the square brackets.

`strip()` method used to strip the square brackets from the string

The above can be simplified into a single statement.

```
>>> numbers = [5, 9, 13, 17]
>>> output = str(numbers).strip("[]")
>>> print(output)
5, 9, 13, 17
```

Getting the length of a list

Use the `len()` function to get the number of elements in a list. This can be useful in a **for** loop – to get the loop to repeat for each of the elements in a list.

Examples:

```
>>> days = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday"]

>>> for i in range (0, len(days)):

   print("Day", str(i+1) + ":", days[i])

Day 1: Monday
Day 2: Tuesday
Day 3: Wednesday
Day 4: Thursday
Day 5: Friday
```

This is the same as saying `range (0, 5)`, because there are 5 elements in the list.

Checking if an element is in a list

The `in` keyword is used to check if an element is in a list.

Example

Code	Result

`names = ["Sally", "Mark", "Tama"]`
`if "Sally" in names:`
`    print("Name is in list.")`
`else:`
`    print("Name is not in list.")`
	`Name is in list.`

It can be used with the `not` logical operator to check if an element is not in a list.

Example

Code	Result

`names = ["Sally", "Mark", "Tama"]`
`if "Sally" not in names:`
`    print("Name is not in list.")`
`else:`
`    print("Name is in list.")`
	`Name is in list.`

Continues next page

Important:

The in keyword and not logical operator can also be used to check if something is in a string.

Example

Code	Result

`name = "Sally"`
`if "Sal" in name:`
`    print("It is in the word.")`
`else:`
`    print("It is not in the word.")`
	`It is in the word.`

Replacing an element in a list

Use an assignment statement in conjunction with the index of the element.

Example:

```
>>> names = ["Sally", "Mark", "Tama"]
>>> names[1] = "Nic"
>>> print(names)
['Sally', 'Nic', 'Tama']
```

Indicates that the element at index 1 of the names list (ie "Mark") will be assigned a value of "Nic" (ie Mark will be replaced by Nic in the list).

Adding elements to a list

Use either the `append()`or `insert()` method.

append() method

Use: Adds the element to the end of a list.

Example:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers.append(23)
>>> print(numbers)
[1, 2, 3, 4, 23]
```

Continues next page

insert() method

Use: Adds the element to a specific index in a list.

Example:

```
>>> numbers = [1, 2, 3, 4, 23]
>>> numbers.insert(4, "more")
>>> print(numbers)
[1, 2, 3, 4, 'more', 23]
```

In brackets is the index number (4), and the element to be inserted ("more").

Slicing a list

You can cut a 'slice' off a list by specifying the index point(s) for the slice.

Syntax: `list[start:stop:step]`

Examples:

```
>>> words = ["bat", "cat", "hat", "mat", "rat"]
>>> new_list = words[1:3]
>>> print(new_list)
['cat', 'hat']
```

If no start index is given, the slice starts at the beginning of the list (see example below).

```
>>> words = ["bat", "cat", "hat", "mat", "rat"]
>>> new_list = words[:3]
>>> print(new_list)
['bat', 'cat', 'hat']
```

A negative index counts from the **end** of the list (as shown in the example below).

```
>>> words = ["bat", "cat", "hat", "mat", "rat"]
>>> new_list = words[:-1]
>>> print(new_list)
['bat', 'cat', 'hat', 'mat']
```

If no stop index is given, the slice will include up to the end of the list (see below).

```
>>> words = ["bat", "cat", "hat", "mat", "rat"]
>>> new_list = words[3:]
>>> print(new_list)
['mat', 'rat']
```

Sorting lists

Use the sort()to sort elements in a list.

Use: Elements can be sorted in alphabetical order (for strings) or number order (for numbers):

Elements can be sorted in order or reverse order.

Example – Sorting in order:

```
>>> animals = ["lynx", "lion", "elephant", "zebra"]
>>> animals.sort()
>>> print(animals)
['elephant', 'lion', 'lynx', 'zebra']
```

Example – Sorting in reverse order

```
>>> animals = ["lynx", "lion", "elephant", "zebra"]
>>> animals.sort(reverse=True)
>>> print(animals)
['zebra', 'lynx', 'lion', 'elephant']
```

Include this to sort in reverse order.

Removing and returning an element from a list

The pop() method is used to remove and return an element from a list. By default, the last element on the list is removed from the list and returned.

Example:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> del_number = numbers.pop()
>>> print(del_number)
5
>>> print(numbers)
[1, 2, 3, 4]
```

You can also specify an element to be removed from a list – using its index.

Example:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> del_number = numbers.pop(2)
>>> print(del_number)
3
>>> print(numbers)
[1, 2, 4, 5]
```

Nested lists

You can have a list inside another list. Look at the example below for a program that checks the user's login and PIN code.

```
credentials = [

    ["xander", "78596"],
    ["maryanne", "82164"],
    ["ben", "45942"]
    ]

login = input ("Enter login: ")
pin = input ("Enter PIN: ")

if [login, pin] in credentials:
    print("Welcome!")
else:
    print("Access denied.")
```

The main list is `credentials`

Inside the `credentials` list are three nested lists (each list separated by a comma).

Checks to see whether the list specified by the user (login, pin) is in the `credentials` list.

1.7 Working with dictionaries

Many of the same methods that can be applied to lists, can also be applied to dictionaries. The table below summarises some key ways in which to work with dictionaries in Python.

Accessing values

The values in a dictionary are accessed using their key.

Example:

```
print(hero_0["name"])
```

Gets the value attached to the "name" key in the hero_0 dictionary

Checking if a key or value is in a dictionary

The in keyword is used to check for a key or value in a dictionary.

Examples:

To check for a key

```
if "name" in hero_0:
    print("The superhero's name is:", hero_0["name"])
else:
    print("There is no superhero name.")
```

Checks whether the key "name" is in the hero_0 dictionary

To check for a value

```
if "Materdon" in hero_0["name"]:
    print("That is the superhero's name.")
else:
    print("That isn't the superhero's name.")
```

Checks whether the value "Materdon" is attached to the key "name" in the hero_0 dictionary

Adding items

Syntax: `dictionary[key] = value`

Example:

```
hero_0 = {
"name": "Materdon",
"power": "flight",
}
hero_0["speed"] = 12
```

Adds the item with the key "speed" and the value 12 to the hero_0 dictionary.

Changing values

To change a value, assign a new value to a key.

Example:

```
hero_0["strength"] = 6
```

Changes value attached to the `"strength"` key (in `hero_0` dictionary) to `6`.

Removing and returning an item

Use the `pop()` method (works the same way as for lists).

Iterating over a dictionary

You can iterate (loop) over a dictionary using a `for` loop (works the same as for strings and lists).

Nested dictionaries

In the same way that lists can be nested inside another list, dictionaries can also be nested. Dictionaries could be nested in:

- a list
- another dictionary.

1.8 Functions

Functions are blocks of reusable code. There are inbuilt python functions such as `print()`, `input()`, `range()`, `str()`, `int()`, and so on.

You can also create your own functions.

Creating a function

A function is created using the keyword `def`.

Example:

The function below outputs "Welcome to the program".

```
def welcome():
    print("Welcome to the program.")
```

Calling a function

A function only runs when you call it. To call the function above, you would use the following in your main program: `welcome()`

Using parameters

Parameters are temporary variables that are declared when a function is defined.

Look at the example below.

Parameter (temporary variable named `user`) that is available for use only in this function.

```
def welcome(user):
    print("Welcome", user)

welcome("Eli")
```

Parameter value that is sent to the `welcome` function. In other words, for the welcome function, the temporary variable user, has a value of `"Eli"`.

More than one parameter value can get passed to a function.

Example:

```
def welcome(fname, lname, id):
    print("Welcome", fname, lname + ", ID:", id)

welcome("Eli", "Matua", "9536B")
welcome("Jacqui", "Samuels", "6945L")
welcome("Maxi", "Tau", "5624T")
```

Returning a value

Functions can also return a value back to the main program.

Example:

```
def get_num():
    num = int(input("What is the number? "))
    return num


number = get_num()
print("The number is", str(number))
```

The value stored in the variable named `num` is returned to the part of the main program where the `get_num` function is called.

The `get_num` function is called here. So the value for `num` is set to the value returned by the `get_num` function.

1.9 Testing and debugging

You need to test and debug your code as you create it. If you don't, it can be very hard to track down any mistakes.

One tool to help you with testing is to include `print` statements at 'key places' in your code. You saw an example of this earlier on:

```
player_name = input("Hi! What is your
name?")
print(player_name)
```

This line is added to help check the code as you go. When the program is complete and you have finished testing, the print lines can be deleted.

Earlier we also saw examples of how to check calculations and logical expressions in a program by inputting different values and checking that the program:

- does not display an error
- does what it is expected to do (eg the calculation is correct).

1.10 Setting out program code clearly

Remember that your program code will not only be used by the computer. You, or someone else, may need to look through your code and work on it in the future.

As your program gets longer, it can become difficult to read the code.

It is important that you set your code out clearly, so that it is easy for people to read.

Indentation

- Indents are important for selection and iteration structures. They won't work properly if you don't indent correctly.
- Indents also help to make the code easier to read.

White space between blocks of code

Another way to set out your program code clearly is to separate blocks of code with a blank line.

1.11 Commenting

Comments are an important part of a program. They are used to describe the code:

- ☐ **function** (what it is for)
- ☐ **behaviour** (what it should do).

This is important because sometimes when you (or someone else) looks at the code, it can be difficult to know what the code is doing and why.

A comment starts with a hash (#) followed by a space.

Example:

```
print("Print this message")  # This is a comment
```

You can also add block (multiline) comments as follows.

```
# This is an example of how a
# comment can be spread over
# more than one line.
```

A **docstring** is a special type of comment added at the start of a function. It:

- indicates what the function is used for
- is always put immediately after the definition statement
- starts and ends with triple quotes (""" … """)
- can be a single line or multiple lines.

Examples

Single line docstring

```
def computer_winner():
    """Message output if computer wins."""
easygui.msgbox("The computer wins!", "Result")
```

Multiple line docstring

```
def get_inputs():
     """Gets number of children and number of
     teachers from user.
     """
    children = int(input("How many students are there? "))
```

Merit

For the **step up to Merit Grade**, you must:

- use meaningful variable names
- make sure your program code follows the conventions for Python
- test and debug your program in an organised way, using different types of input.

For more information on this, see the following sections **Merit and Excellence Resources** file.

1. Variable naming conventions
2. Following conventions
3. Effective testing and debugging

Excellence

For the **step up to Excellence**, you need to:

- test and debug using expected, boundary, and invalid input
- use input validation (where possible)
- avoid the use of literals
- use actions, conditions, control structures and functions effectively.

For more information on this, see the following sections **Merit and Excellence Resources** file.

4. Comprehensive testing and debugging
5. Checking input validity
6. Literals, variables, derived values, and constants
7. Effective coding

Learning Activity 1.1: Review of programming techniques

In this activity, you will review the programming techniques you should already be familiar with.

Scenario: Sandwich maker

Your friend has started an online sandwich store. The store allows people to select ingredients to make their own sandwich.

People get to select the following for their sandwich:

- Bread
- Meat
- Garnish

Create a program that allows the user to:

- select options for bread, meat, and garnish to create their sandwich
- calculates the total cost of the sandwich
- check their order and press 'Confirm' if they want to proceed to checkout, or 'Make a change' if they want to make any changes to the order.

For now, you only need to include the following options and prices for sandwich ingredients:

Bread		Meat		Garnish	
Wholemeal	$1.00	Chicken	$2.69	Onion	$1.69
White	$0.80	Beef	$3.00	Tomato	$1.00
Cheesy White	$1.20	Salami	$4.00	Lettuce	$2.00
Gluten Free	$1.40	Vegan Slice	$3.30	Cheese	$2.50

Get your teacher to check your program before moving on to the next section.

SECTION 2
USING PYGAME (PART 1)

- Review of programming techniques
- **Using Pygame (Part 1)**
- Using Pygame (Part 2)

WHAT YOU WILL LEARN

How to use Pygame to create a computer game in Python. This includes:

- installing Pygame
- basic setup for any program using Pygame
- creating a surface
- creating a basic game loop
- creating animating shapes
- collision detection
- displaying messages.

In this section, you build the first part of a basic ‘Snake’ game. This will include completing the following learning activity.

Learning Activity	
2.1	Adding a shape

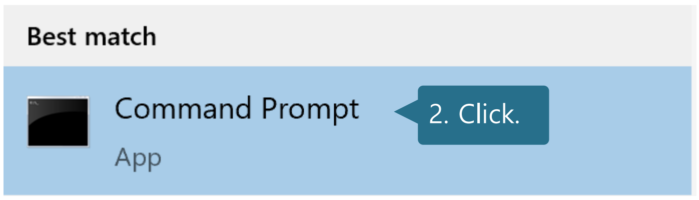
2.1 Installing Pygame

Pygame is an open-source, cross-platform Python framework that can be used to develop 2D games in Python.

In this section, you will learn how to create a basic Snake game in Python, using the Pygame framework.

Pygame is installed via the Terminal/Command Prompt using the `pip` tool. On a Windows machine:

1. Press ⊞ and type **cmd**.


Best match

Command Prompt
App

You'll see your user name here.

Enter this:

```
C:\Users\XXXXXXX>py -m pip install -U pygame
```

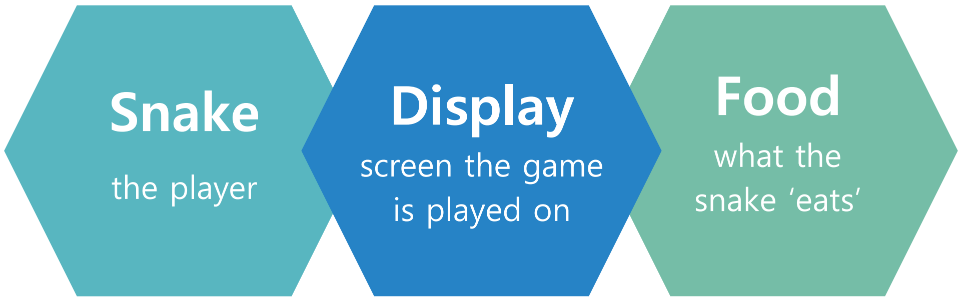
Note: For other operating systems (OSs), use `python3` rather than `py` (above).

To check that Pygame has installed properly, and to see a basic game created using Pygame, enter the following:

Use `python3` rather than `py` for non-Windows OSs)

```
C:\Users\XXXXXXX>py -m pygame.examples.aliens
```

Once Pygame is installed, you can use it to create the Snake game. There are three key components to the game:



In this section we will look at how to create these three key components. First, however, let's look at how to set up the basic structure that will be used for any game created using Pygame.

Suggested IDE for creating code

You can create the code using any suitable IDE: You could use IDLE (which comes with Python), or a more sophisticated IDE such as Visual Studio Code (**VS Code**).

An advantage of using an IDE such as VS Code is that it gives you code prompts, etc, which makes programming easier. You can download VS Code for free from:

https://code.visualstudio.com/.

Notes:

- The screenshots in this guide are created using VS Code. If you use a different IDE, what you see might look slightly different (eg the text may be coloured differently). However, this will not make a difference to the code itself.

2.2 Pygame basic setup

1. Create a new file and name it `snake1.py`.
2. Enter the following.

```
import pygame
pygame.init()



pygame.quit()
quit()
```

Initialises (starts) all Pygame modules.

All code for the game will be added here (between the initialising and quitting of Pygame).

Quits (stops) all Pygame modules.

This quits the app itself.

3. Save the file. **Note**: Going forward, you won't be reminded to save your file. You should do this regularly; at least before you test each main section of code.

2.3 Creating a surface

A Pygame surface is like a blank sheet of paper to which you can add various content.

- There are different types of surfaces for different content.
- You can have more than one surface.
- A surface can be any size you like.

For any game, you must have a **display surface.** This creates the screen or window in which the game is displayed.

To create this you use the following:

You can use any suitable variable name here.

These are the dimensions (width and height, in pixels) for the 'screen'.

```
screen = pygame.display.set_mode((1000,720))
```

- Create the screen for your snake game. Use whatever dimensions you like for the screen. However, keep in mind the devices the game will be played on. The dimensions above will work for most tablets and devices with bigger screens.
- Remember, the code should go after the line that initialises Pygame (see the highlighted code in the example below):

```
import pygame
import time
pygame.init()

screen = pygame.display.set_mode((1000,720))

time.sleep(5)
```

Excellence

Reminder: For the **step up to Excellence**, you need to do the following.

- **Avoid the use of literals**: Rather than hard coding specific dimensions for the screen, use constants for these values.
- **Ensure the program is flexible and robust**: One way to do this is to make the display resizable – so that the user can resize the game display if they wish. To do this, add the highlighted code shown below.

  ```
  screen = pygame.display.set_mode((1000,720), pygame.RESIZABLE)
  ```

- Read the information below for how to test that you have set the screen properly.

Using a 'wait' command for testing

If you try to run the app now, you won't see anything. This is because it is currently set to quit immediately (without any delay).

To check it is working correctly, you can use a simple 'wait' command. This will keep the screen up for a set period.

To make this work do the following:

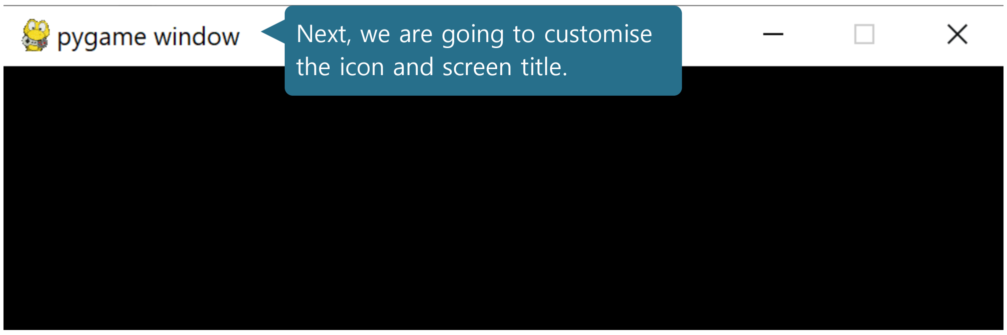
- import the `time` module
- use the `sleep` function as follows.

```
time.sleep(5)
```

Entering the above code will get the program to 'wait' for 5 seconds before quitting Pygame and the app itself. You could change this to a longer period if you wish.

Once you have added the above code, you can test the program as you would any other: Double-click on the `snake1.py` program file in File Explorer (⊞ + E).

When you run the app you should see a screen that looks something like this.



Customising the screen icon

We are going to change the icon for the screen using the PNG file **snake_icon.png**, which you can find in your **Workbook Resources** folder.

IMPORTANT

Any images you include in an app, **must** be saved in the same folder as the app.

Save a copy of the snake icon into the same folder as your **snake1.py** file.

There are two steps to changing the screen icon.

1. Upload the icon using the `load()` image function.
2. Display the icon using the `set_icon()` surface function.

Add the code highlighted below:

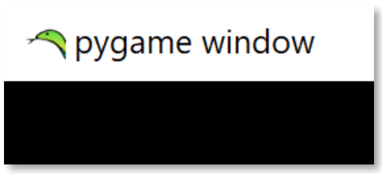
```
import pygame
import time
pygame.init()

screen = pygame.display.set_mode((1000,720))
game_icon = pygame.image.load('snake_icon.png')
pygame.display.set_icon(game_icon)
time.sleep(5)
```

Loads the icon.

Displays the icon.

▶ Test the app. The screen logo should now look like this:



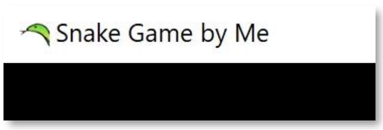
Customising the screen title

The screen title (caption) is set using the `set_caption()` surface function.

Code:

```
pygame.display.set_caption("Snake Game by Me")
```

Result:



Snake Game by Me

▶ Add a suitable title for your snake game.

2.4 Creating a basic game loop

Up until now, we have used the `time.sleep()` function to ensure the game screen only closes after a set period. We are going to replace this with a basic game loop, which will ensure the game continues to run until a 'quit event' takes place.

> **Pygame quit event**
>
> An example of a quit event for Pygame is when the user clicks the 'X' on the top right of the screen.

Look below at how to amend your code to add the basic game loop. Add this code.

```
…
pygame.display.set_caption("Snake Game by Me")
~~time.sleep(5)~~
quit_game = False
while not quit_game:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            quit_game = True
pygame.quit()
quit()
```

Delete this.

Add this (the game loop). See the notes on the next page for what the code does.

- `pygame.event.get()` receives all events from the user (ie user inputs).
- `if event.type == pygame.QUIT` checks if the event type is a QUIT event.
  Note: `pygame.QUIT` is a predefined Pygame constant.

▶ Test your app. The screen should now stay open until you press the X.

Now that you have the surface and basic game loop, it is time to create the snake and the food for the snake to eat.

We are going to break down the code into the following:

The snake	A red rectangle is used to represent the snake.
Snake animation	Code that controls how the snake is moved, based on keyboard input from the user (arrow keys).
Game play when the snake hits the 'walls' or itself (collision detection)	If the snake hits the boundaries of the screen (ie the 'walls') or itself, it dies, and the game is over.
Display message: "You died:"	When the snake dies, a message is displayed to the user.
Snake food	A blue rectangle is used to represent food the snake can eat.
Grow the snake	Each time the snake eats some food, it grows.
Keep track of the score	The game needs to keep track of the score and display it once the game is over.

Let's look at the code to create each of the above components of the game.

2.5 The snake (rectangle)

Before we create the snake, we are going to initialise:

- some colour variables for the different colours used in the game
- x and y coordinates for the snake's position on the screen.

Initialise colour variables

Pygame uses the RGB (Red, Green, Blue) colour model. When you set a colour variable, you use the RGB values.

Examples:

Black = 0, 0, 0 **White** = 255, 255, 255 **Red** = 255, 0, 0

Colour pickers for selecting other colours

For other colours, use an RGB colour picker such as the one at:

- https://www.rapidtables.com/web/color/RGB_Color.html OR
- https://www.w3schools.com/colors/colors_picker.asp.

Alternatively, enter **rgb colour picker** as the search words in a Google search, and this will bring up a colour picker in your browser.

Look at the line of code to be added (highlighted). This is to add the green used for the screen surface.

```
…
pygame.display.set_caption("Snake Game by Me")

green = (188, 227, 199)

quit_game = False
…
```

This is a data type called a **tuple**. See the definition box that follows for what a tuple is.

Tuple: Used to store multiple items in a single variable. It is ordered and **unchangeable** (unlike lists which are ordered and changeable).

▶ Following the same syntax used for the green, initialise the following colours:

- a red of your choice - for the snake
- a blue of your choice – for the food
- black – for the text (eg to display score and other messages)
- white – for the screen colour when the game is over.

Initialise snake coordinates

Coordinates

Coordinates (x and y) are used for the position of items on the screen.

- x = 0 and y = 0 is the top left of the screen.
- As you move to the right, the x value increases.
- As you move down, the y value increases.
- For rectangles and squares, you set the x and y coordinates for the top left of the rectangle/square. Look at the example below to add the snake (rectangle).

Add the following variables (highlighted). These are for the snake's x and y coordinates (ie the snake's position on screen).

```
quit_game = False

snake_x = 490
snake_y = 350
```

Note: Some Maths is needed to get the x and y coordinates for the snake:

- The screen's width is 1000.
- The snake is going to be 20 x 20 pixels.
- To get the centre point (width-wise), we say (1000-20)/2 (ie **490**).
- A similar calculation is done to get the centre point height-wise: (720-20)/2= **350**)

Excellence

Reminder: For the **step up to Excellence**, you need to avoid the use of literals.

Rather than hard coding the x and y coordinates for the snake, use derived (calculated) values.

Create a rectangle – for the snake

To create the snake, add the two lines of highlighted code below:

```
…
while not quit_game:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            quit_game = True

    pygame.draw.rect(screen, red, [snake_x, snake_y, 20, 20])
    pygame.display.update()
```

The table below explains the above code.

`pygame.draw.rect()` function draws a rectangle. It must include the following (in brackets, in the following order): • **surface** on which the rectangle should be drawn (`screen`) • **colour** for the rectangle (`red`) • **x** and **y coordinates** for the rectangle • **width** and **height** for the rectangle (width = `20`px; height = `20`px).
`pygame.display.update()` function updates the surface to include the change you have just made (adding the rectangle)

Excellence

Reminder: For the **step up to Excellence**, you need to avoid the use of literals.

For example, use constants for the dimensions of the snake rectangle.

▶ Test your program to make sure the snake (red square) is displaying as expected. **Note**: You could use a different colour or dimensions for your snake.

▶ Save the file.

2.6 Snake animation

Next, we are going to add the code to get the snake to move.

However, first save a copy of the file as **snake2.py**.

Note: It is a good idea to save different versions of a program as you create each main part of the program. This makes it easier to 'roll back' changes if you ever need to.

Before adding the movement, we need to add some new variables to:

- create a 'clock' that will be used to set the speed at which the snake moves
- hold the updating values for the x and y coordinates.

Look at the highlighted code below for what should be added.

```
clock = pygame.time.Clock()

quit_game = False

snake_x = 490
snake_y = 350

snake_x_change  =  0
snake_y_change  =  0
```

To get user input from the keyboard, the event type `pygame.KEYDOWN` is used. Look at the code on the next page for what should be added (highlighted).

Excellence

Reminder: For the **step up to Excellence**, you need to **avoid the use of literals**.

Rather than hard coding specific values for the snake's position change and for the `clock.tick`, use constants.

```
while not quit_game:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            quit_game = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                snake_x_change = -20
                snake_y_change = 0
            elif event.key == pygame.K_RIGHT:
                snake_x_change = 20
                snake_y_change = 0
            elif event.key == pygame.K_UP:
                snake_y_change = -20
                snake_x_change = 0
            elif event.key == pygame.K_DOWN:
                snake_y_change = 20
                snake_x_change = 0

    snake_x += snake_x_change
    snake_y += snake_y_change

    screen.fill(green)

    pygame.draw.rect(screen, red, [snake_x, snake_y, 20, 20])
    pygame.display.update()

    clock.tick(10)
```

`if event.key` checks to see which key the user has pressed using the Pygame constants `pygame.K_LEFT` (for left arrow), `pygame.K_RIGHT` (for right arrow), and so on.

Changes screen (surface) colour from default black to green.

Sets the speed at which each iteration of the game loop runs in frames per second (fps). In this case, it is set at 10 fps.

- Test your program to make sure the snake is moving as expected. Try out all the different arrow keys to check it is moving correctly.
- Try different `clock.tick()` values to see how changing the value alters the speed at which the snake moves.
- Save the file.

Excellence

Reminder: For the **step up to Excellence**, you need to:

- **include expected, boundary, and invalid cases in your testing** – eg what happens if a letter key is pressed rather than an arrow key?
- **check the fps** – make sure you choose an fps that is appropriate for the expected output device/s (eg tablet, desktop PC, etc). For more details, see https://www.gamingscan.com/best-fps-gaming/, or do an internet search using keywords: **fps different devices** or **frame rate different devices**.

2.7 Collision detection

If the snake touches the walls (or part of its own body), it dies, and the game is over. To create this game play, we need to use **collision detection**.

1. Create a copy of the file and save it as **`snake3.py`**.
2. To add collision detection, add the `if` statement highlighted below. It checks for when the snake's x and y coordinates are greater than or equal to the boundaries of the screen. The code shown is for a screen 1000 by 720. If you have used different dimensions, adjust the values accordingly.

```
            …
            elif event.key == pygame.K_DOWN:
                snake_y_change = 20
                snake_x_change = 0

    if snake_x >= 1000 or snake_x < 0 or snake_y >= 720 or snake_y < 0:
            quit_game = True

    snake_x += snake_x_change
    snake_y += snake_y_change
    …
```

Excellence

Reminder: For the **step up to Excellence**, you need to **avoid the use of literals**.

3. Test your program. When the snake touches the edge of the screen, the program should quit.

Merit

Reminder: For the **step up to Merit Grade**, you need to test and debug your program in an organised way. The following can help you do this.

- Adding print lines to check your snake position.
- Reducing the frame rate (fps) to slow down the movement of the snake.

4. Save the file.

2.8 Displaying messages

Below are the steps to follow when displaying a message using Pygame.

1	Create a Font object - using `font.Font()` or `font.SysFont()` method.
2	Create a Text surface object (object on which the text is rendered) – using `render()` method.
3 [OPTIONAL]	[Only do this step if you want your text to be in a 'text box'] Create a Rectangle object for the Text surface object you created in Step 2 – using `get_rect()` method.
4	Copy the Text surface object to your Display surface object (ie your screen) – using `blit()` method.
5	Update the Display surface object – `display.update()`

1. Save a copy of your program as **`snake4.py`**.
2. Add the highlighted code on the next page that has been added for each of the above steps. Refer to the notes that follow the code for details of what the code does.
3. Make any adjustments to the code you wish (eg you might want to use a different font, a different colour for the text or text box, etc).

Excellence

Reminder: For the **step up to Excellence**, you need to **avoid the use of literals**.

4. Test the program. Make sure the text is displaying as you expect it to and that it is displaying when it should (ie when the snake hits the sides of the screen).
5. Save the file.

```
clock = pygame.time.Clock()

font = pygame.font.Font("freesansbold.ttf", 50)          1
#font = pygame.font.SysFont("arialblack", 50)


def message(msg,txt_colour, bkgd_colour):
    txt = font.render(msg, True, text_colour, bkgd_colour)       2
    text_box = txt.get_rect(center = (500, 360))       3
    screen.blit(txt, text_box)       4

…
…
  clock.tick(10)

message ("You died!", black, white)
pygame.display.update()       5
time.sleep(3)

pygame.quit()
quit()
```

Calls `message` function and sends parameters for `msg` and txt_`colour` and `bkgd_colour`.

Add so message doesn't disappear immediately. You might like to set it for longer when testing.

Notes:

- A **function** has been used to create the text box and text. This is so it can be used for more than one message in the game, without needing to duplicate code each time.
- **Step 1**: Here two options are given, showing you how to use:
  - the built-Pygame font (called `freesansbold`) OR
  - a font on your system (`arialblack` in this case). You can use any font on your system: Usually the font names are written as one word without spaces or underscores.

  The second argument (ie the information in the brackets) is for the font size.

Excellence

Reminder: For the **step up to Excellence**, you need to ensure the program is **robust and flexible**.

One way to do this is to package the fonts used in the program along with the program itself. To do this, put font files for the fonts you want to use in the same folder as the program.

- **Step 2**: The `render()` method is applied to the font object created in Step 1. The arguments are as follows:
  - The **message** to be displayed (`msg`)
  - **Antialiasing** (boolean: `True` or `False`). `True` smooths the edges of the font.
  - **Font colour** (`black`)
  - **Background colour** (`white`). This value is optional. If you leave it out, there will be a transparent background.
- **Step 3**: Creates the text box (as a rectangle). The arguments for the `center()` method = x coordinate, y coordinate.

  The coordinates given in the example set the centre of the rectangle to line up with centre of the screen. **Reminder**: The screen is 1000 x 720, so the centre of the screen will be 1000/2 (`500`) and 720/2 (`360`).
- **Step 4**: The Pygame `blit()` method is used frequently. It draws one image onto another. The arguments are:
  - **source** - image to be drawn – in this case `txt` (ie the text to be displayed)
  - **destination** - image on to which it is drawn – in this case `text_box` (ie the text box).

    Note: If you do not include a rectangle for a text box (Step 3), you can give the coordinates as the **destination** for where you want the source to be displayed. For example: `screen.blit(txt, (400, 100))`

2.9 The food (other shape)

Next you are going to create the food for the snake. Using what you have already learned, you should be able to do this yourself. Complete the Learning Activity that follows to add the food for the snake.

Learning Activity 2.1: Adding a shape

You have already learned how to create a rectangle shape (for the snake). Now use a different shape for the food the snake will eat.

Follow the instructions below.

1. Save a copy of your program as **`snake5.py`**.
2. Use a suitable **`draw`** method to create the food. You could use a rectangle (as you did for the snake), or try a different shape like a circle.

   For example: `pygame.draw.circle().` The parameters are as follows.

```
circle(surface, color, center, radius, width)
```

- `surface` = the surface on which the circle should be drawn
- `color` = circle colour
- `center` = x and y coordinates for the **centre** of the circle.
  IMPORTANT: See instruction 3 on the next page for how to determine the coordinates for the food.
- `radius` = radius of the circle (ie measurement from centre of the circle to its edge). This is half of the full size (diameter) of the circle.
- `width` (optional) = line thickness. If not indicated, the circle will be filled; otherwise it is outlined.

Other shapes in Pygame

If you want to use a different shape, research it for yourself here:

- https://devdocs.io/pygame/ref/draw OR
- http://www.pygame.org/docs/ref/draw.html OR
- do an Internet search for: **pygame draw module**.

Excellence

Reminder: For the **step up to Excellence**, you need to **avoid the use of literals**.

Note: You won't be reminded of this going forward. You should be able to identify where constants, variables, or derived values can be used instead of literals.

Continues next page

3. Randomly position the food on the screen - use the `random` module (you should already know how to use it). If you need a reminder, do an Internet search for: **python random module**.

   Note: You need to think of the game area as a grid – to enable collision detection. Look at the example below:

   ```
   food_x = round(random.randrange(20, 1000 - 20)/20)* 20
   ```

   - Assuming your circle has a radius of 10px (ie it is 20px in diameter), setting the range for the random value to be from 20 to 1000-20 means that the food will never be offscreen.
   - Dividing the randomly chosen number by 20, rounding it, and then multiplying it by 20, ensures the position for the food will always be a multiple of 20. This allows it to work with the snake position, which changes by 20px each time it moves. In other words, both the snake and food are on a 20px grid.

4. Add code to:

   4.1. check whether snake touches the food (ie you will need to use collision detection – see Section 2.7)

   4.2. set a new random position for the food if the snake touches the food.

5. Give the user the option to **quit** the game or **play again** when they die.

   Notes:

   - You will need to adjust some of your existing `quit_game` code for this.
   - If you want the user to enter a letter to indicate they want to quit (eg X) or play again (eg A), you can use `pygame.KEYDOWN` (eg `pygame.K_x` and `pygame.K_a`)

6. Put the game loop inside a function.

7. Test the game and save it.

Merit

Reminder: For the **step up to Merit Grade**, you need to test and debug your program in an organised way. Doing the following will help with this:

- Add print lines to check your snake and food position.
- Slow down your frame rate while testing.

2.10 Grow the snake

Each time the snake eats some food, it grows. If it hits its own body, the snake dies. A simple way to achieve this is to use lists for the snake head and body – as shown in the highlighted code below.

Note: If you need a reminder of how lists work (including nested lists), refer to Section 1.6 of this Workbook.

```
def draw_snake(snake_list):
    for x in snake_list:
        pygame.draw.rect(screen, red, [x[0], x[1], 20 , 20])
…
    snake_x_change = 0
    snake_y_change = 0
    snake_list = []
    snake_length = 1
…
~~pygame.draw.rect(screen, red, [snake_x, snake_y, 20, 20])~~
    snake_head = []
    snake_head.append(snake_x)
    snake_head.append(snake_y)
    snake_list.append(snake_head)
    if len(snake_list) > snake_length:
        del snake_list[0]
    for x in snake_list[:-1]:
        if x == snake_head:
            game_over = True
    draw_snake(snake_list)
    pygame.display.update()
```

This function goes before the main game loop.

This code is deleted because it is included at the start of the `draw_snake` function (above).

Note that this is created in a nested list. The `snake_head` list is nested inside the `snake_list` list.

Used to detect if snake head touches snake body. **Note**: -1 is used to count from the end of the list.

- Save a copy of your program as **`snake_6.py`**.
- Add the code above, as well as a line of code to increment the `snake_length` variable by 1 when the snake touches the food.
- Test your program.

You now have the basics of the snake game in place. In the next lesson, we will look at how to replace the shape you used for the 'food' with an image. We will also look at how to track and display the score and high score for the game.

SECTION 3
USING PYGAME (PART 2)

You are here

- Review of programming techniques
- Using Pygame (Part 1)
- **Using Pygame (Part 2)**

WHAT YOU WILL LEARN

How to refine a game using Python. This includes:

- using images rather than shapes for sprites
- tracking and displaying a score
- saving a high score to a file (file reading and writing)
- updating and displaying a high score from a file.

In this section, you will refine your snake game.

Learning Activity	
3.1	Tracking the player's score
3.2	Using time to determine the score
3.3	Updating and displaying the high score

3.1 Adding an image as a sprite

When we talk about the use of an image to represent something (eg an item, a character, etc) in a game, it is usually referred to as a sprite. Up until now you have used a simple shape to represent the food the snake eats. Let's look at how to use an image instead, to make your game more interesting.

1. Save a copy of your snake game as **`snake7.py`**.
2. Replace your existing code to create the food with the following.

```
food = pygame.Rect(food_x, food_y, 20, 20)
apple = pygame.image.load('apple_3.png').convert_alpha()
resized_apple = pygame.transform.smoothscale(apple, [20,20])
screen.blit(resized_apple, food)
```

What's happening in the above code?

- First a rectangular surface is created using `pygame.Rect`, and assigned to a variable named `food`. Note that the arguments for `pygame.Rect` are the same as for `pygame.draw.rect`: x and y coordinates, width, height. However, `pygame.Rect` and `pygame.draw.rect` are different:
  - `pygame.Rect` creates a rectangular surface onto which something can be copied/displayed (blit)
  - `pygame.draw.rect` creates a rectangular object.
- Next the image is loaded and assigned to a variable (`apple`). Note the following:
  - You can use png, jpg, and bmp file formats. For a full list of supported formats, see https://www.pygame.org/docs/ref/image.html.
  - In this example, **apple_3.png**, is used, but you could use one of the other apple images supplied in your **Workbook Resources** folder.
    - If you decide to use a red apple, you might wish to change the colour of your snake.
    - If you use a green apple, you might wish to change the background screen colour.
  - `convert_alpha()` helps to speed up blitting (for png images). Use `convert()` instead for jpg files.

Continues next page

- Next, the image is scaled (made smaller), so that it will fit the surface we have created (`food`). It is assigned to a variable called `resized_apple`.
    - `pygame.transform.smoothscale` ensures a less pixelated look for the image. `pygame.transform.scale` will be more pixelated, but also 'crisper'. The arguments for this method are:
        - the variable to be transformed (scaled) – `apple` in this case
        - the dimensions for the variable (20 wide by 20 high – which is the same as the surface - `food`).
- Finally, the blitting takes place: The variable (image) `resized_apple`, is copied onto the surface (`food`).

3. Test your program. **Note**: If you used a circle for the food in **`snake7.py`**, you may need to make some adjustments to the:
    - code used to detect when the snake touches the food
    - the starting point for the snake (so that it works on a grid of 20px – ie round the starting point down to the closest multiple of 20).

3.2 Scoring

Every time the snake eats some food, it grows, and the player's score goes up by 1 point. You already have a variable that can be used to track this: `snake_length`.

With what you have already learned, you should be able to create the code to determine and display the player's score. Complete the activity on the next page to do this.

Learning Activity 3.1: Tracking the player's score

Create and implement a function that uses the `snake_length` to get and display the player's score.

- The player should only get 1 point for each 'segment' added to the snake (ie the snake head does not count as a point).
- Display the score at the top right of the screen. **Note**: If you wish, you can do this without creating a rectangle for the text.
- Use a different font style and colour than you used for the 'Quit or play again' text.
- Save the file as **`snake8.py`**.
- Test the program to ensure the score is being:
    - determined correctly
    - displayed as expected.

Learning Activity 3.2: Using time to determine the score

In Learning Activity 3.1, you used the number of 'segments' added to the snake to determine the score.

Another common way to determine scores in a game is to use time: the longer the player 'stays alive', the higher their score. In the context of the snake game, this would only make sense if you automatically started movement of the snake as soon as the game starts.

To get some practice on how time can be used to determine a score, follow the instructions below.

1. Save a copy of your game as `snake8_alt.py`.
2. Use the `time` module to time how long the player stays alive. To do this use the `time.time()` method.

The `time.time()` method

When it is called, the `time.time()` method returns the time since **epoch**. **Epoch** is the point at which time starts – usually Thu Jan 1 00:00:00 1970.

You can use this to determine the time between two time points as follows.

- Create a variable to hold the start time
  eg `start_time = time.time()`
- Add the rest of your game and then, to get the time that has passed since the `start_time` was set, use:
  `round(time.time() – start_time)`

3. Create a variable to set the score as the amount of time that has passed since the start time.

3.3 Reading and writing to file (saving high score)

We want to be able to keep track of the high score for the game. To do this we need to save the high score in a separate file. This is known as **writing** to a file.

Look at the highlighted code below which does this.

```
exit_font = pygame.font.Font("freesansbold.ttf", 30)
score_font = pygame.font.SysFont("arialblack", 20)

def load_high_score():
    try:
        hi_score_file = open("HI_score.txt", 'r')
    except:
        hi_score_file = open("HI_score.txt", 'w')
        hi_score_file.write("0")
    hi_score_file = open("HI_score.txt", 'r')
    value = hi_score_file.read()
    hi_score_file.close()
    return value
…
       food_x = round(random.randrange(20, 1000 - 20)/20)*20
       food_y = round(random.randrange(20, 720 - 20)/20)*20

    high_score = load_high_score()

    while not quit_game:
…
```

Uses `open()` method to try to open a file called `HI_score.txt` and open it in read-only (`'r'`).

Uses `open()` method to open file in write mode (`'w'`). If file does not exist, creates file.

Writes content to file. (Overwrites any existing content.)

Add inside main game loop function.

Notes:

- `try` … `except` is used in Python to test a block of code for an error (`try`), and if one is present, the `except` block of code is run to handle the error.

  In the example above:

  - The program tries to open a file called `HI_score.txt`. If no such file exists, the `except` block executes.
  - In the `except` block, a new file with the name `HI_score.txt` is created, and a value of `0` is written to the file.

- After the `try` … `except` blocks, the content in the file is open in read-only mode, then the file is read (`hi_score_file.read()`), and assigned to the `value` variable.

- Then the file is closed (`hi_score_file_close()`) and `value` is returned.

- Make a copy of your **`snake8.py`** file (<u>**not**</u> the **`snake8_alt.py`** file) and save it as **`snake9.py`**.
- Add the highlighted code on the previous page.
- Test the program.

  Important: Once you have run the program, you should see the `HI_score.txt` file added to your program folder, and you should see that the file contains a value of `0`.

  The high score will <u>**not**</u> yet be displayed in the game (you will do this next).

3.4 Updating and displaying a high score from a file

Next, you need to add the code needed to:

- update the high score
- display both the current score and the high score
- save the new high score when the program closes.

You have already been introduced to all the code you will need to do this, so complete Activity 3.3 to add this functionality.

Learning Activity 3.3: Updating and displaying the high score

Use what you have learned to add functions that do the following.

1. Save a copy of your file as `snake10.py`.
2. **Update the high score**: The code should check if the current score is greater than the high score. If it:
   - **is**, the current score should be set as the updated high score
   - **is not**, the existing high score should be set as the updated high score.
3. **Display both the current score and the high score**: Adjust the existing function that displays the score, so that the high score is also displayed.
4. **Save the new high score when the program closes**: The code should:
   - open the `HI_score.txt` file in write mode
   - write the new updated high score
   - close the file.
5. **Check that you have documented your program**: Make sure you have included comments where relevant.

Excellence

Reminder: For the **step up to Excellence**, you need to **make sure your program is flexible and robust**.

Below are some ways to do this.

- **Ensure you handle unexpected input**. For example, will the high score be saved if the person presses the X button to quit rather than the Q key?
- **Check user input**. For example, if the player presses the X button, you could check with them whether they really want to quit.
- **Use classes and objects (object-oriented programming)**.
  - For more details, refer to section 8 on **Using classes and objects** in the **Merit and Excellence Resources**.

3.5 Other Python and Pygame tools

It is not possible to cover all the possible Python and Pygame modules, functions, etc in this Workbook. It is important that you are able to use the internet to research other tools you might need.

Below are some useful resources for:

- **Python**:
    - https://docs.python.org/3/library/index.html
    - https://www.w3schools.com/python/python_reference.asp
    - https://www.tutorialspoint.com/python/index.htm

- **Pygame**:
    - https://www.pygame.org/docs/
    - https://devdocs.io/pygame/
    - https://realpython.com/pygame-a-primer/

If you can't find what you need, try an internet search using carefully chosen keywords.

Search results from sites like StackOverflow, StackExchange, GeeksforGeeks, Reddit, or CodeProject.

Note: For the practice assessment that follows, you may find the Python `math` module useful – especially the `dist()` function (to determine the distance between sprites – for collision detection).

APPENDIX: PRACTICE ASSESSMENT

In this practice assessment, you need to develop a computer program by using complex programming techniques. The brief for the computer program is given in your **Workbook Resources/Practice Assessment Resources** folder (file name: **91906v1 Practice Assessment Brief**).

Important: Read through **all** the instructions in this task before you start work on it.

What you will be assessed on

You will be assessed on how well you do the following.

- Write code for the program to allow it to perform the specified task
- Use complex techniques to develop the program
- Set out the program code
- Document the program
- Test and debug the program

Use the checklist below to ensure you meet all the requirements of the assessment. As you complete an item, tick it off in the 'Student to tick' column.

Item	Student to tick	Assessor to tick
1. Write code for a program to perform the task specified in the brief. The computer program needs to use/do the following.		
• Variables storing at least two different types of data (eg numeric, string/text, Boolean)		
• All of the following control structures: ☐ Sequence ☐ Selection ☐ Iteration	 ☐ ☐ ☐	 ☐ ☐ ☐
• Get input from a user, file, sensors, or other external source		
• Produce output		

Continues next page

Item	Student to tick	Assessor to tick
• Use the following complex programming techniques:		
☐ Using a graphic user interface (GUI) that allows the user to interact with the program by inputting values, clicking a button, etc (eg using Pygame)	☐	☐
☐ Reading from, or writing to, files	☐	☐
☐ Using third party or non-core API, library, or framework (eg Pygame)	☐	☐
☐ [OPTIONAL] Object-oriented programming using class(es) and objects you define yourself	☐	☐
2. Ensure the program code is set out clearly. For example, make sure you have used: • indentation correctly • spaces between blocks of code.		
3. **[MERIT GRADE]** Make sure that the code follows the conventions of the programming language you have used. In other words, you need to stick to the guidelines and commonly accepted ways of doing things for the programing language.		
4. **[EXCELLENCE GRADE]** Ensure the program is well-structured, and is a logical response to the task specified in the brief.		
5. **[EXCELLENCE GRADE]** Ensure the program is flexible and robust. Examples of ways to do this include the following. • Use functions, methods, actions, conditions, and control structures effectively • Include input validity checks • Ensure the program correctly handles expected, boundary, and invalid values • Use constants, variables, and derived values – rather than literals		
6. Document the program with comments.		
7. **[MERIT GRADE]** Document the program with appropriate names and comments that clearly describe the function of the code and how it behaves (ie what it does).		
8. Test and debug the program to ensure it works correctly on a sample of expected input.		

Continues next page

Item	Student to tick	Assessor to tick
9. **[MERIT GRADE]** Test and debug the program in an **organised way** to make sure it works on both: ☐ expected input ☐ relevant boundary input.	☐ ☐	☐ ☐
10. **[EXCELLENCE GRADE]** Test and debug the program **comprehensively** to ensure it works correctly for: ☐ expected input ☐ relevant boundary input ☐ invalid input.	☐ ☐ ☐	☐ ☐ ☐

Evidence to be submitted

You need to submit the following to your assessor.

- The completed program – in electronic format.
- Records of how you tested and debugged the program. This could include a testing plan, annotated screenshots (eg showing the code you have tested and changes you made as a result of testing), a vlog showing how you conducted testing, etc.

NOTES

Kura
SOLUTIONS

support@kurasolutions.co.nz
www.kurasolutions.co.nz