**Writing to a file**

[**17 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023700/comments#fl-comments)

**Storing data is important, as it enables the user to retain information for future use.**

Data can be stored for many reasons, for example:

* Sensor data for analysis in a spreadsheet
* Configuration settings for a complex experiment

In programming you can use many different data structures to store data, such as variables, lists, or dictionaries. Although using these data structures is convenient, they have a flaw. As soon as the code finishes running, your data is lost.

To store data for reuse you need to write it to a file.

**Writing to a file**

You are going to create a program which will ask the user for their name and save it to a file called names.txt.

Create a new program and save it as write\_to\_file.py.

The first line of code uses the open() function to open a file called names.txt:

f = open("names.txt", "w")

The parameter w specifies that the file should be opened in write mode. The open file will then be referred to by the variable f for easier use.

**Tip:** If you have an existing file of the same name, names.txt, Python will delete the contents of that file and overwrite it with the data from this project. Now would be a great time to back up any data you don’t want to lose.

With the file created, you now need to store something inside.

Use the input function to ask the user their name and capture their answer via keyboard input. Use the variable name to store the answer:

name = input("Hello, what is your name? ")

To show that the variable has stored the name correctly, you can print the name to the screen.

print("Hello " + name)

Next, write the contents of the variable name to the file f using the write() method:

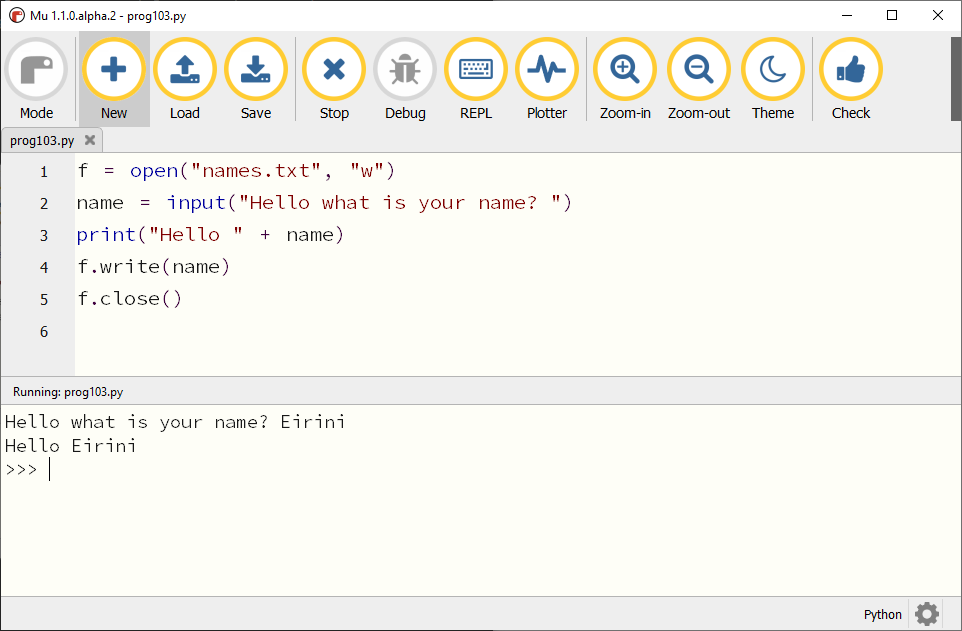
f.write(name)

A method is simply a function that belongs to, or is part of, another structure called a **class**; in this case you are using the file class. A method works in the same way as a function, but is called slightly differently. The write() method handles all of the tasks needed to write data to a file.

The final step in the program is to close the file to ensure that the contents are saved correctly and reduce the risk of data loss. Use the close() method:

f.close()

Run your program and enter your name when prompted.



**Open the file**

Using a text editor, such as Notepad or TextEdit, open the new names.txt file and you will see your name within it.

**Add another name**

Run the code again, enter a different name, and then reopen the names.txt file using a text editor.

When the code is run again, it overwrites the contents of the file with the new name. So when working with files in this manner you can create and destroy the contents of a file, but not update it.

How can more than one name be added to names.txt? For this you need to learn how to **append** to a file.

**Appending to a file**

The dictionary definition of append is ‘to add to the end of a written document’, and the same is true in programming.

To enable your program to append to a file, you only need to make two changes.

The first line created a link to the names.txt file and made it writable using w. To append to the file, all you need to do is change the w to an a.

f = open("names.txt", "a")

The second change you need to make is when the data is written to the file. You should still write the variable name to the file, but also add \n; this inserts a newline character after the name has been appended, so that the next name is written to the next line of the file.

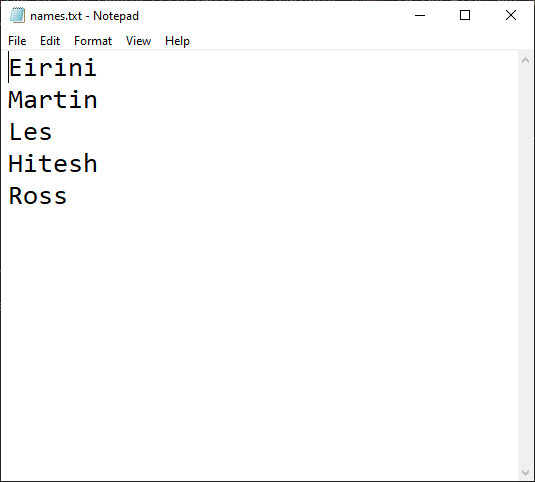
f.write(name + "\n")

**Tip:** It is common for new programmers to miss the \n. Their data will still be saved to the file, but as a single long line of text.

**Run the code**

Save and run the new program a few times, each time adding a new name.

Open names.txt in a text editor and you should see that each name is on a new line, creating a list of names.



Can you think of when you would write to a file rather than append?

# Reading from a file

[**15 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023701/comments#fl-comments)

**In the previous step you learnt how to write data to a file, but how can you read data from a file?**

Files can store lots of useful data. For example, a file can be used to record the time and temperature for a weather station project. This data can then be used in a spreadsheet to chart the data graphically. This data would be worthless without a means to read and use the data. It is like owning a book which is never read: the knowledge is contained within, but never used.

Python and other programming languages can read data from a file in much the same manner as data is written to it.

Using a text editor, create a list of fruits and vegetables and save it as a text file called food.txt.

Apples

Oranges

Bananas

Tomatoes

Cucumber

Carrots

Peas

Create a new program and save it as food\_read.py.

This program will read the contents of the file food.txt and print it to the screen.

Just as in the previous step, open the file and use f as a reference to work with the file.

**Note:** You need to use read mode, by using the parameter r instead of w (write) or a (append).

f = open("food.txt","r")

Next, read the contents of the file into a variable called data and print it.

data = f.read()

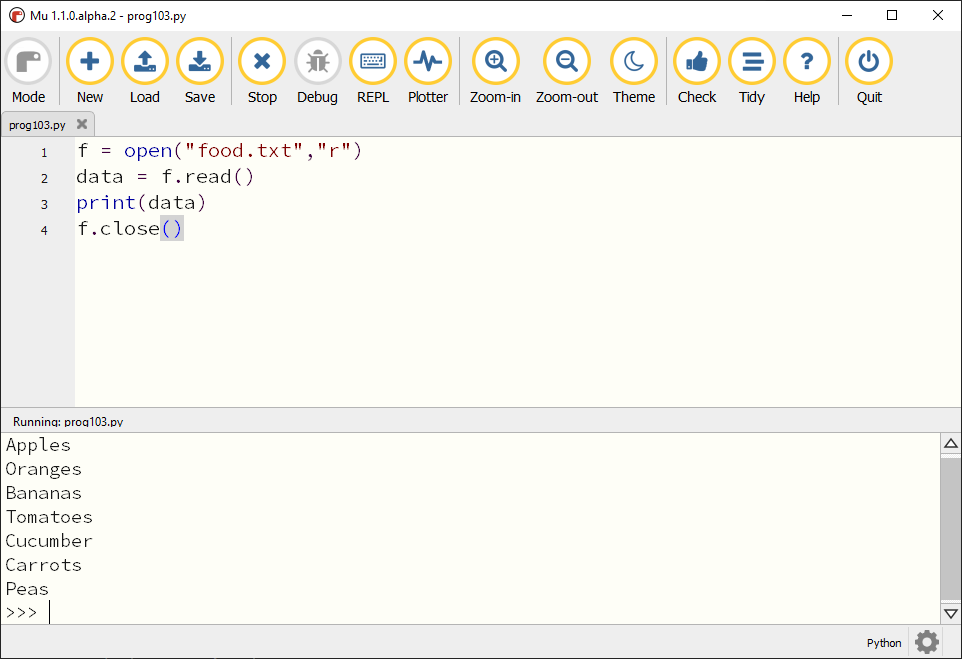
print(data)

Don’t forget to close the file.

f.close()

Leaving files open can cause data loss and can also prevent access when files are used by multiple concurrent users.

Run the program. Python will print the list all at once as a multiline string, showing each item on its own line.



### Iterating over the file contents

Sometimes it can be useful to iterate over the contents of a file, reading the contents line by line.

Open the file in the same manner as before.

f = open("food.txt", "r")

Using a for loop, read the contents of the file f line by line. Each time through the loop, a new line from the file is stored in the variable line and then printed to the screen.

for line in f:

print(line)

**Tip:** for loops are loops that iterate a set number of times. This code will read each line from the file until it reaches the end of the file, when the for loop terminates.

Once again, it is important to close the file.

f.close()

### Using a with statement

Notice how you have to close the file each time your program finishes. Ensuring that the file is opened and closed correctly is good file management as it reduces the risk of data loss. This is good practice and Python has built-in functions to simplify this.

Using a with statement you can open a file, do whatever you want with it, and then automatically close it.

The with statement also provides a means to manage any errors, for example if the file is not present, or the file name is incorrect.

This program will open a file and store it in the variable f, read the contents of the file to the variable data, and then print it to the screen.

with open("test.txt") as f:

data = f.read()

print(data)

**Note:** No code is needed to close the file, as the with statement handles closing the file once the contents have been read.

**Text files and binary files**

[**18 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023702/comments#fl-comments)

**Python can create files for humans to read, and it can also create files that are intended only for computers to read.**

In the previous steps you have been writing and reading data to text files. In each project, text files were created and saved in such a manner that a human can read them if they want to, but in reality the text files have already been interpreted by the computer. Computers do not understand human language, and so values are written in a way that a computer can process, known as binary.

Binary files can be used to store any data; for example, a JPEG image is a binary file designed to be read by a computer system. The data inside a binary file is stored as raw bytes, and is not readable by humans.

**Using a binary file**

Here is a short program. Read the code line by line; can you work out what the code will do?

data **=** [100,24,255]

buffer **=** bytes(data)

**print**(buffer)

f **=** open("binary.txt", "bw")

f**.**write(buffer)

f**.**close()

**How does this code work?**

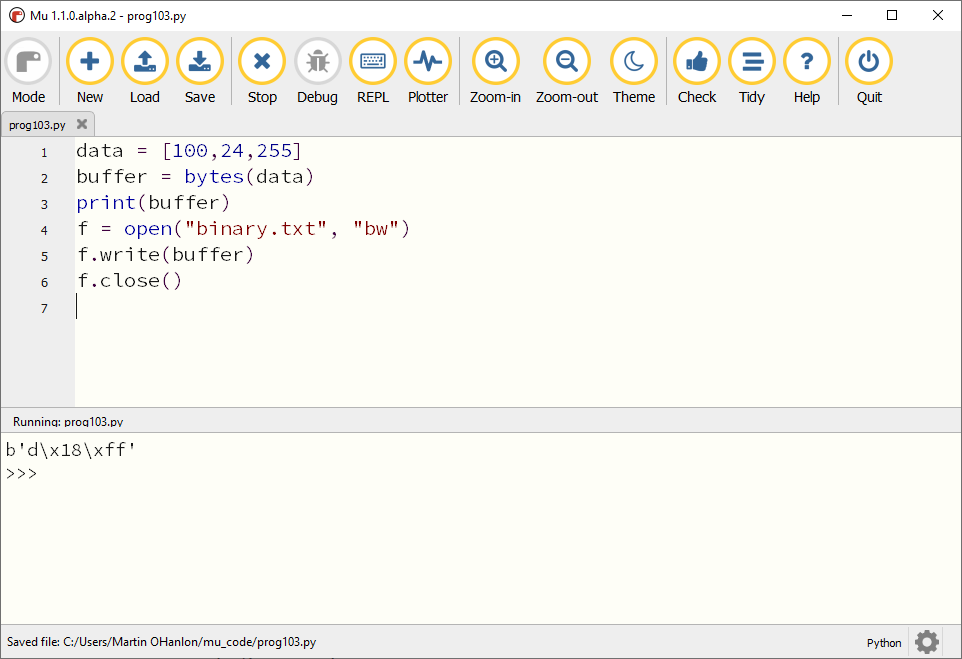
The program starts by creating a list called data containing three numbers. The data list is then converted to bytes (binary numbers) and stored in the buffer variable, which is then printed to the screen.

A new file called binary.txt is created and the mode set to bw so that bytes can be written into the file, making it a binary file.

The contents of the buffer variable are written to the file and finally the file is closed.

**Test the code**

Copy the code above into your Python editor, save it, and run the code. You can look at the contents of binary.txt using a text editor.



Does it match what you saw in the REPL/console?

Your text editor has tried to open the file, but has failed to read the contents correctly. This is because the text editor is trying to interpret the file contents as text.

**Reading the file**

Reading binary files is very similar to reading text files. Have a look at this code and see if you can work out what it does.

f **=** open("binary.txt", "br")

binary **=** f**.**read()

**print**(binary)

data **=** list(binary)

**print**(data)

f**.**close()

This code opens the file and sets the mode to binary read, "br". It then reads the contents of the file into the variable binary and prints it to the screen.

The data type of the binary variable is bytes. The bytes data is converted into a list using the list() function and stored in the variable data.

When you run this code, you should see the following:

* The contents of the file, printed to the screen as bytes
* The original list, converted back from the binary data

In this step you learnt more about how Python can work with files that are intended for both humans and machines to understand.

**Creating a game with a scoring mechanism**

[**17 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023703/comments#fl-comments)

**The score is a key component in many video games. Game creators can use scores to drive the player forward to beat their past score and do better. But what happens when the power is turned off? Where do those scores go?**

To understand how scores are saved, you will create a simple maths game that captures the user’s score as they play, and then shows the high score at the end of the game.

To explain how the code works I will use *decomposition*, which means breaking down a complex problem into small steps that are easier to understand.

The game works like this:

* Ask a question
* Store the answer
* Check the answer
* Respond based on the answer to the question:
  + Increase the score if the answer is correct
  + Keep the current score if the answer is incorrect
* Once all the questions have been asked, show the player their score
* If the player’s score is greater than or equal to the current high score
  + Update the player’s high score
* Else
  + Keep the current high score
* Loop back and start again

Decomposing a problem like this can help make it clearer how to code it.

**Creating the game**

Create a new file called maths-game.py containing the following code. Review the program before running it, and predict what you think will happen.

highscore **=** 0

**while** True:

score **=** 0

**print**("Welcome to the Maths Quiz")

**print**("Can you answer three questions and score maximum points?")

**print**("Question 1: What is the product of 2x2x2?")

answer **=** int(input("Your answer :>> "))

**if** answer **==** 8:

**print**("Correct")

score **=** score **+** 1

**print**("Your score is", score)

**else**:

**print**("Incorrect, the answer is 8")

**print**("Your score is", score)

**if** score **>** highscore:

highscore **=** score

**print**("The high score is", highscore)

**Review the code**

A variable called highscore is created and initialised to 0 at the start of the game.

highscore **=** 0

The game is run continuously using a while True (infinite) loop. At the start of each *round* of the game the variable score is initialised as zero. This will be used to keep score during each round of the game.

**while** True:

score **=** 0

A welcome message and instructions on how to play are displayed at the start of each round.

**print**("Welcome to the Maths Quiz")

**print**("Can you answer three questions and score maximum points?")

The first question is asked.

**print**("Question 1: What is the product of 2x2x2?")

answer **=** int(input("Your answer :>> "))

The answer is captured using the input() function. The answer is cast to an integer using int(). Notice how int() wraps around the input statement, converting the answer to the integer data type. The player’s answer is stored in the variable answer.

**Tip:** For this question the answer is an integer, but if you think that your users will enter a non-integer number you can use float() to preserve any decimal places.

An if statement is used to check the answer. If the answer is correct, the player will be told they are right and their score will be increased by one point.

**if** answer **==** 8:

**print**("Correct")

score **=** score **+** 1

**print**("Your score is", score)

**else**:

**print**("Incorrect, the answer is 8")

**print**("Your score is", score)

**Ending the game**

At the end of the game, the current high score is replaced if the score of the player is greater than the highscore.

**if** score **>** highscore:

highscore **=** score

**print**("The high score is", highscore)

**Create more questions**

Create at least two more questions and add them to your quiz.

You may find it useful to print some blank lines to make the game easier to follow. Adding \n to the start or end of a string to be printed will do this, for example:

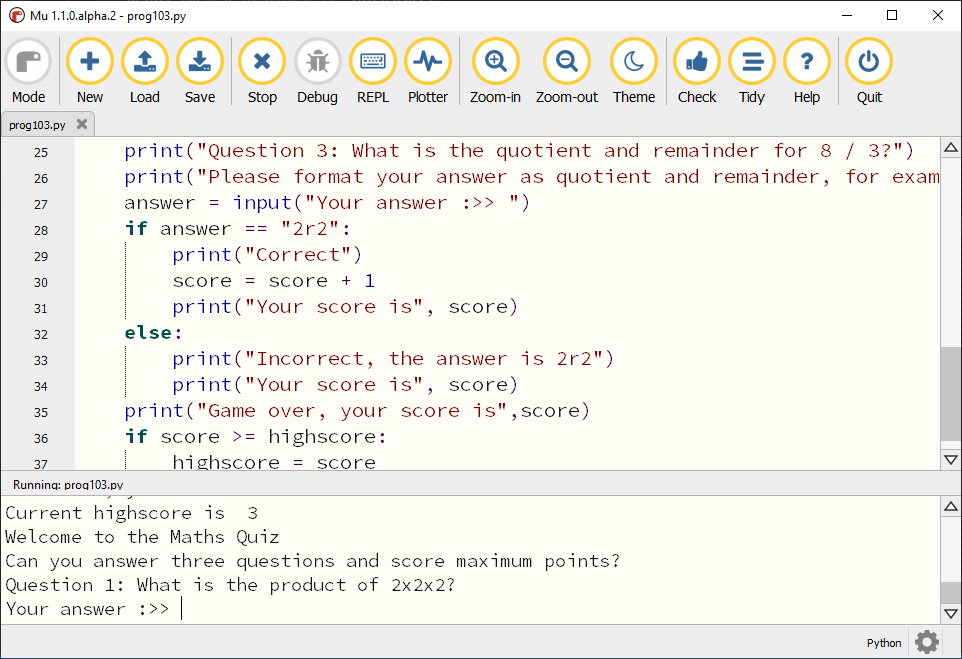
**print**("\nQuestion 1: What is the product of 2x2x2?")

**Test the game**

Save your game and run it. When a question is answered correctly, the score increases by one. At the end of the game, the player will receive their final score.

Play the game a few times, testing different scenarios by varying the number of questions answered correctly each time. After a few loops, close the game and rerun it. What happens to the high score?

You should have noticed that the high score is stored over multiple loops of the game, but not if you close the game and rerun it. Think back to the start of the game: you created a variable called highscore outside of the loop. This means that whenever the game is restarted, the value of highscore will be reset to 0.



**Persistence is important**

This game has no long-term storage. Each time the game is started, the high score is forgotten and reset back to zero.

# Data Persistence

[**16 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023704/comments#fl-comments)

**Persistent storage is vital for modern-day computing**

How might data be stored without computers? You may have an address book which stores the details of your friends and family, indexed by the first letter of their surname. Books in a library are organised into categories and then stored in alphabetical order based on the author’s name. As you may already be aware, data also needs to be stored in the computing world. For this, you need the data to demonstrate persistence.

Persistence in computer science is defined as “the characteristic of a state that outlives the process that created it”. But what does this mean? In some of the previous steps this week you wrote code to write data to external files for future use. The code which wrote these files has since stopped running, but the files that were created still exist — this is persistence. The act of saving and loading files in applications is the most basic example of the concept, but there are other examples. Every time you listen to a piece of digital music, or view a digital photo, you are opening a file that has previously been created and saved.

But why is persistence so important? With persistence the code can save important information to an external file. This could be something as trivial as a high score or as important as patient medical data. Code can also be used to access the data in the file in the future. If persistence did not exist then the code would simply forget and reset each time it is used. Many programming languages have methods and frameworks to save and retrieve data from files. In Python there are the basic file operations to read and write to files. But these can be developed to work with more complex data structures, just like the address book example above, offering a logical and efficient means to store and access data. For example, you can store address book data in a file using a string, as demonstrated previously, or you could structure the data into a dictionary that uses the name of a person to retrieve their details.

Many beginner programming projects don’t take advantage of data persistence. Where would persistence be useful? In the comments section, tell other learners about a time when you delivered an activity which could be adapted to save and retrieve data.

**Saving a high score**

[**24 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023705/comments#fl-comments)

**The maths quiz is missing a persistent high score. As soon as the game ends, the score data is lost.**

In this step you will use a file to store the high score data between games.

**How will the game work?**

Your maths quiz program needs to be extended to read the high score from a file and write it back if the player beats it.

Your program should follow this process:

* Load the high score from highscore.txt
* Set the current score to zero
* Ask the questions and adjust the current score as before
* Once all the questions have been asked, show the player their score
* Check the player’s score against the high score
  + If the player’s score is greater than or equal to the high score
    - Write the score to the highscore.txt file
* Loop back and start the quiz again and set the score to zero

**Loading the high score**

The highscore variable is currently initialised to 0 each time the game starts. Change your program to load this value from a file named highscore.txt instead.

**with** open("highscore.txt", "r") **as** f:

highscore **=** f**.**read()

highscore **=** int(highscore)

**print**("The high score is",highscore)

**Saving the high score**

Add this code to your program to save a new high score to the highscore.txt file after the questions have been answered.

**if** score **>=** highscore:

highscore **=** score

**print**("You have set a new high score")

**with** open("highscore.txt", "w") **as** f:

f**.**write(str(highscore))

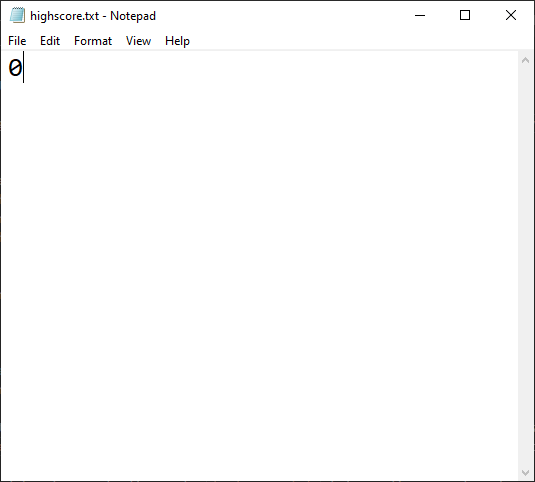
**else**:

**print**("Better luck next time")

**Test your program**

The first time you run your program, it will raise an error: FileNotFoundError: [Errno 2] No such file or directory: 'highscore.txt'. This is because there is no highscore.txt file to read.

Use a text editor to create an empty highscore.txt file in the same directory as this program, and enter 0 into the file.



Why did you put 0 into the file? Share your thoughts in the comments section.

Try the game a few times, making sure to stop and rerun the program. The high score should now be saved.

# Exception handling

[**20 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023706/comments#fl-comments)

**When you tried to open a file that didn’t exist, Python raised an error and your program stopped. In this step you will explore how to deal with errors as they occur.**

### The try ... except statement

Programming languages have ways to handle exceptions (issues or problems that occur when code is run). In Python there is a try ... except statement that is used for such an occasion. This is often also known as a **try**, **catch**.

The statement works by first trying to run code that is indented inside the try statement. If there is an error or exception, the program **catches** the error, which **triggers** the exception, executing the code indented under the except.

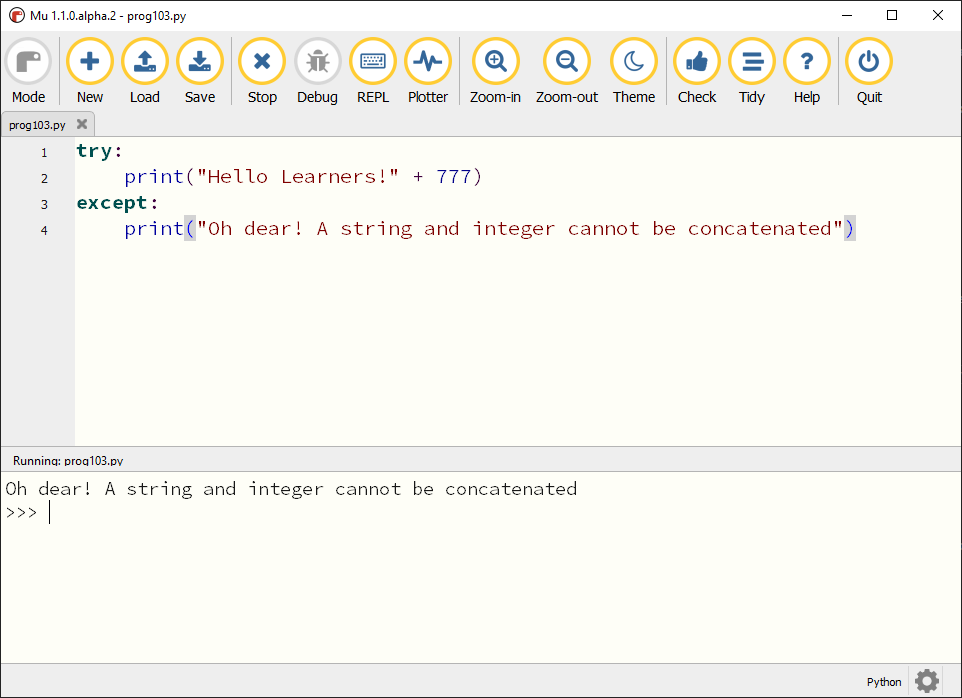
**try**:

**print**("Hello Learners!" **+** 777)

**except**:

**print**("Oh dear! A string and integer cannot be concatenated")

In this example, the try statement will attempt to print a message. If **any** error occurs, the except part is triggered and the error message is printed.



It is not good practice to catch all errors, because if an error occurs that you weren’t expecting, it should not be hidden from the user. Your program should anticipate the type of error that you wish to **catch** and then take an appropriate action.

The type of error raised by the above program is a TypeError. You can explicitly tell the except to only catch TypeError using except TypeError.

**try**:

**print**("Hello Learners!" **+** 777)

**except** TypeError:

**print**("Oh dear! A string and integer cannot be concatenated")

You can also retrieve information about the error using except TypeError as error, which will allow you to print the error type and message.

**except** TypeError **as** error:

**print**(type(error))

**print**(error)

### Adding exception handling to the game

In the game you used the highscore.txt file to store the highest score. When there was no highscore.txt file, the code raised an error. Delete your highscore.txt file and reload the game to see the error message again. Using exception handling, the program can be adapted to handle errors in a friendlier manner.

This code snippet adds an exception handler to your game. It tries to open the highscore.txt file and read the contents; a FileNotFoundError error is raised if the file does not exist. The program will then create the file and write 0 to it.

**try**:

**with** open("highscore.txt", "r") **as** f:

highscore **=** f**.**read()

highscore **=** int(highscore)

**print**("The high score is",highscore)

**except** FileNotFoundError:

**print**("Creating a new highscore.txt file")

f **=** open("highscore.txt", "w")

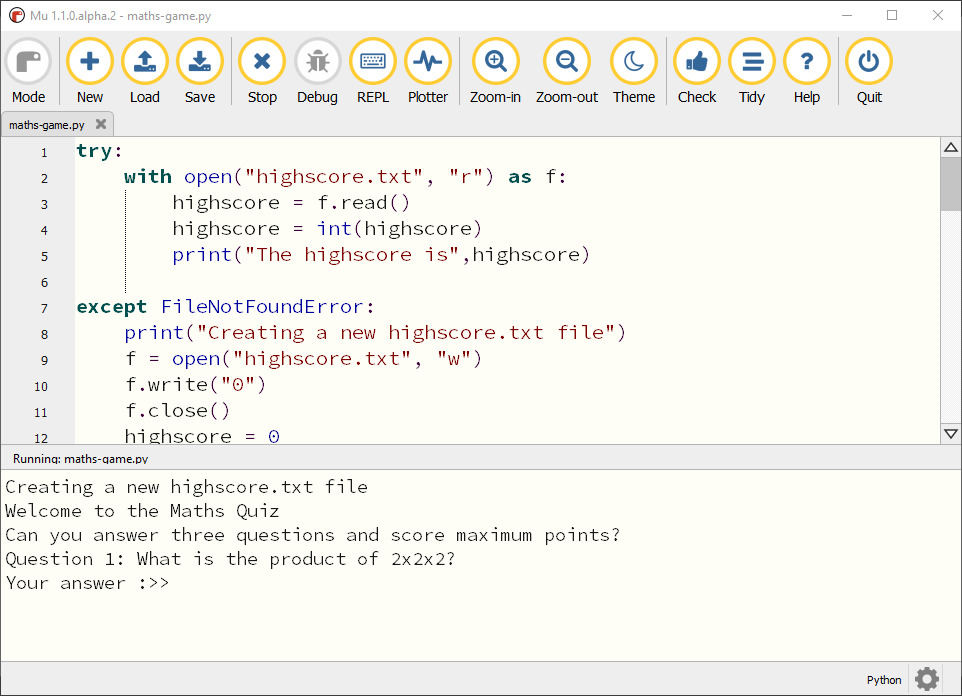
f**.**write("0")

f**.**close()

highscore **=** 0

### Test the code

Run the code and note how the highscore.txt file is created if it didn’t previously exist.



**Storing a table of high scores**

[**22 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023707/comments#fl-comments)

**The game is great, but it can be better! In this step you will improve the game with new features that will store the players’ names and their scores, and display the top ten scores in a high score table.**

To do this you will have to structure the data in the file.

A simple record structure for this type of data would be a single line for each player, containing the name of the player followed by a single space and then the player’s score:

Sura 5

Les 2

Michael 1

This structure allows you to understand the relationship between the name and the score for multiple players.

**How will the game work?**

Decomposing the game into steps, it should work like this:

* Open the highscores.txt file
* Load the data from the file into the program
* Display the high score table
* Ask the questions, adjust the current score, and show the player their final score
* Ask the player for their name
* Insert the player’s name and score
* Save the data to the highscores.txt file
* Loop back and start the quiz again

There are four new processes which you will need to add or replace in your quiz program to add the high score table:

1. Load the high scores data
2. Display the high scores
3. Add a new score to the high scores data
4. Save the high scores data

**The code to complete these processes is below; your task is to add these code snippets to your quiz.**

**Load the high scores data**

To keep track of the high scores in your game, the names and scores will need to be stored in your program.

Use this code to read the high scores data from a highscores.txt file into two separate lists, scores and names.

**print**("Loading high scores")

scores **=** []

names **=** []

**try**:

**with** open("highscores.txt", "r") **as** f:

**for** line **in** f:

line **=** line**.**strip("\n")

line **=** line**.**split(" ")

names**.**append(line[0])

scores**.**append(int(line[1]))

**except** FileNotFoundError:

**print**("No high scores file found")

The code creates the two lists to hold the high scores before opening the highscores.txt file for reading.

The for loop iterates over every line of text in the file. For each line, the program:

* Removes (strips) the newline \n character at the end of the record
* Converts the contents of the line into a list, splitting the string where a space (“ “) occurs (between a player’s name and their score)
* Appends the name to the names list and the score to the scores list

**Display the high scores**

Before the start of the round of the game, the high scores should be displayed to the player.

This code will read the values from the names and scores lists and print them to the display.

**print**("High scores")

**for** pos **in** range(len(names)):

**print**(pos **+** 1, names[pos], scores[pos])

The for loop iterates through each name in the names list and displays the player’s position, name, and score.

**Add a new score**

Once the game has asked all the questions, the scores and names lists are updated with the name of the last player and their score.

In order to save their high score, you will need the player’s name. Prompt the user to enter it during the game.

name **=** input("Whats your name? ")

This code will compare the player’s score to the current high scores and place the player’s name and score in the correct position in the lists.

position **=** 0

**for** compare\_score **in** scores:

**if** score **<** compare\_score:

position **=** position **+** 1

scores**.**insert(position, score)

names**.**insert(position, name)

scores **=** scores[:10]

names **=** names[:10]

To determine the player’s position, their score is compared against previous high score, from highest to lowest. The position variable is increased by one (moving them down the high score table) until their score is less than the high score being compared.

The player’s score and name are inserted into the corresponding lists at the correct position.

As only the top ten scores are needed, the names and scores lists are trimmed to a maximum of ten.

**Save the high scores data**

The data can be saved after the player’s has been added to the scores and names lists.

**print**("Saving high scores")

**with** open("highscores.txt", 'w') **as** f:

**for** pos **in** range(len(names)):

f**.**write(names[pos] **+** " " **+** str(scores[pos]) **+** "\n")

A record structure of name of player + a space + the score + \n is used to store the contents of the names and scores lists in the highscores.txt file.

**Testing your program**

Save and run the code a few times.

* First, try the code with no highscores.txt file.
  + Does the game create the file?
* Does the game save the name and scores correctly each time the game loops? Remember to test different scores!
* Does the high score table work as expected?

**External modules/libraries/packages**

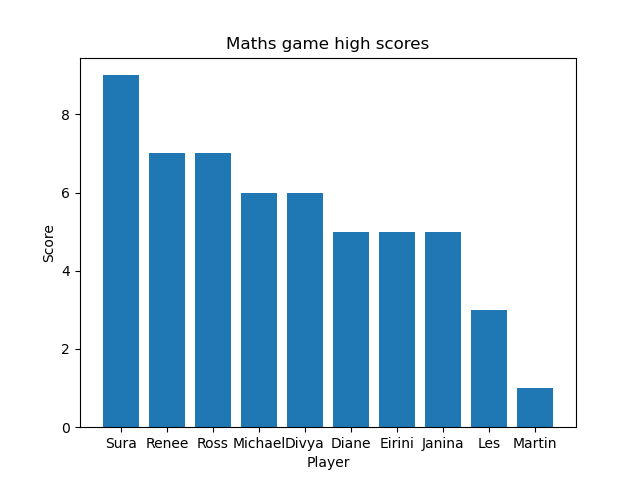
[**19 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023709/comments#fl-comments)

**One of the reasons Python is so popular is because of the amazing community that has formed around the language.**

You will rarely meet a more helpful, inclusive, and open group of people! These self-styled Pythonistas share their knowledge; produce amazing and helpful resources for learners of all levels; and gather in numerous online forums, eager for the chance to help someone with a programming problem. Their activities also help to improve the language itself.

Programmers make Python an easier language to use by producing a wide variety of modules that you can download — for free! — and install on your computer to help you perform any number of tasks.

In this **optional** step, you will install and use the Python library matplotlib to create a bar chart of the high scores data.



**Installing matplotlib**

Modules such as random or time are part of the **Python Standard Library**, and do not need to be installed.

Additional modules must be installed using their package.

How you install a package will depend on:

* The set-up of your computer
* How you are using Python
* Your access rights

The [Installing Python packages](https://projects.raspberrypi.org/en/projects/install-python-packages) guide provides advice on how to install Python packages for typical scenarios.

You will need to install the [matplotlib](https://matplotlib.org/) package to complete this practical activity.

You can test whether matplotlib has been installed by creating a small Python program to import the module.

**import** matplotlib.pyplot **as** plot

**print**("matplotlib imported")

When you run the program, you should see the message matplotlib imported. If you see an error, such as ModuleNotFoundError: No module named 'matplotlib', the module hasn’t been installed correctly. Make sure the matplotlib package is correctly installed before you move on.

**Creating a chart**

You can use matplotlib to create a chart of the data held in the highscores.txt file.

To recap, the top ten names and scores are stored in the format name + space + score + \n. The data is loaded into two lists, names and scores, by this code:

**print**("Loading high scores")

scores **=** []

names **=** []

**try**:

**with** open("highscores.txt", "r") **as** f:

**for** line **in** f:

line **=** line**.**strip("\n")

line **=** line**.**split(" ")

names**.**append(line[0])

scores**.**append(int(line[1]))

**except** FileNotFoundError:

**print**("No high scores file found")

To use matplotlib in your maths game program, you will need to import it.

**import** matplotlib.pyplot **as** plot

**Tip:** It is best practice to include all imports at the top of your program.

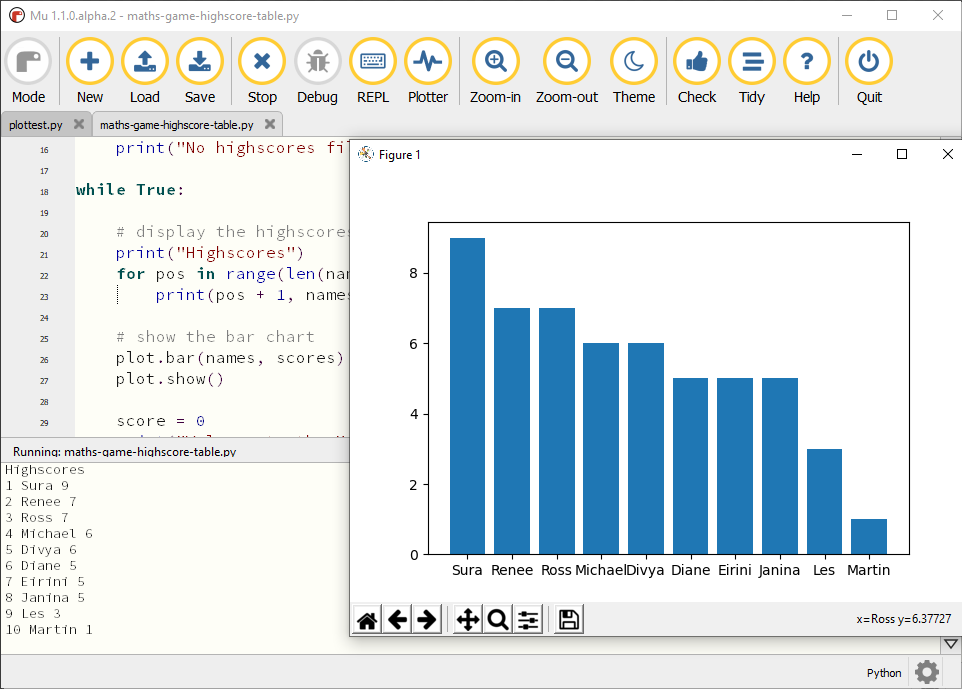
Use the matplotlib bar function to create the chart using the data in the names and scores lists.

plot**.**bar(names, scores)

Display the chart using the shot function.

plot**.**show()

Run your program to see the chart of top scores.



Before displaying the chart you can change the title and the names of the axis by setting the chart’s properties.

plot**.**bar(names, scores)

plot**.**title("Maths game high scores")

plot**.**ylabel("Score")

plot**.**xlabel("Player")

plot**.**show()

Using external Python packages can reduce the amount of work you need to do to add functionality to your programs.