# Structuring data in CSV files

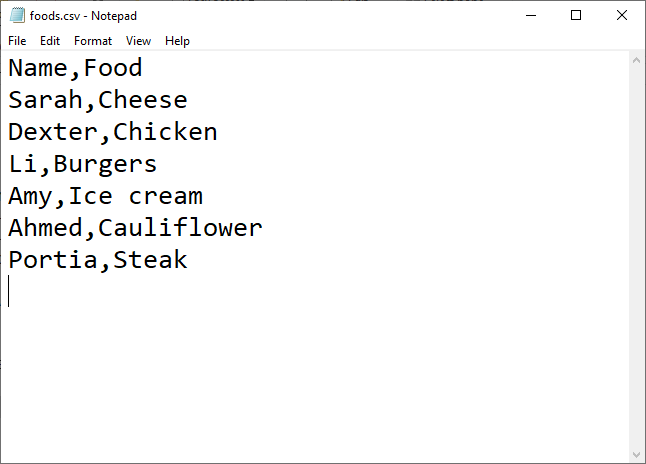
[**12 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023712/comments#fl-comments)

**Most of us have used a spreadsheet to record data. It is one of the most basic tools used for analysing financial and scientific data. But have you ever used comma-separated values (CSV) files?**

CSV files store data in a structure that can be used (read and written to) by a range of different spreadsheet software packages. Python programs can also interact with CSV files; you can use a Python program to save data in a CSV file, and later open that same CSV file in a spreadsheet. You can also use the data from a CSV file directly in Python to visualise data.

### What is a CSV file?

Here’s a CSV file opened in a text editor:



CSV files contain data structured so that a comma separates individual items in the file, and each record is on a new line of the file.

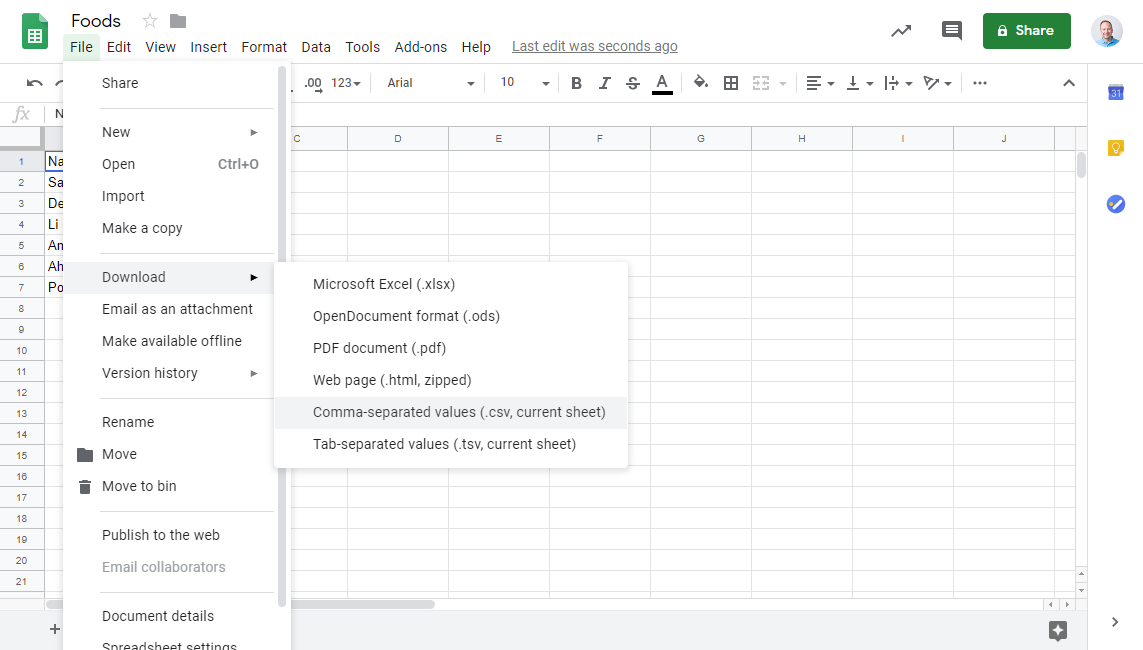
In the image above there are two columns, **Name** and **Food**, and on each row there is the name of a person and their favourite food.

In a spreadsheet application (such as Excel or Google Sheets), use this data to create a spreadsheet.

| **Name** | **Food** |
| --- | --- |
| Sarah | Cheese |
| Dexter | Chicken |
| Li | Burgers |
| Amy | Ice cream |
| Les | Lasagne |

### How to export a spreadsheet as CSV

Saving a spreadsheet to a CSV file is straightforward, so most spreadsheet applications have this feature. You can find the option in the File menu. Save the file as foods.csv.



Open foods.csv in a text editor and take a look at the contents.

You can see that the structure of the raw CSV file is the same as the data in the spreadsheet. There are field names on the first row to identify what each column refers to. Between each item on the rows there is a single comma, which is a delimiter, a way of splitting data into columns. This is used to separate the name of the person from their favourite food. For the next person, the CSV file uses a new line to identify that this is the start of a new line of data.

CSV is a common and simple data structure that works extremely well and offers the ability to store complex data in a way that is simple to understand.

# Using CSV in Python

[**24 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023713/comments#fl-comments)

**Comma-separated value files offer simple, persistent, structured storage. You can work with CSV files using standard Python file methods, but using the Python CSV library makes it easier to handle data from a CSV file.**

In the previous step you created a CSV file called foods.csv which contained the names and favourite foods of a group of people:

Name,Food

Sarah,Cheese

Dexter,Chicken

Li,Burgers

Amy,Ice cream

Last week you learnt how to read from and write to files, and this same principle can be used when working with CSV files.

### Reading the contents of a CSV file

You should recognise the code below from last week. It reads the data from the CSV file and prints all the entries.

**with** open("foods.csv","r") **as** f:

data **=** f**.**read()

**print**(data)

### Writing to a CSV file

A CSV file can be written to or appended to using the open() method just like any other file:

name **=** "Les"

food **=** "Lasagne"

f **=** open("foods.csv", "a")

f**.**write(name **+** "," **+** food **+** "\n")

f**.**close()

This example appends a new line to the CSV file that adds a new name and that person’s favourite food to the CSV file. It uses two variables, name and food, to reference the person and their food choice.

Reading and writing data to a file using the standard file methods in Python means that the data is read as a string. This means that you may need to use string slicing to select the data that you want. It is often easier to work with CSV files using the csv library.

**Reminder:** String slicing allows you to take a slice of a longer string, for example removing www. from the start of the URL www.google.com to give google.com.

### Using the CSV library to read a file

Python has a special library installed as standard which can work with CSV files. Using this library, you can write code to read from and write to CSV files with greater flexibility than using standard file handling methods.

**import** csv

**with** open('foods.csv') **as** csvfile:

favourites **=** csv**.**reader(csvfile, delimiter**=**',')

**for** row **in** favourites:

**print**(row)

This code first imports the Python CSV library. It then opens foods.csv and creates a reference to the file as csvfile. The code then reads the file using the csv.reader() method and stores the data to an object called favourites. The data in favourites is stored so that it can be iterated over row by row, with each row (a line from the original file) being output as a list. The code does this and prints each row to the REPL as a list.

**Tip:** An object is a collection of data, and functions that can act on that data.

### Using the CSV library to write to a file

You can add another row of favourite foods to the CSV file using the same CSV library. Take a look at the code below. Can you work out what each line is doing?

**import** csv

name **=** "Portia"

food **=** "Steak"

**with** open('foods.csv', mode**=**"a") **as** csvfile:

favourites **=** csv**.**writer(csvfile, delimiter**=**',')

favourites**.**writerow([name,food])

This code will open the file in a similar way as before, but with the mode set to "a" for append. This will enable the code to append new data to the file foods.csv. This matches the syntax used in the open() method.

The csv.writer() method creates an object, favourites, which you can use to write CSV data to the file. The delimiter argument is used to specify that each data item should be separated by a comma.

A new row is appended to the end of the file using the favourites.writerow(). The data for the row is passed as a list of the variables name and food.

### CSV and data types

By default, all data items held within CSV files are treated as strings. However, you can also be explicit and use " " (speech marks) to denote where data within a CSV file is a string and where it is not.

The following CSV data contains two fields: a name and an age.

"Sarah",79

"Dexter",12

"Li",28

"Amy",45

Note how the names are enclosed in " " to denote that the data is a string.

You can use csv.reader to automatically cast data in " " to strings, and data not in " " to float type, by using the quoting=csv.QUOTE\_NONNUMERIC argument.

**import** csv

**with** open('ages.csv', newline**=** '') **as** csvfile:

favourites **=** csv**.**reader(csvfile, delimiter**=**',', quoting**=**csv**.**QUOTE\_NONNUMERIC)

**for** row **in** favourites:

**print**(row)

**print**(type(row[0]), type(row[1]))

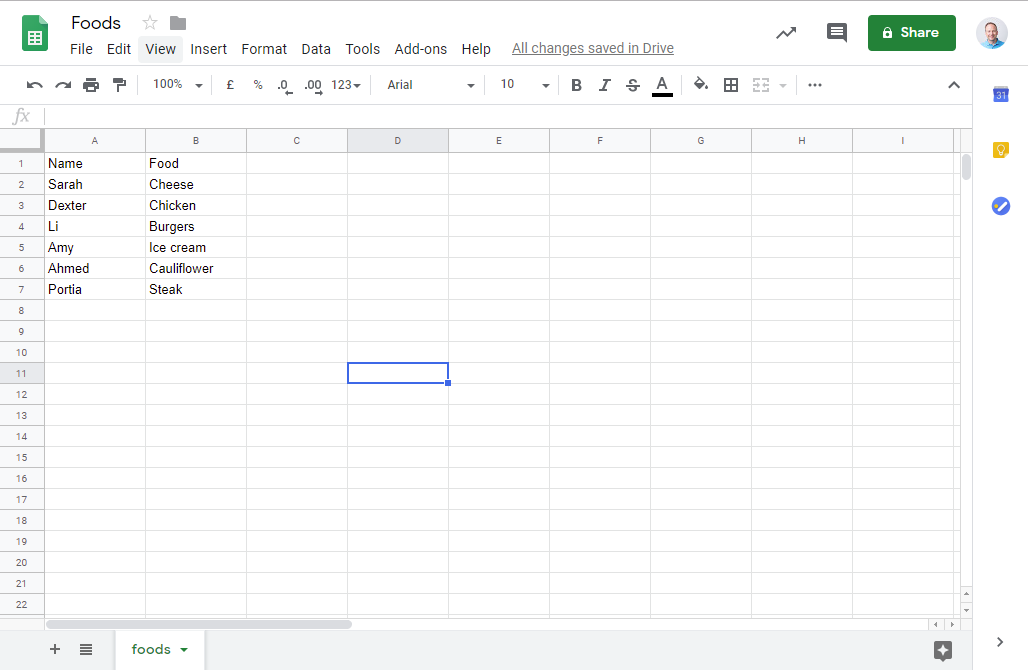
Create a CSV file using a text editor which contains a string that is enclosed in " ", and a number that is not, and create a program to read this CSV data.

# What are the limitations of using CSV?

[**5 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023714/comments#fl-comments)

**CSV is a great way to store structured data, but it has its limits. As the data gets more complex, a more complex data structure is required.**

CSV effectively stores data in a two-dimensional array; data is stored as rows of fields, just as in a spreadsheet.



For projects that only require basic data to be stored, for example a person’s name and their favourite food, a CSV works well. What if you need to store more complex data, with more links between pieces of data? For example, in a secondary school, each child will have many different classes with different teachers and will generate lots of data such as their grades and attendance records. How would you record all that data in a useful way? You would need to use something like a database, which you’ll look at next week.

CSV is a great general-purpose structure for simple string and numeric data, but if you need to store other data types, such as lists, tuples, variables, and so on, you would need to use a structure that offers the functionality required. Later this week you will look at a data structure that offers this flexibility, while still remaining human-readable: JSON (JavaScript Object Notation).

**Structuring data in your programs using dictionaries**

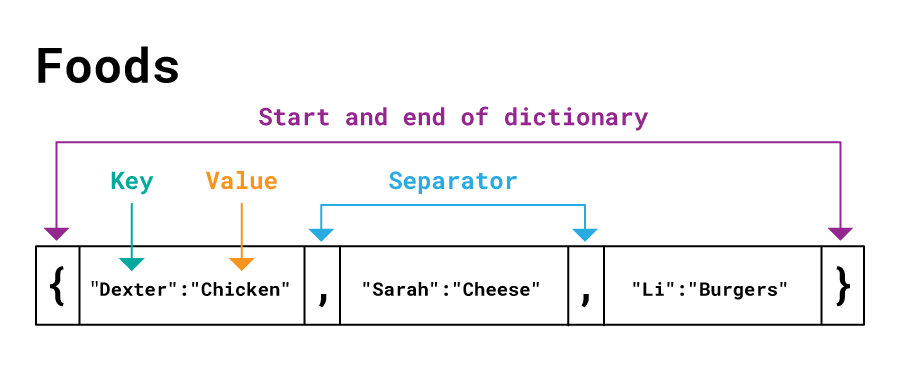
[**12 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023715/comments#fl-comments)

**As well as structuring data when it is stored, you should also consider how it is stored in your programs.**

Dictionaries are a useful data structure in Python, and a simple but powerful way of structuring data in your programs.

You may think of a dictionary in programming as being like a reference book that is used to understand the meaning of words. If you want to find the meaning of a word, you find the word in the dictionary, and then read the meaning.

In the programming world, a dictionary is a data structure which is used to store data using a key and value system, just like a paper dictionary, with the word you are looking for being the **key** and the definition being the **value**. I can store any type of object in a Python dictionary, and when presented with a key, the dictionary will return the value associated with that key. To create a Python dictionary, the entries are written with each key and corresponding value separated by a colon. Each key–value pair in the dictionary is separated by a comma:



In this step you’ll see how to read from, write to, and remove data from a dictionary so that you can use this structure in your projects.

**Storing data in a dictionary**

Here, I have created a dictionary called foods and am using the dictionary to store the favourite food of three people. In your Python editor, create this same dictionary.

foods **=** {"Sarah":"Cheese", "Dexter":"Chicken", "Li":"Burgers"}

The name of each person is the key, and if I use that key, the dictionary will tell me the value associated with that key, in this case the person’s favourite food. To see all of the people (keys) and their favourite foods (values) I can write code to print the entire dictionary.

**Reading an individual value**

There will be times when I want to see just a single person’s favourite food. How can I find out Sarah’s favourite food? Add this line to your code and then click **Run**.

**print**(foods["Sarah"])

To retrieve Sarah’s favourite food, you need to get the value which is stored for the key "Sarah" in the foods dictionary. When this code is run, you will see that Sarah’s favourite food is "Cheese".

Modify the program to print Dexter’s and Li’s favourite foods.

**Adding a new person**

To add another person and their favourite food to the foods dictionary, you can assign a new value to a new key. You do this by using the name of the dictionary foods and the key (in this case, the name of the person), in square brackets, and then assigning the value (their favourite food) to it. Add these lines to your code:

foods["Amy"] **=** "Ice cream"

**print**(foods)

This code will add "Amy" and her favourite food, "Ice cream", to the dictionary. The second line of code will print the entire dictionary to the REPL, so that you can see that Amy is there.

**Tip:** When you use a dictionary, you always retrieve values by **key**, never by position (index).

**Updating a person**

If Sarah changed her mind, how could you update her favourite food? Add these lines to your code:

foods**.**update({"Sarah":"Steak"})

**print**(foods)

This code updates a single value in the foods dictionary using the update method. This method requires the key, in this case the name "Sarah", to identify which value to change. It also requires the new value, in this case "Steak", which will replace the previously stored value. The final line of code prints the contents of the foods dictionary to check that the change has been made.

You could also update the dictionary in the same way using:

foods["Sarah"] **=** "Steak"

**Deleting a person**

What happens if Dexter decides that he no longer wants to be part of the dictionary? Add these lines to your code:

**del** foods["Dexter"]

**print**(foods)

To remove Dexter from the dictionary you use the del statement, which requires the name of the dictionary and the key to be removed. Run this code and you will see that Dexter has been removed from the dictionary.

**Why is a dictionary useful?**

Dictionaries offer a simple yet powerful structure. Think back to the quiz game in week one. The final version of that game had a high score table for ten players. The same high score system could be stored as a dictionary. In this case, the player name would be the key, and the value would be their score.

highscores **=** {"Sarah":3, "Li":2, "Anthony":1}

With the high score table structured as a dictionary like this, you could guess what this code does without knowing the previous project.

**Tip:** when initialising a dictionary with a lot of data, you might find it easier to read if you split the code over multiple lines, for example:

highscores **=** {

"Sarah":3,

"Li":2,

"Anthony":1

}

**Questions**

* **When might you want to use a dictionary using keys that are not people’s names?**
* **When may you want to use other data types (i.e. not strings or integers) as the values in a dictionary?**

**An introduction to JSON**

[**16 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023716/comments#fl-comments)

**JavaScript Object Notation, or JSON, is another structured method of storing data.**

JSON is a lightweight data interchange format that is easy for humans to read and write, while remaining easy for machines to parse and generate.

**Tip:** **Parsing** is the process of data being split into its component parts, for checking or other processing.

JSON is regularly used when working with application programming interfaces (APIs). An API is an interface that allows different programs to talk to each other. APIs are often provided by internet services, such as Twitter, Facebook, and NASA, and JSON provides a standard format for those services to exchange data with your programs.

JSON is remarkably similar to Python’s dictionary data type, in that it uses a key–value pair to store data.

The following Python dictionary contains details about Neil Armstrong, the first person on the moon.

data **=** {

"Name": "Neil Armstrong",

"Age": 82,

"Hobbies": ["Aircraft design", "Fishing", "Astronaut"]

}

If the data in this Python dictionary were converted into a JSON format, it would be structured like this:

{"Name": "Neil Armstrong", "Age": 82, "Hobbies": ["Aircraft design", "Fishing", "Astronaut"]}

Python’s json module allows you to convert dictionaries to JSON, and vice versa. Using this module simplifies the code required to persist structured data using JSON.

**Storing data as JSON**

Create a new Python program called store-json.py. In this program, import the json module and create a new dictionary that contains information about yourself or someone you admire:

**import** json

data **=** {

"Name": "Neil Armstrong",

"Age": 82,

"Hobbies": ["Aircraft design", "Fishing", "Astronaut"]

}

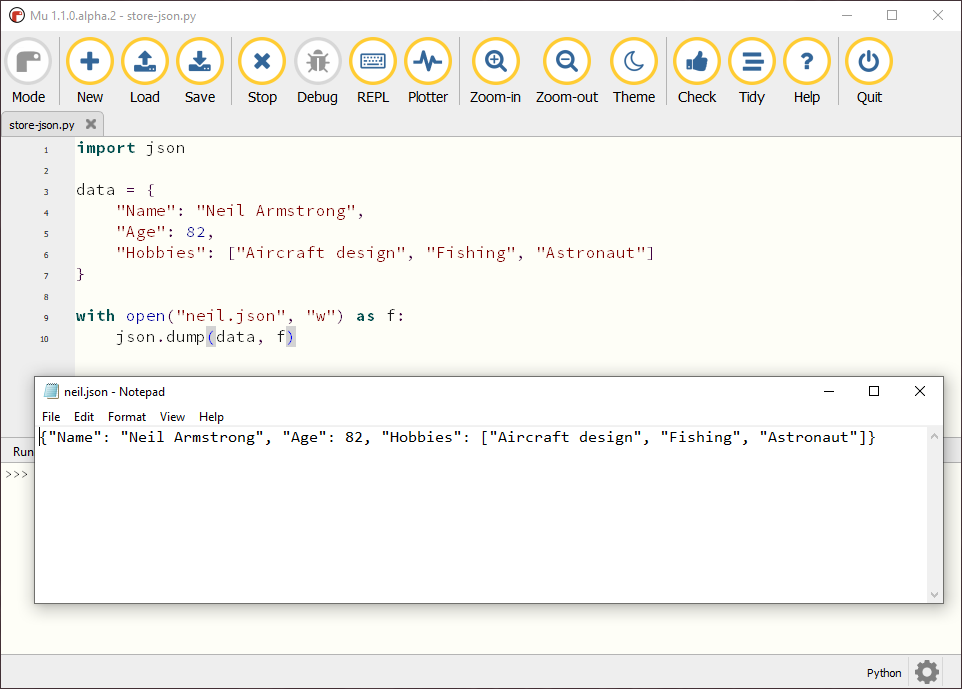
You can save the data dictionary to a JSON file by opening a new file in write "w" mode and using the json.dump function:

**with** open("neil.json", "w") **as** f:

json**.**dump(data, f)

In the example above, the JSON data will be written to a file called neil.json.

Run the program and then open the file using a text editor. You should see data from your dictionary stored in a JSON format.



**Loading JSON data**

Try loading the JSON data you just stored back into the program, using the following code:

**import** json

**with** open("neil.json", "r") **as** f:

data2 **=** json**.**load(f)

**print**(data2)

This opens the JSON file in read "r" mode, and uses the json.load method to read the JSON file.

You should see that data2 is a dictionary. You can obtain the individual values from this dictionary by using the relevant keys, as you saw earlier in the course, for example:

age **=** data2["Age"]

**print**(age)

You may have noticed that I have chosen three different data types to be stored in the JSON file and dictionary. JSON is much more flexible than CSV, in that you can store many different data types, mixing integers, strings, and lists, and can even store another dictionary. Even with this complexity, JSON still provides a syntax that is easy to read and write and can be used to store complex data.

JSON is an amazing tool for structuring data. Over the next few steps you will see how to use it to make a game with persistent structured storage.

## Other structured data formats

**You are surrounded by structured data, in your home and on your commute to work. Electronic signs at the train station, bus station, and on the motorway all receive data in a structured format.**

In your home you will turn on the television and use the programme guide to find your favourite programme. In order to find it, you will have to look at a certain time and channel. The data used to make that guide is structured to ensure that different broadcasters all provide data in a standard format that can be used in the schedule. If you have a smart energy meter, this will also use structured data to display how much energy your home is using in graphs and monetary values. Your home is built using materials that are measured and ordered using structured data.

For a more detailed example, at the train station the data that the information screens present is structured in such a way that you can find out when your train will depart, and the platform it will depart from. This data is structured using JSON.

The data in the train information JSON file is broken down into sections which hold the information for stations, platforms, times, etc. It is structured in such a way that the receiving device can extract the data and display it on the screens around the station.

In the computing world there are many other structured data formats, such as:

* HTML, Hypertext Markup Language, used to create web pages
* GIF, Graphical Interchange Format, a highly compressed image format commonly used for animations
* JPEG, Joint Photographic Experts Group, a file format for images providing high quality and small file sizes

Some of these structured data formats are specialised, such as the JPEG and GIF image formats. In the case of JPEG and GIF, these data structures were created by corporations/organisations so that images could be exchanged between different systems. There are many other specialised data formats that are proprietary in nature and are typically made to be secret or to protect the intellectual property of the creators. This means that the data cannot be readily understood or modified without breaking a licence agreement.

Other data structures, such as HTML and JSON, are generic data structures that were developed to be multipurpose and to be reused in many different applications, while still being readable by humans. This enables a user to read and modify the data based upon their needs, without any licences being breached. Examples of generic data structures are HTML, which is used to create content on the World Wide Web, and JSON, which is used to exchange data.

### Examples of data structures

The JPEG file format is a binary file that stores the large of amounts of data generated by an image. It has segments in the code to identify the start of an image, and the end. In the main body there are markers which are used to instruct the computer to build the image. All of this structure is hidden from view.

To create an HTML document there is a start, identified by <html>, and a corresponding </html> to end. Inside the document the structure of HTML uses tags to surround content for the page. Each tag is opened and closed, and the content is placed between these tags, with the tags explaining how the content is to be displayed. This format, combined with the fact that HTML files are text files, mean that the contents can be understood by human beings as well as machines. Here’s an example:

<html>

<body>

<h1>Hello World</h1>

</body>

</html>

All of these structured data formats share the same aim: to ensure data can be transferred consistently between machines across the globe.

# Creating an RPG

[**15 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023718/comments#fl-comments)

# Saving game progress

## Creating an RPG

**Many role-playing games (RPGs) put you in the shoes of a brave adventurer who is battling a great evil, saving the world, or maybe trying to stop a pandemic. In this step you will start to create your own game using Python, making use of JSON to store persistent data.**

RPGs have been popular ever since Dungeons and Dragons first appeared in 1974. This type of game generally has characters with attributes such as:

* Health
* Inventory of items
* Character name
* Location

Each one of these attributes is a piece of data that is unique to a character. In the original RPGs, this data was recorded on a character form that the player used during their game. Fast-forward to the 21st century and as well as pen and paper adventures there are computer RPGs, whose code saves the characters’ attributes and reloads them each time the game starts.

### The adventure awaits you!

Here is the code for an adventure game in which the player must make it to the garden with a key and a potion in order to win the game. Every time the player moves, their health reduces by one point.

Read through the program and predict how you think the game works.

**Tip:** This is a lot of code and it may seem confusing, but by using the comments in the code, tracing the execution, and playing the game you should be able to follow it.

**def** **showInstructions**():

*# Print a main menu and the commands*

**print**("""

RPG Game

========

Get to the Garden with a key and a potion.

Avoid the monsters!

You are getting tired; each time you move you lose one health point.

Commands:

go [north | south | east | west]

get [item]

""")

**def** **showStatus**():

*# Print the player's current status*

**print**("---------------------------")

**print**(name **+** " is in the " **+** currentRoom)

**print**("Health : " **+** str(health))

*# Print the current inventory*

**print**("Inventory : " **+** str(inventory))

*# Print an item if there is one*

**if** "item" **in** rooms[currentRoom]:

**print**("You see a " **+** rooms[currentRoom]["item"])

**print**("---------------------------")

*#-# CODE WILL BE ADDED HERE IN THE NEXT STEP #-#*

*# Load data from the file*

*# Set up a new game*

name **=** None

health **=** 5

currentRoom **=** "Hall"

inventory **=** []

*# A dictionary linking a room to other room positions*

rooms **=** {

"Hall" : { "south" : "Kitchen",

"east" : "Dining Room",

"item" : "key"

},

"Kitchen" : { "north" : "Hall",

"item" : "monster"

},

"Dining Room" : { "west" : "Hall",

"south" : "Garden",

"item" : "potion"

},

"Garden" : { "north" : "Dining Room" }

}

*# Ask the player their name*

**if** name **is** None:

name **=** input("What is your name, Adventurer? ")

showInstructions()

*# Loop forever*

**while** True:

showStatus()

*# Get the player's next "move"*

*# .split() breaks it up into an list array*

*# e.g. typing "go east" would give the list:*

*# ["go","east"]*

move **=** ""

**while** move **==** "":

move **=** input(">")

move **=** move**.**lower()**.**split()

*# If they type "go" first*

**if** move[0] **==** "go":

health **=** health **-** 1

*# Check that they are allowed wherever they want to go*

**if** move[1] **in** rooms[currentRoom]:

*# Set the current room to the new room*

currentRoom **=** rooms[currentRoom][move[1]]

*# or, if there is no door (link) to the new room*

**else**:

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

*# If they type "get" first*

**if** move[0] **==** "get" :

*# If the room contains an item, and the item is the one they want to get*

**if** "item" **in** rooms[currentRoom] **and** move[1] **in** rooms[currentRoom]["item"]:

*# Add the item to their inventory*

inventory **+=** [move[1]]

*# Display a helpful message*

**print**(move[1] **+** " got!")

*# Delete the item from the room*

**del** rooms[currentRoom]["item"]

*# Otherwise, if the item isn't there to get*

**else**:

*# Tell them they can't get it*

**print**("Can't get " **+** move[1] **+** "!")

*# Player loses if they enter a room with a monster*

**if** "item" **in** rooms[currentRoom] **and** "monster" **in** rooms[currentRoom]["item"]:

**print**("A monster has got you ... GAME OVER!")

**break**

**if** health **==** 0:

**print**("You collapse from exhaustion ... GAME OVER!")

**break**

*# Player wins if they get to the garden with a key and a potion*

**if** currentRoom **==** "Garden" **and** "key" **in** inventory **and** "potion" **in** inventory:

**print**("You escaped the house ... YOU WIN!")

**break**

*#-# CODE WILL BE ADDED HERE IN THE NEXT STEP #-#*

*# Save game data to the file*

Create a new program called rpg.py and copy the [RPG code](https://rpf-futurelearn.s3-eu-west-1.amazonaws.com/Programming+103+(Data)/Code/rpg.py) from above into the program.

Play the game a few times and get a feel for how the game works. Was your prediction of how the game works correct? Is your understanding now different?

**Saving progress in a game is a form of persistence. In modern games, this happens seamlessly as the player progresses. In this step you will add this feature to your RPG (role-playing game) program.**

The name, location, health, and inventory for the player is reset every time your program starts. You need to create a data structure to store this information, which can be saved to an external file and read when the game restarts. This will enable the player to start where they left off.

**Adding the features**

In this task you will edit the code shown in the previous step to add the **save** and **load** features.

The first change in the code is to import the JSON library, as you will be using it to store the player data in a structured format.

**import** json

After importing the JSON library, you should open a gamedata.json file and retrieve the game data.

You will again use a try … except exception handler. Using this handler, you can create something similar to the quiz game, where if there was a file containing the data needed by the game, it would automatically load the data at the start. If, instead, this was the first time the player was playing the game, it would create a new file and save the data to it.

Locate the section of the program where the data should be loaded.

*#-# CODE WILL BE ADDED HERE IN THE NEXT STEP #-#*

*# Load data from the file*

Add the code to open the file and retrieve the game data.

**try**:

**print**("Retrieving player details")

**with** open("gamedata.json", "r") **as** f:

gamedata **=** json**.**load(f)

name **=** gamedata["playername"]

health **=** gamedata["playerhealth"]

currentRoom **=** gamedata["playercurrentRoom"]

inventory **=** []

You should be able to identify the lines of code which:

* Start the exception handler (trying to run the code)
* Open the gamedata.json file
* Load the JSON data into a dictionary
* Retrieve the player’s name, health, and current room from the dictionary
* Create an empty inventory for the player, forcing the player to search for the key and potion every time in order to beat the game

An exception is raised if no gamedata.json file is found.

Add the code to capture this exception and set up a new game only if the file is not found.

**except** FileNotFoundError:

**print**("No previous game found. Starting a new game.")

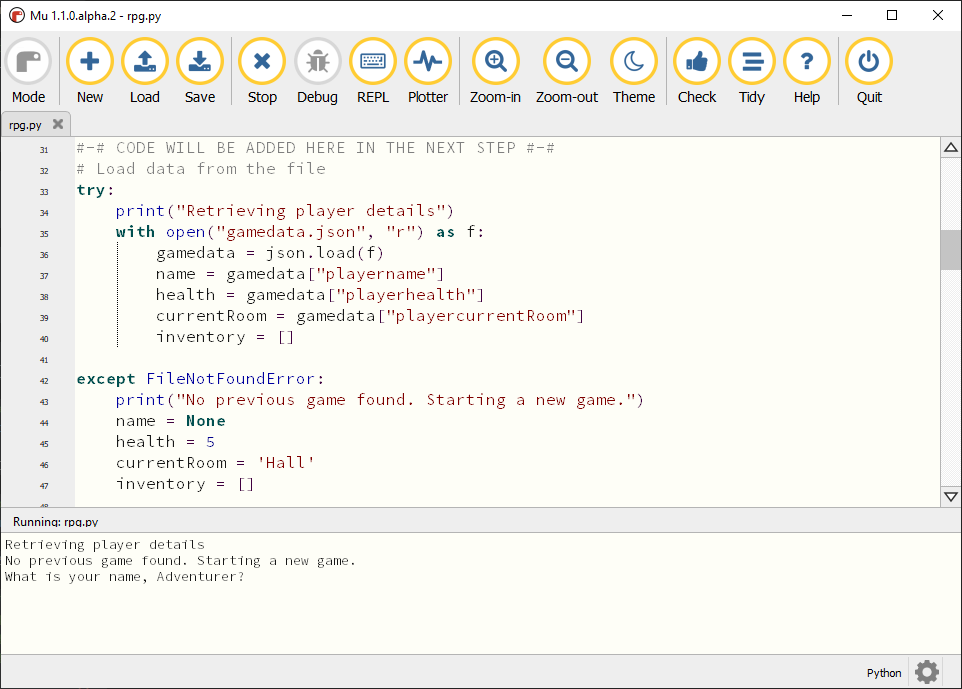
*# Set up the game*

name **=** None

health **=** 5

currentRoom **=** "Hall"

inventory **=** []



**Note:** The code to set the four player attributes (name, health, currentRoom, and inventory) to their default values is now indented under the exception handler.

**Saving the player’s attributes**

Every time the player gets an item or moves to a new location, their attributes should be written to a JSON file called gamedata.json.

These attributes are stored in a dictionary using appropriate data types: as strings (name and current room) and an integer (health).

Locate the section of the program where the game data should be saved.

*#-# CODE WILL BE ADDED HERE IN THE NEXT STEP #-#*

*# Save game data to the file*

Create a new dictionary and populate its values with the player’s name, health, and current room.

gamedata **=** {

"playername": name,

"playerhealth": health,

"playercurrentRoom": currentRoom

}

The json.dump() function is used to save the data into the gamedata.json file.

**with** open("gamedata.json", "w") **as** f:

json**.**dump(gamedata, f)

**Test the game**

Save and run the game a few times.

* What happens the first time the game is run?
* What happens when the game is played again?

**What items is your player carrying?**

There is a data structure in your game to save the player’s attributes such as their name, health, and current location, and it looks like this:

gamedata **=** {

"playername": name,

"playerhealth": health,

"playercurrentRoom": currentRoom

}

But what it doesn’t contain are the items needed to complete the game: the potion and the key. How can you add a backpack to the code, so that the player can carry those items?

* How could you store the inventory inside the gamedata dictionary?
* How could you remove an item from the room, if a player already has it in their inventory when the game starts?

**Tip:** A list can be stored as a value inside a dictionary, enabling data to be nested within another data type.

# Using JSON with a public API

[**11 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023720/comments#fl-comments)

**JSON is not just a format for storing data, but also one for transmitting data. As mentioned earlier, JSON is commonly used with APIs (application programming interface) on the internet.**

In this **optional** step, you will install and use the Python library requests to download JSON data from NASA.

### Installing requests

You will need to install the [requests](https://requests.readthedocs.io/en/master/) package to complete this practical activity. Installing requests will be similar to how you installed matplotlib earlier in the course.

You can refer to the [Installing Python packages](https://projects.raspberrypi.org/en/projects/install-python-packages) guide if you need advice on how to install Python packages.

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

**import** requests

**print**("requests imported")

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

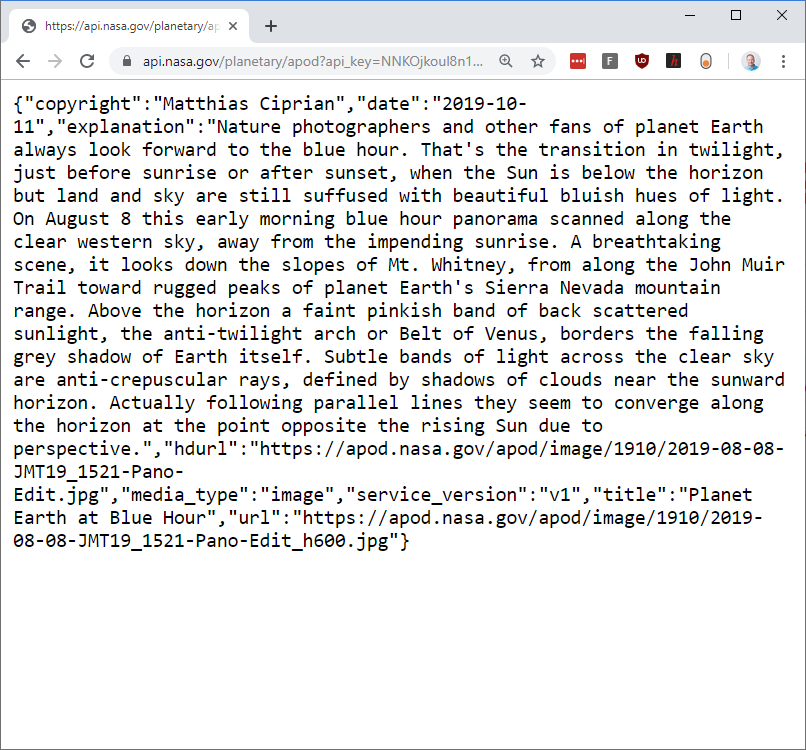
### Using the NASA API

NASA provides an [API](https://api.nasa.gov/) that can be used to extract scientific and astronomical data and images taken during various missions. NASA offers this data in JSON format for free via a public API, and you can work with that data using Python.

The NASA Astronomy Picture of the Day (APOD) data can be downloaded from this URL:

<https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY>

When you open the URL in your browser, you should recognise that the data returned is in a JSON structure.



You will now use Python to download this data, parse it to extract the image URL, and show the picture.

Create a new Python program, import the requests module, and use its get function to open the NASA APOD URL.

**import** requests

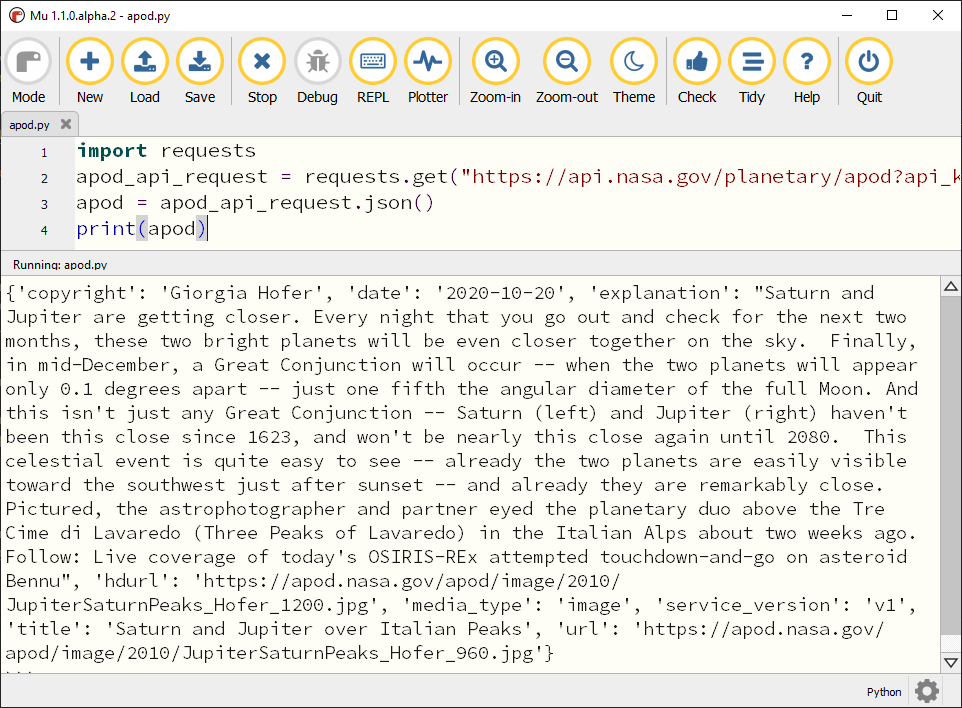
apod\_api\_request **=** requests**.**get("https://api.nasa.gov/planetary/apod?api\_key=DEMO\_KEY")

You can now convert the JSON data into a dictionary.

apod **=** apod\_api\_request**.**json()

When you print the data, you should see multiple pieces of data in the response, for example copywrite, explanation, title, and url.

**print**(apod)



As with any JSON data, you can extract individual elements, such as the picture’s url, using the key:

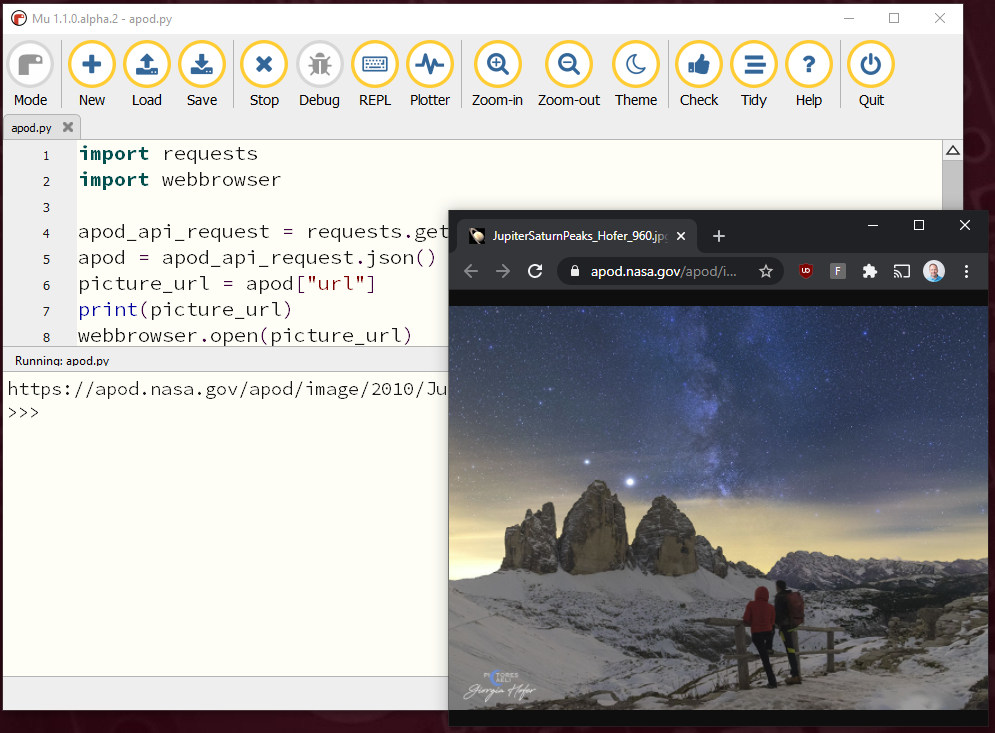
picture\_url **=** apod["url"]

**print**(picture\_url)

You can use the standard Python webbrowser module to open the picture’s URL and display it.

**import** webbrowser

webbrowser**.**open(picture\_url)



**What do you think the advantages are of using a structured data format such as JSON to pass data between different programs through APIs?**

**Validating user input**

[**27 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023721/comments#fl-comments)

**Data validation is the process of checking that data coming into a program is valid and appropriate and can be dealt with by your program.**

Without data validation, programs may raise errors, not act in the way expected, and be open for abuse.

For example, when buying items online, the store will check that the number of items being purchased is:

* A number
* Not negative
* Less than the stock available

If they didn’t check this, it could cause the store to stop working, order items which cannot be delivered, or create issues with their stock management software.

Data for important tasks cannot be instantly trusted; it needs validation to ensure that it is fit for purpose.

In the next two steps you will learn about how to validate data from users and file input.

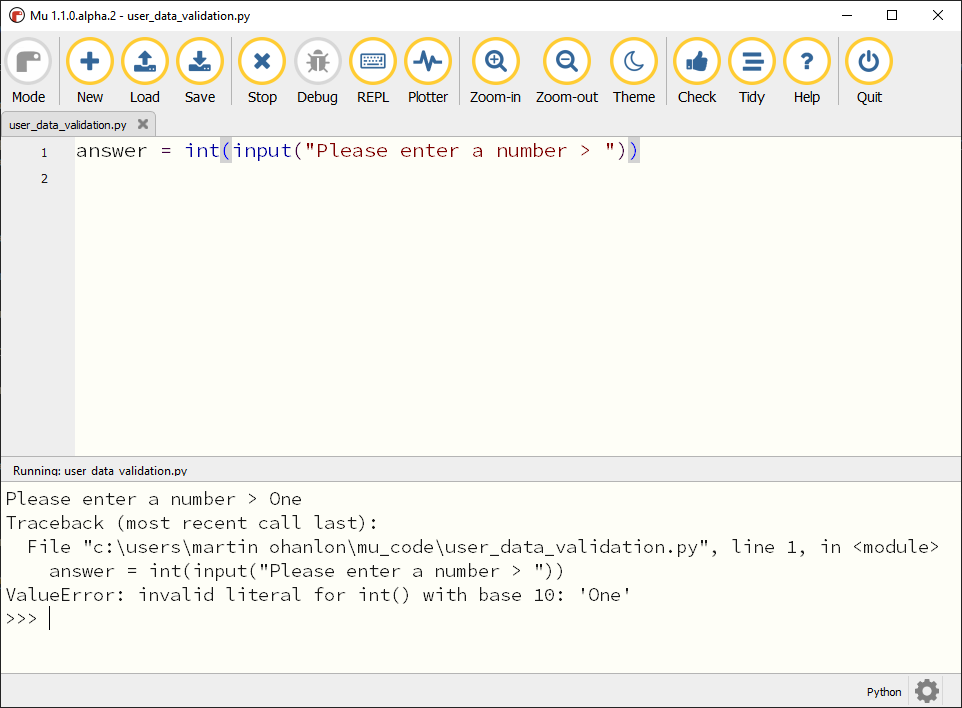
**Validating the data type**

To ensure that data input from users is of the right type and appropriate, it needs to be validated.

In week one you created a maths game which asked the user to enter a number, using code similar to this:

number **=** int(input("Enter a number > "))

If you run this program and enter text and not a number, Python will raise a ValueError and the program will stop.



To validate that the data input by the user is of the correct type, you can use exception handling to capture the ValueError and continue. For example:

**try**:

number **=** int(input("Enter a number > "))

**except** ValueError:

**print**("This was not an integer")

While this approach would stop the program from raising an error and stopping, the user only has one opportunity to enter the correct value, and the program would still be likely to crash, as a valid number had not been captured. A more sophisticated solution is often needed that validates a user’s input and ensures that a correct value is entered before continuing.

Review the code below and predict what you think it will do:

**def** **enter\_an\_integer**():

number **=** input("Enter a number > ")

valid\_integer **=** False

**while** **not** valid\_integer:

**try**:

number **=** int(number)

valid\_integer **=** True

**except** ValueError:

number **=** input("Invalid. Enter a number > ")

**return** number

valid\_number **=** enter\_an\_integer()

**print**("Thank you")

Run the program and test it with the following scenarios:

* Entering a valid number
* Repeatedly entering an invalid number
* Entering a decimal number/float e.g. 1.23

By creating a function which uses a while loop to continually test whether the user’s input is an integer, only valid integer numbers will be returned.

Using the same approach, can you create another function which validates that the user has entered a float?

**Validating the data contents**

After validating that a value is of the right data type, you may also need to check that the value is *appropriate*. This may include checking that the value input is within a range (for example, between 1 and 10) or one of a finite number of options (for example, months of the year). This type of validation is known as *bounds* or *range checking*.

In the example below, a user is asked to enter their age. Any values below 0 and above 117 are noted as invalid.

age **=** int(input("How old are you? "))

**if** age **<** 0 **or** age **>** 117:

**print**("That is not a valid age")

Validating someone’s age may be different depending on the scenario. It may be that the user must be over a certain age before they can use your program. Setting an upper age range is more complicated, as you don’t wish to penalise anyone for their age, but as higher numbers are entered, it becomes increasingly unlikely that the value is correct.

I used the upper value of 117, as there is no one alive older than 117 today. Validating above this very high age only benefits a small number of individuals, whereas it provide no protection for the larger number of 17-year-old’s who may accidentally entering the 117. Careful thought should be put into determining validation bounds.

It is also common to need to check that data input is one of a series of options, for example a day of the week. The program below uses a list to denote the possible options and then checks that the data input is one of those options:

days\_of\_the\_week **=** ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

day **=** input("What day is it? ")

**if** day **not** **in** days\_of\_the\_week:

**print**("This is incorrect")

**Note:** The condition day not in days\_of\_the\_week will return True if the value of day is *not* contained in the list days\_of\_the\_week.

**Validating a date**

For this challenge, use the techniques described in this step to create a program that will prompt the user to enter:

* The year
* The month as a word (January to December)
* The day in the month (1 to 31)

You program should:

* Validate that the user input has the correct data types
* Use *range* checking to ensure the month is valid and that the day in the month is between 1 and 31

As an optional challenge, consider how you would validate the day in the month based on the month entered.

**Validating file data**

[**9 comments**](https://www.futurelearn.com/courses/programming-103-data/7/steps/1023722/comments#fl-comments)

**The data held within files should also be validated before it is used, as it may have been corrupted, either through errors or intentionally.**

In this step you will modify your RPG program to validate the data loaded from the JSON file before starting the game.

**Why do you need data validation in a game?**

For a long time, video games have had cheat codes; some provide more lives, special powers, or invulnerability; others unlock special costumes and items. The makers of games add these features for fun, but sometimes players add their own by hacking the code to gain an unfair advantage.

In your RPG, the player has health which decreases by one each time they move. The original amount of health given to a player is stored in the variable health and it is set to the value of 5 at the start of the game.

Remember that that the player’s health is checked during the game:

**if** health **==** 0:

**print**("You collapse from exhaustion ... GAME OVER!")

**break**

What would happen if the health variable contained a negative number?

Here, I have edited the gamedata.json file in a text editor so that the value of key playerhealth is -1:

{"playername": "Player", "playerhealth": **-**1, "playercurrentRoom": "Garden", "inventory": ["key", "potion"]}

**Note:** Manually updating the value stored for the playerhealth key is similar to how players can hack a game to cheat.

If you were to run the code now, the player would never run out of health! Why is that?

The conditional statement only checks the player’s health to see if the player has reached 0, and so the game only ends if their health is at exactly 0.

You should validate that the player’s health is greater than -1 when loading the data.

This code snippet will validate the player’s health, and if they have cheated, it will delete the gamedata.json file and reset the game:

**import** os

**if** health **<** 0:

os**.**remove("gamedata.json")

**print**("You have cheated; game data delete!")

name **=** None

health **=** 5

currentRoom **=** "Hall"

inventory **=** []

Add this functionality to your RPG program so that a cheater is detected.

**Note:** The Python os module is used to remove (delete) the file.

Use a text editor to modify the gamedata.json file, change the playerhealth value, and check your validation.

In what other ways could the health value be manipulated to give the player an advantage?

* How could the data validation be made more robust?
* In what other ways can a player cheat, and how can you defend against these ways?