



University  
of Exeter

## COURSEWORK SPECIFICATION

COMM109 – Programming with Python

Module Leader: Dr Phil Lewis

Academic Year: 2024/25

---

Title: **Coding Wordle**

Submission deadline: **midday 12pm Tues 10th December**

This assessment contributes **60%** of the total module mark and assesses the following **intended learning outcomes**:

- Implement software to solve a given problem that utilises the control structures and data structures available in the Python language.
- Understand how to build applications that make use of Python libraries as required by the problem.
- Select appropriate design patterns to achieve specific objectives within a software project.
- Use industry-standard approaches for testing and deploying software.
- Formulate a software design to a problem from a given field or industry beyond computer science or data science.

This is an individual assessment, and you are reminded of the University's regulations on collaboration and plagiarism. You must avoid plagiarism, collusion, and any academic misconduct behaviours. Further details about academic honesty and plagiarism can be found at <https://ele.exeter.ac.uk/course/view.php?id=1957>.

**This is an individual project.**

You may not work with others on code, or collaborate with fellow students.

**This assessment is classed as AI-prohibited.**

You are not permitted to use Generative AI to write any content or code towards this assignment.

**All submitted code must be your own work.**

Submitted files will be checked for similarities to flag cases of plagiarism / collusion / use of Generative AI.

Where there is a submission that is flagged as suspicious, we reserve the right to viva students by asking them to reproduce aspects of the work under controlled conditions, and initiating further action if we are not satisfied that code you have submitted is your own work.

If you include "borrowed" code you risk submitting code that another student also submitted, which will be detected as collusion (even if variables / functions are renamed).

## Overview

In this assessment you will work to write code to enable a user to play the word guessing game of Wordle.

You can play and learn the game rules and mechanics on the official Wordle site which is part of the New York Times website:

<https://www.nytimes.com/games/wordle/index.html>

# Wordle

Get 6 chances to guess a 5-letter word.

## How To Play

- Each guess must be a valid 5-letter word.
- The colour of the tiles will change to show how close your guess was to the word.

## Examples

W	O	R	D	Y
---	---	---	---	---

W is in the word and in the correct spot.

L	I	G	H	T
---	---	---	---	---

I is in the word but in the wrong spot.

V	O	G	U	E
---	---	---	---	---

U is not in the word in any spot.

## Notes:

The Wordle word is selected at random from a list of common 5-letter words. The guesses made by a user must also be a valid 5-letter word.

If the player correctly guesses the word within six attempts, they win the game.

If the player has failed to guess the word after six attempts, they lose the game and the correct word is revealed.

# INSTRUCTIONS

In this assessment you must write Python code that can run the game of Wordle.

Example code to run the user interface has already been written as is available in file **wordle\_frontend.py**. The code in this file defines functions that display the game grid and allow the user to input commands and guesses. This file must not be edited.

You must write the corresponding backend functions (as defined in this document) in a file named **wordle\_backend.py**. These should process the user inputs and game logic such that, when combined with the frontend user interface functions, the game of Wordle can be played.

In addition to this file you must also create submit a set of files containing code to test and validate the functions you have written (as described on p10).

When your **wordle\_backend.py** code is working you should be able to test the game functionality using:

- **play\_wordle.py** to run game in the VS code terminal via a text interface
- **play\_wordle.ipynb** to run game in a Jupyter notebook via a graphical interface

You must submit the following items:

- **wordle\_backend.py** file
- set of **test\_.py** files
- a completed **AI declaration** saved in PDF format

You must save all these into a single zip file, upload via the ELE module page.

When writing the game code the information about the state of the current game must be stored in a Python dictionary named **game\_state**. This must be structured as below:

**Structure to be used for game\_state dictionary**

key (str)	object to be stored
secret_word	<b>str</b> stores word to be guessed
word_length	<b>int</b> stores length of secret word
max_guesses	<b>int</b> stores the max number of guesses that can be made
grid	<b>list</b> stores the letters entered in the wordle grid using a nested list, so that each item in the inner list stores an individual character of the guesses made.
colour_grid	<b>list</b> stores the colour formatting for the wordle grid, using a nested list, so that each item in the inner list stores an individual character representing the square colour: <b>w</b> for white, <b>g</b> for green, <b>y</b> for yellow and <b>-</b> for grey
i	<b>int</b> stores the current "cursor" row position (i.e. index value identifying the row for the next character entry in the wordle grid)
j	<b>int</b> stores the current "cursor" column position (i.e. index value identifying the column for the next character entry in the wordle grid)
status	<b>str</b> stores the current state of the game as a string either "playing" "won" or "lost"
wordle_words_file	<b>str</b> stores the name of the file that contains the set of words that are allowed to be the secret word
all_words_file	<b>str</b> stores the name of the file that contains all words in the English dictionary used to check allowable guesses (this includes different length words for non-standard grid sizes)
popup_queue	<b>list</b> this list is used to queue messages that should be displayed by the user interface. It should be initialised to an empty list.
<b>Additional Entry only required if attempting the optional tasks</b>	
active_letters	<b>dict</b> stores which letters are active and inactive using True / False values. At the start all letters are active with True value. Guessed letters that are not in the secret word updated to be False. i.e. {"A": True, "B": True "C": False .... }

You may if you wish add additional entries into the **game\_state** dictionary (e.g. it may be sensible to store the list of valid words for guesses rather than reloading it from the file).

## Part A. Core Functionality

10 marks

Write function: `initialise_game()`

**arguments:**        `all_words_file,`  
                      `wordle_words_file,`  
                      `secret_word,`  
                      `max_guesses,`  
                      `word_length`

**returns:** `game_state` dict

### *Requirements:*

The function arguments should make use of the default values as given:

<code>wordle_words_file</code>	<code>None</code>
<code>secret_word</code>	<code>None</code>
<code>max_guesses</code>	<code>6</code>
<code>word_length</code>	<code>5</code>

The function should create and fill the **game\_state** dictionary with entries suitable for starting a game (see the appendix to see an example showing how the different entries should be initialised).

If the **secret\_word** argument is defined it should be used as the hidden word.

If the **secret\_word** argument is left unfilled, then a suitable word should be selected at random from the words contained in the **wordle\_word\_file**

#### 4 marks

Write function `reset_game()`

**arguments:**            `game_state`  
                         `secret_word`

**returns:** N/A

#### *Requirements:*

This function should be able to reset the items in the **game\_state** dictionary so that a new game can be played. It should allow an optional argument **secret\_word** with default value **None**. If the secret word argument is not defined a new word will be chosen at random from the words contained in the **wordle\_word\_file**

**6 marks**

Write function `validate_guess()`

**arguments:** `game_state`

**returns** boolean True / False value

*Requirements:*

This function should take the current guess and check that it is valid by ensuring it is:

- i) the right length;
- ii) present within the word list found in the `all_words_file`



## 10 marks

Write function `process_guess()`

**arguments:** `guess`,  
`secret_word`

**returns:** **list** containing the colour codes for the guessed letters:

### *Requirements:*

There should be one entry in the list for each position in the guess, with the following values that depend upon whether the letter is in the secret word and where it appears.

In the case of multiple characters occurring in the secret word and/or guess the processing should colour letters in the order below e.g. if the word was "bears" the guess was "stirs" the last "s" would be marked green, and the first "s" letter marked grey.

You should explore the behaviour of the real game if you are unsure how different cases of duplicate characters are handled.

entry	signifies	colour
'g'	correct letter in the right position	green
'y'	correct letter in the wrong position	yellow
'-'	incorrect letter	grey

Example of returned list: `["g", "g", "-", "-", "y"]`

## 15 marks

Write function `process_command()`

**arguments:** `game_state`,  
`command`

**returns:** `boolean`

### Requirements:

This function should take a user command passed as a **str** and update the **game\_state** accordingly.

command	action
single letter (upper or lower case <b>A</b> to <b>Z</b> )	the letter should be added to the end of current guess unless there is no space in the current guess (the grid will be unchanged in this case).
keyword " <b>delete</b> " or the symbol "-"	remove the last letter that was entered in the current guess, and update the cursor position <b>j</b> accordingly
keyword " <b>clear</b> " or the symbol "_"	clear all letters that are entered into the current guess, and update the cursor position <b>j</b> accordingly
keyword " <b>enter</b> " or the symbol "+"	the user guess should be checked. If it is valid, it should be processed so it is compared to the secret word. The <b>colour_grid</b> should be updated to reflect the accuracy of the guess, and the cursor coordinates <b>i</b> and <b>j</b> should update to reflect that a new guess will be started. If the game is complete (word is guessed or all guesses have been made) then the <b>status</b> entry is updated to " <b>win</b> " or " <b>lose</b> ". However the grid is left as it is (i.e. does not reset automatically).
keyword " <b>restart</b> " or the symbol "!"	resets the game grid and <b>game_state</b> dictionary so it starts a new game with a new random secret word from the <b>wordle_word_file</b> and <b>status</b> entry set to " <b>playing</b> "

After the command has been processed, the function should return a value that indicates whether the game has ended returning:

<code>True</code>	if the game has not ended
<code>False</code>	if the game has ended

Note when writing this function you may find it easier to construct separate functions to perform each option, and use the **process\_command()** function to call the appropriate one.

## Part B. Messages and Exceptions

### 5 marks

#### Game messages

Add the following function to your **wordle\_backend.py** file that can queue a "pop-up" message for the front end to display:

```
def show_popup(game_state, content, mode="text"):
    game_state['popup_queue'].append((content, mode))
    return
```

Ensure that the **process\_command()** function triggers the **show\_popup()** function so that the front end will display the following messages at the appropriate time.

"You won!"

"You lose! The word was \_\_\_\_\_"

"You have chosen to restart! The word was \_\_\_\_\_"

"Restarting game"

"Invalid word\nUse clear or delete to edit your guess "

"Invalid command"

"Can't add more letters\nUse clear or delete to edit your guess."

For this task call **show\_popup()** with the message as the **content** argument and mode set to **"text"**

As the completeness of your work will be graded with the aid of automated code, you must take care to reproduce the messages exactly as shown above (inserting the current secret word as appropriate) otherwise the automated grading method may not validate the task as completed.

**5 marks**

**Raising Exceptions:**

Ensure that the code raises the following exceptions at the appropriate time.

`"No valid words in wordle words file"`

`"No valid words in all words file"`

`"The secret word is not of the correct length"`

`"Max guesses must be 1 or more"`

`"Word length must be 1 letter or more"`

You do not need to raise other exceptions for cases where Python already raises exceptions e.g. `FileNotFoundError`.

## Part C. Unit Tests

### 15 marks

Write suitable tests to check the functions you have written in **wordle\_backend.py** work correctly.

You need to analyse and design the tests to make sure they are sufficient to check that the processing of the game state is in line with the official online wordle game, and meet the requirements as defined in this document.

Each test should be contained in its own file with a suitable name, e.g. your files might include:

**test\_initialise\_game.py**

**test\_process\_command\_enter\_letter.py**

**test\_process\_command\_delete\_letter.py**

**test\_process\_command\_clear\_guess.py**

**test\_process\_command\_quit.py**

**test\_process\_command\_enter\_guess.py**

**test\_validate\_guess.py**

**test\_process\_guess.py**

**test\_win\_game.py**

**test\_lose\_game.py**

**test\_no\_valid\_wordle\_words\_exception.py**

**test\_no\_valid\_all\_words\_exception.py**

**test\_secret\_word\_incorrect\_length.py**

**test\_max\_guesses\_exception.py**

**test\_word\_length\_exception.py**

If an issue is found your test should raise an **AssertionException** with an appropriate message.

## Optional Work I: Extra functionality

(up to 20 marks maximum to be awarded for work from this section)

### 5 marks

Ensure that the **active\_letters** dictionary is stored in the **game\_state** and is kept up to date as the game is played.


### 5 marks:

Ensure all the **wordle\_back\_end.py** functions work when:

- i) a different number of max guesses is specified when initialising the game.
- ii) a different word length is specified when initialising the game.

### 5 marks:

Add the following command possibility:

command	action
"share" or "@"	if the game is in the "win" or "lose" state, you should encode the game result as a multi-line string that uses unicode characters:  i.e. green, yellow and white blocks to draw the game grid. It should then pass this string as the <b>content</b> argument to the <code>show_popup()</code> function, setting the <b>mode</b> to "share".

### 5 marks

Allow the user to enter multiple commands, e.g. "s t a r e+" should enter all five letters of the guess and process the guess.

### 5 marks

Add a functionality to keep track of the game history, storing the number of games lost, and won by number of guesses made, keeping track in a data file stored in the current working directory.

You may implement this as you wish but your `process_command()` should display the game history either when the game is won or lost or when the entered command is set to "history". This should be done by calling `show_popup()` with **mode** set to "history" and **content** set to contain a list of integers with structure:

index 0	number of games lost
index 1	number of games won in 1 guess
index 2	number of games won in 2 guesses
index 3	number of games won in 3 guesses etc

### 5 marks

Add a functionality so that the game state persists such that in the case that the code is interrupted when the grid is partially completed; the game will continue from that place when the program is restarted.

## Optional Work II: Optimisation and performance

(up to 10 marks maximum to be awarded for work from this section)

### Up to 5 marks

Optimise the efficiency of your algorithms for speed. Submissions will be tested against an example model answer and these marks will be awarded for code that runs performs significantly better than a default implementation.

### Up to 5 marks

Allow a **“hint”** command with shortcut **“\*”**. When a user requests a hint your code should fill the current entry in the game grid with a prefilled guess (but should not yet enter and process the guess).

This hint must only make use of the existing information available to the player. Should the user request an alternative hint a different word should be presented.

You must try to code a hint selection method such that it will enable the secret word to be found in fewest guesses.

Please reference any sources for any method that you do not develop yourself.

Submissions will be tested against an example model answer using a randomised set of example wordle games, and the marker will have the discretion to allocate marks based on how the hints are chosen in terms of speed, quality and originality.



## Appendix:

Example content of game\_state dictionary after initialisation for standard game

key (str)	value
secret_word	e.g. 'inert'
word_length	5
max_guesses	6
grid	[ [' ', ' ', ' ', ' ', ' '], [ ' ', ' ', ' ', ' ', ' '], [ ' ', ' ', ' ', ' ', ' '], [ ' ', ' ', ' ', ' ', ' '], [ ' ', ' ', ' ', ' ', ' '], [ ' ', ' ', ' ', ' ', ' ']]
colour_grid	[ ['w', 'w', 'w', 'w', 'w'], [ 'w', 'w', 'w', 'w', 'w'], [ 'w', 'w', 'w', 'w', 'w'], [ 'w', 'w', 'w', 'w', 'w'], [ 'w', 'w', 'w', 'w', 'w'], [ 'w', 'w', 'w', 'w', 'w']]
i	0
j	0
status	'playing'
wordle_words_file	'wordle_words.txt'
all_words_file	'english_dict.txt'
popup_queue	[]