**Candidate number:**

# Summary

| **Mark /50** | **Total %** |
|---|---|
| | |

## Mark breakdown: Countdown

| | |
|---|---|
| Random character selector | **/10** |
| Dictionary reader | **/10** |
| Word lookup algorithm | **/10** |
| Program design | **/20** |
| **Total** | **/50** |

# Marking criteria: Countdown

Each of the first three section has a potential mark of 10, with 5 marks for function and 5 marks for style. Functionality marks are assigned cumulatively so if the first criteria must pass in order to get marks for the second criteria and so on. Style is assessed based on the style criteria and will be supported by the output of python lint evaluation of the code.

- Random character selector [10 marks]

  5 marks for functionality. Functionality marks accumilate and will be checked by running the code:

  - Does the function `select_characters` execute without errors - 1 mark
  - Does the function ask for an input from the user - 2 marks
  - Does the function return a string length 9 - 3 marks
  - Does the returned string contain the correct number of consonants and vowels - 4 marks
  - Are the consonants and vowels randomly selected - 5 marks

  Up to 5 marks for style. Style marks are assigned individually and checked by python lint and manually checking the source code against the criteria listed below.

  - 1 mark for layout including consistent indentation, line length, etc.
  - 1 mark for effective identifier naming, apopriate variable scoping and appropriate loop nesting.
  - 1 mark for type hinting and using default values where apropriate.
  - 2 marks for doc strings and use of commenting.

- Dictionary reader [10 marks]

  5 marks for functionality. Functionality marks accumilate and will be checked by running the code:

  - Does the function `dictionary_reader` execute without errors - 1 mark
  - Does the function return a list of words - 2 marks
  - Does the function take a string argument - 3 marks
  - Does the list of words filter out punctuation characters - 4 marks
  - Does the list contain all relevant words in the file (words of length over 9 can be removed) - 5 marks

  Up to 5 marks for style. Style marks are assigned individually and checked by python lint and manually checking the source code against the criteria listed below.

  - 1 mark for layout including consistent indentation, line length, etc.
  - 1 mark for effective identifier naming, apopriate variable scoping and appropriate loop nesting.
  - 1 mark for type hinting and using default values where apropriate.
  - 2 marks for doc strings and use of commenting.

- Word lookup algorithm [10 marks]

  5 marks for functionality. Functionality marks accumilate and will be checked by running the code:

  - Does the function `word_lookup` execute without errors - 1 mark
  - Does the function compare the string argument with a list of words - 2 marks
  - Does the function resample the characters of string argument to check for shorter words - 3 marks
  - Does the function return at least one correct answer most of the time - 4 marks
  - Does the function return a complete list of the correct words every time - 5 marks

  Up to 5 marks for style. Style marks are assigned individually and checked by python lint and manually checking the source code against the criteria listed below.

  - 1 mark for layout including consistent indentation, line length, etc.
  - 1 mark for effective identifier naming, apopriate variable scoping and appropriate loop nesting.
  - 1 mark for type hinting and using default values where apropriate.
  - 2 marks for doc strings and use of commenting.

- Program design [20 marks]

  Program design marks will be awarded for the features included in your overall solution. 10 marks will be awarded for effective functionality within the specification. 10 marks will be awarded for bonus features.

  The marks for effective functionality within the specification are assigned for:

  - Does the program `countdown.py` execute without error - 2 marks
  - Does the program ask for suitable consonant and vowel inputs - 4 marks
  - Does the program evaluate the users guess at the end and report the points scored - 6 marks
  - Does the program use apropriate scope for module variables - 8 marks
  - Does the program make good use of __main__ and have the potential to function as an independent module - 10 marks

  Bonus features - these are features that are not in the scope of the module learning materials but are possible with additional research. 2 marks are available for each feature:

  - **Accurate probability distribution:** The Countdown game show doesn't actually have an equal number of each letter on their piles of consonants and vowels. Make your random function reflect the probability of the non-standard probability distribution that is used in the real game (google can be used to find the letter representation).
  - **User input validation:** Users can be unpredictable. If the user inputs something unexpected you should ask for another input until they enter one of the options correctly.

- **ASCII art user interface:** Command-line interfaces focus on functionality by using basic formatting which often isn't the most visually appealing. However, using ascii art can be a way to make them slightly more fun at the beginning and the end of a program without interfering with the functionality.
- **A countdown:** The countdown game show on TV is centered around a physical 'countdown' which lasts 30 seconds. This is usually achieved using threads which are beyond the scope of this course but it is feasible that this could be implemented with some online research.
- **A testing framework:** Any good software has a full testing phase before code is deployed and executed. You don't need to implement a full testing suite but provide a basic test case for each function and return the outcomes of the tests each time the program is run.