Assignment 2 — Hangman

The classic word game, one function at a time

Fakhir Shaheen *(sec - A,B)*

DUE Tue 20 Oct 2026, 11:59 PM **WEIGHT** 3.375% **COVERS** Lectures 1–12 **FILES** 2 **CHECKER** `check.py`

In this assignment, you will implement a variation of the classic word game, Hangman. The computer picks a word, and the player tries to guess its letters.

Objectives

- Writing and calling functions in Python
- Using loop mechanisms to repeat a computational process until a condition is reached
- Using sequential data structures to store and manipulate data such as strings, tuples and lists

Collaboration

- **Students are NOT permitted to look at or copy each other's code or "code structure" / logic. Seeking "help" from seniors or friends is also prohibited. If caught, students will receive a 0 in A2.**
- **AI tools (ChatGPT, Claude, Gemini, Copilot, or any other) are allowed ONLY for explanations, NOTHING ELSE. Not for writing code, fixing code or debugging or writing solutions. Do not paste this sheet or your file into any AI tool. If you use one in any way, it is plagiarism and you get 0 in A2.**

Although this handout is long, the information is here to provide you with context, useful examples, and hints, so be sure to read carefully.

How this assignment works

- Work top to bottom. Read a task, answer its keys, then write its code. Tasks open one at a time: the next opens when this one passes.
- ✎ means pen on this sheet, Shell closed. Only calculators are allowed. ⌨ means at the machine, in `a2.py`.
- The deliverable is `a2.py` in your `a2` folder. Tasks 1 to 8 must PASS, and each is worth the same.
- `check.py` checks your functions the way you can yourself. After `a2.py` has run, it types calls into the Shell, such as `hangman("hi", False)`, and for a game it types the guesses too. Run it as often as you like, and hand in from its menu.
- This assignment assumes Labs 1 to 6 and uses Lectures 1 to 12.

Before you start: get the files

Download the A2 kit from the course calendar and unzip it: `a2.py`, `words.txt` and `check.py`. **Save them all in the same folder**, `a2`, and do not rename them.
Run `a2.py` before writing any code to ensure your files are saved correctly. The starter code

loads words from the `words.txt` file. You should see the following output in your Shell:

```
Loading word list from file...
  55900 words loaded.
```

Hangman Game Overview

You will implement a function called **hangman** that will allow the user to play hangman against the computer. The computer picks the word, and the player tries to guess letters in the word.

The general behavior we want to implement is described below. We will break this down into steps and provide further functional specifications later on in the description.

1. The computer selects a word at random from the list of words in `words.txt`. Note that `words.txt` contains words in all lowercase letters.
2. The user is given a certain number of guesses at the beginning.
3. The user inputs their guess, and the computer either:
   - reveals the letter if it exists in the secret word,
   - informs the user if their guess is invalid *(i.e. longer than 1 character, not a letter, or has already been guessed)* and does not penalize or reveal anything, or
   - penalizes the user and updates the number of guesses remaining if the guess is valid and does not exist in the secret word.
4. The game ends when either the user guesses the secret word or the user runs out of guesses.

We will also introduce a feature to the game to make the game easier for the user. The user will be able to input a special "help" character, `!`, that reveals an unguessed letter at the expense of losing more guesses.

1 The Three Helper Functions (~55 min)

To approach the game, we will break down the problem into logical subtasks, creating three helper functions. This is a common approach to computational problem solving.

The starter code in `a2.py` has implementations for the helper functions `load_words` and `choose_word`. You may read the docstrings associated with each function to understand what they do.

Important: Do NOT change the name, input parameters, or specifications of any of the provided functions! You may add helper functions, but changing the definitions of the given functions will cause `check.py`'s calls to fail. Leave the helper code as it is, too.

Testing. `check.py` calls each function by itself, from the Shell, with words and lists of its own. So a helper function must **return** its answer and print nothing, and it must not change the list it is given. If your function is correct, its row in the report says PASS:

```
Task 01  Determine if the Player has Won PASS
```

TASK 1 Determine if the Player has Won

EASY a2.py ~15 min

⌨ This function will be useful in determining when the hangman game has been won *(i.e., the user has guessed all the letters in the secret word).*

Your task: Implement the function `has_player_won` according to its docstring. It returns `True` if all the letters of `secret_word` are in `letters_guessed`, and `False` otherwise.

Example usage, in the Shell after you run `a2.py`:

```
>>> secret_word = "apple"
>>> letters_guessed = ["e", "i", "k", "p", "r", "s"]
>>> print(has_player_won(secret_word, letters_guessed))
False
```

Why: `a` and `l` are in the word and not in the list, so the player has not won yet.

Testing: run `check.py`. It types six calls of this kind: words with repeated letters, an empty word, an empty list.

Pen on this sheet, Shell closed, before you write Task 1 in `a2.py`. Only calculators are allowed.

Key 1.1 Predict what `has_player_won("banana", ["b", "n", "a"])` should give.
(a) `False`: banana has six letters, and only three were guessed.
(b) `True`: every letter of the word is in the list.
(c) `False`: the list is not in the order of the word.

Key 1.2 The word is `planet` and the list is `["t", "e", "n", "a", "l"]`. The function gives `False`. Work out the one letter that must join the list to make it give `True`.

Answer: .....

Key 1.3 The secret word is the empty string, `""`, and the list is `["x"]`. What should the function give?
(a) `False`: nothing in the list was a right guess.
(b) An error: an empty word cannot be checked.
(c) `True`: no letter of the word is missing from the list.

TASK 2 Revealing Letters

EASY a2.py ~20 min

⌨ The game shows the secret word with the letters guessed so far in their places, and an asterisk (`*`) for every letter not guessed yet.

Your task: Next, implement the function `get_word_progress` according to its docstring. This should be fairly similar to `has_player_won`.

Hint: Think about ...

- if you need to store information as you loop over a data structure
- how you want to add information to your accumulated result

Example usage:

```
>>> secret_word = "apple"
>>> letters_guessed = ["e", "i", "k", "p", "r", "s"]
>>> print(get_word_progress(secret_word, letters_guessed))
*pp*e
```

Why: `p` and `e` were guessed, so they show. `a` and `l` were not, so each is a star.

Testing: run `check.py`. It types six calls of this kind.

Pen on this sheet, Shell closed, before you write Task 2 in `a2.py`. Only calculators are allowed.

Key 2.1 Predict what `get_word_progress("letter", ["t", "e"])` should give.

Answer: .....

Key 2.2 Predict how many `*` there are in the string that `get_word_progress("mississippi", ["s", "p"])` gives.

Answer: .....

Key 2.3 The word is `moon` and the list is `["x", "o", "z"]`. Which string should the function give?

(a) `*oo*`
(b) `*o**`
(c) `oo`

TASK 3 Get Available Letters

EASY `a2.py` ~20 min

Before each guess, the game shows the letters that have not been guessed yet.

Your task: Next, implement the function `get_available_letters` according to its docstring. This function should return the letters in alphabetical order.

Hint: You might consider using `string.ascii_lowercase`, which is a string comprised of all lowercase letters:

```
>>> import string
>>> print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz
```

Example usage:

```
>>> letters_guessed = ["e", "i", "k", "p", "r", "s"]
>>> print(get_available_letters(letters_guessed))
abcdfghjlmnoqtuvwxyz
```

Why: Six letters were guessed, so 20 of the 26 are left, still in alphabetical order.

Testing: run `check.py`. It types five calls of this kind.

Pen on this sheet, Shell closed, before you write Task 3 in `a2.py`. Only calculators are allowed.

Key 3.1 Predict the first five letters of the string that `get_available_letters(["c", "a", "t"])` gives.

Answer: .....

Key 3.2 Predict how many letters there are in the string that `get_available_letters(["z", "a", "z"])` gives.

Answer: .....

Key 3.3 The list is `["q", "b"]`, in that order. The function gives a string back. How does that string begin?
(a) `acde`: alphabetical order, without `b` and `q`.
(b) `cdef`: everything up to the first guessed letter goes too.
(c) `qb`: it gives back the letters of the list.

2 The Game (~250 min)

Now that you have written some useful helper functions, you can implement the **hangman** function, which takes two parameters: (1) `secret_word`, the secret word the user is to guess; and (2) `with_help`, a boolean representing whether or not the game is to be played with the "**help**" functionality.

Calling the hangman function starts up an interactive game between the user and the computer. In designing your code, be sure you take advantage of the three helper functions, `has_player_won`, `get_word_progress`, and `get_available_letters` from Section 1! **You may write additional helper functions if you need them.**

Important notes:

1. **Make your print statements exactly as in the example games.** `check.py` compares every line your game prints, letter for letter. The box below lists every line, and the appendix contains helpful examples of gameplay.
2. Please refer to the **Helpful Debugging Tips** section near the end if you have trouble debugging.

Every line the game prints

When	The game prints	Task
at the start	`Welcome to Hangman!`	4
	`I am thinking of a word that is N letters long.`	4
before each guess	`-------------` (14 dashes)	4
	`You have G guesses left.`	4
	`You have 1 guess left.` (when G is 1)	5
	`Available letters: L`	4
asking for a guess	`Please guess a letter:` (the prompt of `input`; one space after the colon)	4
a letter in the word	`Good guess: P`	4
a letter not in the word	`Oops! That letter is not in my word: P`	5
not one letter	`Oops! That is not a valid letter. Please input a letter from the alphabet: P` (one line)	7
a letter guessed before	`Oops! You've already guessed that letter: P`	7
`!`, with help	`Letter revealed: X` and then, on its own line, `P`	8
`!`, with fewer than 3 guesses	`Oops! Not enough guesses left: P`	8
the word is complete	`-------------`	4
	`Congratulations, you won!`	4
	`Your total score for this game is: S`	4
no guesses are left	`-------------`	6
	`Sorry, you ran out of guesses. The word was W.`	6

`N` is the length of the secret word, `G` the guesses remaining, `L` what `get_available_letters` returns, `P` what `get_word_progress` returns, `X` the revealed letter, `S` the score and `W` the secret word.

Testing your game

By hand. To test your hangman function, you should call it from within the `if __name__ == "__main__":` condition found at the bottom of `a2.py`. Manually set the secret word to make it easier to test your code:

```
if __name__ == "__main__":
    secret_word = "tact"
    with_help = False
    hangman(secret_word, with_help)
```

However, in the end, you should also test your program with **the secret word selected at random**. If you scroll to the bottom of `a2.py`, you will see the lines below. Uncomment the bottom three lines to choose a random secret word and play hangman with it. Feel free to pass in your own secret word when testing your program.

```
if __name__ == "__main__":
    # To test your game, uncomment the following three lines.
    # secret_word = choose_word(wordlist)
    # with_help = False
    # hangman(secret_word, with_help)
```

You can also leave those lines as they are and start a game from the Shell, after you run `a2.py`: type `hangman("tact", False)`.

With `check.py`. In order to test if your game runs properly, please run `check.py`. It starts games the same way, from the Shell, and types the guesses itself. If your function is correct, you should see PASS beside Tasks 4 to 8. If it is not, the report shows the first line of a game that differs. **Before you run `check.py`, put the `#` back in front of your test lines:** a game that starts by itself takes the guesses `check.py` types, and the report then says `EOFError`.

Game Setup

1. The `secret_word` along with the boolean `with_help` are passed into the `hangman` function as parameters.
2. At the start of the game, display how many letters `secret_word` contains.
3. Users start with 10 guesses.

User-Computer Interaction

1. Before each guess, you should display to the user:
   - A row of 14 dashes (`--------------`) to separate individual guesses from each other. **Leaving out the row of dashes will cause `check.py` to fail.**
   - How many guesses they have remaining
   - All the letters that have not yet been guessed
2. Ask the user to supply one guess at a time.
   - The user can type any number, symbol, or letter. Your code should only accept capital and lowercase single letters as valid guesses! (Task 7)
   - If the game is played with help, your code should also accept the help character (`!`). (Task 8)
3. Immediately after each guess, you should display:
   - Whether or not the letter is in the secret word
   - The word with guessed letters revealed and unguessed letters as asterisks (`*`)

Game Termination: a Win

1. The game ends when the user guesses all the letters in `secret_word` or has `0` guesses remaining (Task 6).
2. If the user wins, print a congratulatory message, and tell the user their score.
   - `total_score` = (`guesses_remaining` + 4 × number of unique letters in `secret_word`) + (3 × length of `secret_word`)
   - Example: for a game with secret word `asleep` with 6 guesses remaining, there are a total of 5 unique letters (`a, s, l, e,` and `p`). Thus, the final score is $(6 + 4 \times 5) + (3 \times 6) = 44$.

Example (win): the end of a game with the word `dolphin`, after five wrong consonants.

```
# ... snip ...
You have 5 guesses left.
Available letters: abcegnrstuvwxyz
Please guess a letter: n
Good guess: dolphin
--------------
Congratulations, you won!
Your total score for this game is: 54
```

NOTE: `# ... snip ...` is not part of the output; it indicates that only part of the example game is shown. The blue letter is what the user types.

TASK 4 **The Game: Setup to a Win** HARD `a2.py` ~90 min

⌨ The first version of the game: the user guesses only letters that are in the word, so no guess is lost, and the game ends with a win.

Your task: Write `hangman` so that this user can play the game and win it: the game setup, the turns, and the winning end with its score.

Input: one guess on each line, a small letter that is in the word. Read it with `input("Please guess a letter: ")`.

Output: the lines marked Task 4 in the box above: two at the start, three before each guess, `Good guess: P` after it, and three when the word is complete.

Sample 1 (the checker plays this game; the blue text is what it types)

```
>>> hangman("hi", False)
Welcome to Hangman!
I am thinking of a word that is 2 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: h
Good guess: h*
--------------
You have 10 guesses left.
Available letters: abcdefgijklmnopqrstuvwxyz
Please guess a letter: i
Good guess: hi
--------------
Congratulations, you won!
Your total score for this game is: 24
```

Why: Both letters are in the word, so no guess is lost. The score is $(10 + 4 \times 2) + (3 \times 2) = 24$.

Hint: a `while` loop that goes on until `has_player_won` says `True`. Keep the guessed letters in a list, and add each guess to it. `check.py` also plays `hangman("tact", False)` with `t`, `a`, `c`.

Pen on this sheet, Shell closed, before you write Task 4 in `a2.py`. Only calculators are allowed.

Key 4.1 The call is `hangman("noon", False)`. The user inputs `n`, then `o`. Predict the number the game should print after `Your total score for this game is:`.

Answer: .....

Key 4.2 In that game, predict how many rows of dashes the game should print.

Answer: .....

Key 4.3 The call is `hangman("tact", False)`, and the user inputs `c` first. Which line should the game print for it?

(a) `Good guess: c`
(b) `Good guess: **c*`
(c) `Good guess: tact`

Key 4.4 The call is `hangman("tact", False)`. The user inputs `t`, then `a`. Predict how many letters the next `Available letters:` line should show.

Answer: .....

Key 4.5 A user wins a game with the word `pepper` and still has all 10 guesses. Work out the score the game should print.

Answer: .....

Guesses Remaining

If the user inputs:

1. Anything besides a letter in the alphabet *(e.g. symbols or numbers)*, tell the user that they can only input an alphabet letter. The user loses **no** guesses. Note: when the game is being played with help, `!` is also a valid input. (Task 7)
2. A letter that has already been guessed, print a message telling the user the letter has already been guessed before. The user loses **no** guesses. (Task 7)
3. Any letter that hasn't been guessed before and the letter is in the secret word, the user loses **no** guesses.
4. **Consonants:** If the user inputs a consonant that hasn't been guessed and the consonant is not in the secret word, the user loses one guess.
5. **Vowels:** If the user inputs a vowel that hasn't been guessed and the vowel is not in the secret word, the user loses **two** guesses. Vowels are `a, e, i, o` and `u`. The letter `y` does not count as a vowel. Note: if a user inputs an incorrect vowel that hasn't been guessed and there is only one guess remaining, the user loses and the game is over. (Task 6)

TASK 5 Guesses Remaining

MEDIUM `a2.py` ~30 min

Rules 3, 4 and 5 above: a letter in the word is free, a wrong consonant costs one guess, and a wrong vowel costs two.

Your task: Make a letter that is not in the word cost guesses, and say so. When exactly one guess remains, the line before the guess says `guess`, not `guesses`.

Input: as in Task 4, but a guess may be a letter that is not in the word.

Output: the lines of Task 4, as before. After a letter that is not in the word: `Oops! That letter is not in my word: P`. With one guess remaining: `You have 1 guess left.`

Sample 1 (the checker plays this game)

```
>>> hangman("hi", False)
Welcome to Hangman!
I am thinking of a word that is 2 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: h
Good guess: h*
--------------
You have 10 guesses left.
Available letters: abcdefgijklmnopqrstuvwxyz
Please guess a letter: e
Oops! That letter is not in my word: h*
--------------
You have 8 guesses left.
Available letters: abcdfgijklmnopqrstuvwxyz
Please guess a letter: i
Good guess: hi
--------------
Congratulations, you won!
Your total score for this game is: 22
```

Why: `e` is a vowel and is not in `hi`: two guesses go. The score is $(8 + 4 \times 2) + (3 \times 2) = 22$.

Hint: `check.py` also plays `hangman("tact", False)` with `a`, `b`, `t`, `e`, `c`.

Pen on this sheet, Shell closed, before you write Task 5 in `a2.py`. Only calculators are allowed.

Key 5.1 The call is `hangman("tact", False)`. The user inputs `b`, then `e`. Predict the number the next turn should print after `You have`.

Answer: .....

Key 5.2 The word is `tact` and the user inputs `y`. How many guesses should the user lose?
(a) 2: `y` sounds like a vowel.
(b) 0: only vowels cost guesses.
(c) 1: `y` is not one of `a, e, i, o, u`.

Key 5.3 The call is `hangman("else", False)`. The user inputs `e`, then `o`, then `s`. Predict the number the next turn should print after `You have`.

Answer: .....

Key 5.4 The call is `hangman("hi", False)`. The user inputs `o`, `u`, `h`, `i`. Predict the number the game should print after `Your total score for this game is:`.

Answer: .....

Game Termination: a Loss

If the player runs out of guesses before completing the word, tell them they lost and reveal the word to the user when the game ends.

Example (lose): the end of a game with the word `else`. One guess remained, and a wrong vowel costs two.

```
# ... snip ...
You have 1 guess left.
Available letters: hijklmnopqrstvwxyz
Please guess a letter: i
Oops! That letter is not in my word: e**e
--------------
```

```
Sorry, you ran out of guesses. The word was else.
```

TASK 6 Running Out of Guesses

MEDIUM a2.py ~30 min

⌨ The game ends when the user has `0` guesses remaining. A wrong vowel with one guess remaining ends it too.

Your task: End the game when no guesses remain, with the losing line.

Input: as in Task 5.

Output: the lines of Tasks 4 and 5, as before. When no guesses remain, no new turn starts: the game prints the row of dashes, then `Sorry, you ran out of guesses. The word was W.`, and ends.

Sample 1 (the checker plays this game: `a`, `e`, `i`, `o`, `u`)

```
>>> hangman("my", False)
Welcome to Hangman!
I am thinking of a word that is 2 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! That letter is not in my word: **
# ... snip ...
--------------
You have 2 guesses left.
Available letters: bcdfghjklmnpqrstuvwxyz
Please guess a letter: u
Oops! That letter is not in my word: **
--------------
Sorry, you ran out of guesses. The word was my.
```

Why: Five vowels, none of them in `my`, cost two guesses each: 10, 8, 6, 4, 2, 0.

Hint: the loop of Task 4 now has two reasons to stop. `check.py` also plays `hangman("why", False)` with `u`, `o`, `i`, `e`, `a`.

Pen on this sheet, Shell closed, before you write Task 6 in `a2.py`. Only calculators are allowed.

Key 6.1 The call is `hangman("rhythm", False)`, and every letter the user inputs is a consonant that is not in the word. Predict how many guesses the user makes before the game ends.

Answer: .....

Key 6.2 A user has 1 guess remaining and inputs a vowel that is not in the word. What should the game do?

(a) It ends, with the line `Sorry, you ran out of guesses.` and the word.
(b) It prints `You have -1 guesses left.` and asks again.
(c) It takes one guess only, prints `You have 0 guesses left.` and asks again.

Key 6.3 The call is `hangman("my", False)`. The user inputs `e`, `a`, `o`, and after them only consonants that are not in the word. Predict how many of those consonants the user inputs before the game ends.

Answer: .....

Key 6.4 The word is `else`, the user has guessed `e`, and the guesses run out. Which line is the last one the game prints?

(a) `You have 0 guesses left.`
(b) `Sorry, you ran out of guesses. The word was else.`
(c) `Sorry, you ran out of guesses. The word was e**e.`

Invalid and Repeated Guesses: Hints

1. Use calls to the `input()` function to get the user's guess.
   - Check that the user input is an alphabet letter *(or the help character if the game is played with help)*.
2. Since the words in `words.txt` are lowercase, we suggest converting user input to lowercase so the program only needs to handle lowercase characters.
3. You may find the string functions `str.isalpha()` and `str.lower()` helpful! You can type `help(str.isalpha)` or `help(str.lower)` in the Thonny Shell to see documentation for the functions.

Example usage of `str.isalpha()` and `str.lower()`:

```
>>> my_string = "HeLLoWoRlD"
>>> my_string.isalpha()
True
>>> my_string.lower()
'helloworld'
>>> help(str.isalpha)
...
```

TASK 7 **Invalid and Repeated Guesses** MEDIUM a2.py ~40 min

Rules 1 and 2 of *Guesses Remaining*: a guess that is not one letter of the alphabet, or a letter guessed before, costs no guess. The game only tells the user. Capital and lowercase single letters are both valid guesses.

Your task: Check every guess before you use it. Take a capital letter as its lowercase letter.

Input: one guess on each line: anything the user can type.

Output: the lines of Tasks 4 to 6, as before. For a guess that is not one letter: `Oops! That is not a valid letter. Please input a letter from the alphabet: P`. For a letter guessed before: `Oops! You've already guessed that letter: P`.

Sample 1 (the checker plays this game: `foo`, `2`, `H`, `h`, `i`)

```
>>> hangman("hi", False)
Welcome to Hangman!
I am thinking of a word that is 2 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: foo
Oops! That is not a valid letter. Please input a letter from the alphabet: **
# ... snip ...
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: H
Good guess: h*
--------------
You have 10 guesses left.
Available letters: abcdefgijklmnopqrstuvwxyz
Please guess a letter: h
Oops! You've already guessed that letter: h*
# ... snip ...
```

Why: `foo` is not one letter. `H` counts as `h`, so the `h` after it was guessed before. No guess is lost.

Hint: `check.py` also plays `hangman("tact", False)` with `+`, `t`, `T`, `a`, `c`.

Pen on this sheet, Shell closed, before you write Task 7 in `a2.py`. Only calculators are allowed.

Key 7.1 The word is `tact`, and the user inputs a capital `T` first. Which line should the game print for it?
(a) The `not a valid letter` line: only small letters count.
(b) `Oops! That letter is not in my word: ****`
(c) `Good guess: t**t`

Key 7.2 The call is `hangman("tact", False)`. The user inputs `s`, then `s` again, then `S`. Predict the number the next turn should print after `You have`.

Answer: .....

Key 7.3 In a new game the user inputs `a`, `A`, `3`, `ab`, `b`. Predict how many letters the next `Available letters:` line should show.

Answer: .....

Key 7.4 The user presses Enter on an empty line. Which line should the game print?
(a) The `not a valid letter` line.
(b) Nothing: it asks again.
(c) The `already guessed` line.

The Game with Help

It isn't always easy to beat the computer, especially when it selects an esoteric word. It might be nice if you could ask for some help. To do this you will create a feature of the game that works as follows:

- If you type the special character `!`, the computer will provide you with one of the missing letters in the secret word at a cost of **three** guesses. This character should be the **only** non-letter input that your game accepts as a guess.
- **If you do not have at least three guesses remaining**, the computer will warn you of this and let you try again. You lose **no** guesses.

Note: The user can play the game with this feature only when the `with_help` parameter is **True**. When it is `False`, `!` is not a valid letter.

As a starting point, we suggest writing **a helper function that chooses a letter to reveal**. It should take two arguments: the secret word and the string of available letters *(from `get_available_letters`)*. This helper function should create a string `choose_from` containing the unique letters that are in both the secret word and the available letters. You can then use the following statements to pick a random character `revealed_letter` from that string:

```
new = random.randint(0, len(choose_from)-1)
revealed_letter = choose_from[new]
```

Your helper function should then return this `revealed_letter`. Back in your original game logic, you'll need to add a conditional statement to catch the case of the user inputting `!`. This case, if triggered, can add the letter returned by your helper function to `letters_guessed`, show the new word progress, decrement the remaining guesses by 3, and continue the gameplay.

See the appendix for two examples: a complete game with help (D), and help that is refused (E).

TASK 8 The Game with Help

HARD a2.py ~60 min

⌨ With `with_help` set to `True`, the user may type `!` for help. Help reveals one missing letter and costs three guesses, if the user has at least three.

Your task: Add help to `hangman`.

Input: as in Task 7; `!` asks for help.

Output: the lines of Tasks 4 to 7, as before. For `!` with 3 guesses or more, two lines: `Letter revealed: X`, then `P` alone. With fewer than 3: `Oops! Not enough guesses left: P`.

Sample 1 (the checker plays this game: `t`, `a`, `!`)

```
>>> hangman("tact", True)
# ... snip ...
--------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrsuvwxyz
Please guess a letter: !
Letter revealed: c
tact
--------------
Congratulations, you won!
Your total score for this game is: 31
```

Why: Only `c` is missing, so help reveals it. Help costs three guesses: $(7 + 4 \times 3) + (3 \times 4) = 31$.

Hint: a revealed letter can complete the word. `check.py` also plays `hangman("hi", True)` with `h`, `!`, and `hangman("hi", False)` with `!`, `h`, `i`.

Pen on this sheet, Shell closed, before you write Task 8 in `a2.py`. Only calculators are allowed.

Key 8.1 The call is `hangman("level", True)`. The user inputs `l`, then `e`, then `!`. Predict the letter the game should print after `Letter revealed:`.

Answer: .....

Key 8.2 In a new game with help, the user inputs `!` as the first guess. Predict the number the next turn should print after `You have`.

Answer: .....

Key 8.3 A user has exactly 3 guesses remaining and inputs `!`. What should the game do?
(a) Print `Oops! Not enough guesses left` and take no guesses.
(b) Reveal a letter. The user then has 0 guesses remaining.
(c) Reveal a letter and take no guesses, because so few are left.

Key 8.4 `with_help` is `False`, and the user inputs `!`. Which line should the game print?
(a) `Letter revealed:` and a letter.
(b) `Oops! Not enough guesses left:` and the word so far.
(c) The `not a valid letter` line of Task 7.

Key 8.5 The word is `racecar`, and so far the user has guessed only `r`. The user inputs `!`. Work out how many different letters the game could reveal.

Answer: .....

Marking and Hand-in

Marks. Tasks 1 to 8 are worth the same: 12.5 each, 100 in all. `check.py`'s points are a first mark only: it contains only a subset of the tests that will be run to determine the grade, so passing all of its tasks does not guarantee full credit on Assignment 2. After the deadline, each task is run again on five or six hidden cases: other words, other lists of letters, other guesses, longer games. A task earns full marks when all of them pass, and half when only `check.py`'s own calls pass.

Naming files. Save your solutions with the original file name: `a2.py`. **Do not ignore this step or save your file with a different name! `check.py` will not be able to find your file if you do and you will receive no marks.**

Final submission. **Be sure to run `check.py` and make sure all the tasks pass.** Then hand in with `check.py` too, from its menu. You can hand in as often as you like: the best whole hand-in counts. **Due Tue 20 Oct 2026, 11:59 PM.** Late: minus 10% for each day, up to three days. Nothing after that.

Before you hand in

- ☐ `check.py` says PASS for Tasks 1 to 8.
- ☐ The three test lines at the bottom of `a2.py` have their `#` again.
- ☐ The file is still named `a2.py`, and `words.txt` is beside it.
- ☐ The helper code is as the starter gave it, and so are the names and parameters of the four functions.
- ☐ Every line your game prints is in the box of Section 2. No other `print` is left in the file.
- ☐ You played whole games yourself: one you won, one you lost, one with help, and one with a random word.
- ☐ You typed every line yourself, and can explain every line.

Helpful Debugging Tips

- Remember, `check.py` is testing your printed output against the expected printed output. That makes it necessary that you design your game to match exactly. For instance, **your output needs to have the same number of lines as the expected output**. Therefore, be careful to include everything that our examples print out, including the dashes (`----------`) at the end of a turn!
- If you fail any of the game tasks, the report of `check.py` shows the call, such as `hangman("hi", False)`, and the first line of that game that differs: the expected line and your actual line.

  - **Do not be concerned that the game's answer comes right after the prompt in that report**, as in `Please guess a letter: Good guess: h*`. What `check.py` types does not show, so the prompt and the next line the game prints are one line there.
  - Make sure you provide the exact prompt to `input`. That is, you should be calling `input("Please guess a letter: ")`, with one space after the colon. The prompt is printed too, so `check.py` compares it.

- If the report says `EOFError`, a game asked for one more guess than `check.py` had. Either your test lines at the bottom of `a2.py` are switched on, or your game did not end when it should.

- If you are using the print function, you may be adding extra spaces unintentionally. For example, the following statement inserts a space between the two strings.

  ```
  >>> print('foo', 'bar')
  foo bar
  ```

  You can change what print inserts between strings by passing in an optional `sep` parameter, which is by default one space. Python by default inserts a `\n`, the newline character, at the end of the string to be printed, which is why you normally see each print statement on a new line. You can change this using the optional `end` parameter. You don't need this for the assignment, but it's good to be aware of it.

  ```
  >>> print('foo', 'bar', 'baz', sep=" and ", end='\n')
  foo and bar and baz
  ```

  Alternatively you can use f-strings to format your output. For example:

  ```
  >>> foo = 'foo'
  >>> bar = 'bar'
  >>> baz = 'baz'
  >>> print(f'{foo}{bar} and {baz}')
  foobar and baz
  ```

- Type every line of the game yourself. A line copied from a PDF can hold a character that only looks right, such as a curly apostrophe in `You've`, and `check.py` then finds a difference you cannot see.

- The Thonny console might insert an extra blank line in the output even though there's nothing in the code that tells it to. This should not affect `check.py`.

- Are you scoring things properly? You might want to write a function to help you calculate the score given a word and the number of guesses remaining. Be sure to account for guesses lost using hints as well.

As always, strategically **printing the values of variables** is often a helpful way of checking that the correct values are stored in those variables. Take those prints out again before you run `check.py`.

Appendix: Example Games

Five games, played by the finished `a2.py`. This is how a game looks in the Shell: the blue text is what the user types, and it is not printed by the game. The rest is what the game prints, and your `a2.py` must print the same lines for the same guesses. `# ... snip ...` is not part of the output: it shows that only part of a game is printed here. One line is too long for a column, and continues after ↪: on the screen it is **one** line. In D and E the revealed letter is picked at random, so your game may reveal another letter of `racecar`.

A. Every Kind of Guess

```
Welcome to Hangman!
I am thinking of a word that is 4 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: *a**
--------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: 2
Oops! That is not a valid letter. Please input a
    ↪ letter from the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: foo
Oops! That is not a valid letter. Please input a
    ↪ letter from the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: +
Oops! That is not a valid letter. Please input a
    ↪ letter from the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: t
Good guess: ta*t
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrsuvwxyz
Please guess a letter: e
Oops! That letter is not in my word: ta*t
--------------
You have 7 guesses left.
Available letters: cdfghijklmnopqrsuvwxyz
Please guess a letter: e
Oops! You've already guessed that letter: ta*t
# ... snip ...
```

B. Hangman Example (Winning Game)

```
Loading word list from file...
  55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 4 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: *a**
--------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! You've already guessed that letter: *a**
--------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: s
Oops! That letter is not in my word: *a**
--------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: +
Oops! That is not a valid letter. Please input a
    ↪ letter from the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: t
Good guess: ta*t
--------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqruvwxyz
Please guess a letter: e
Oops! That letter is not in my word: ta*t
--------------
You have 7 guesses left.
Available letters: bcdfghijklmnopqruvwxyz
Please guess a letter: c
Good guess: tact
--------------
Congratulations, you won!
Your total score for this game is: 31
```

C. Hangman Example (Losing Game)

```
Loading word list from file...
  55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 4 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! That letter is not in my word: ****
--------------
You have 8 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: ****
--------------
You have 7 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: c
Oops! That letter is not in my word: ****
--------------
You have 6 guesses left.
Available letters: defghijklmnopqrstuvwxyz
Please guess a letter: 2
Oops! That is not a valid letter. Please input a
    ↪ letter from the alphabet: ****
--------------
You have 6 guesses left.
Available letters: defghijklmnopqrstuvwxyz
Please guess a letter: d
Oops! That letter is not in my word: ****
--------------
You have 5 guesses left.
Available letters: efghijklmnopqrstuvwxyz
Please guess a letter: u
Oops! That letter is not in my word: ****
--------------
You have 3 guesses left.
Available letters: efghijklmnopqrstvwxyz
Please guess a letter: e
Good guess: e**e
--------------
You have 3 guesses left.
Available letters: fghijklmnopqrstvwxyz
Please guess a letter: f
Oops! That letter is not in my word: e**e
--------------
You have 2 guesses left.
Available letters: ghijklmnopqrstvwxyz
Please guess a letter: o
Oops! That letter is not in my word: e**e
--------------
Sorry, you ran out of guesses. The word was else.
```

D. Hangman With Help

```
Loading word list from file...
  55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 7 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: r
Good guess: r*****r
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqstuvwxyz
Please guess a letter: !
Letter revealed: c
r*c*c*r
--------------
You have 7 guesses left.
Available letters: abdefghijklmnopqstuvwxyz
Please guess a letter: !
Letter revealed: a
rac*car
--------------
You have 4 guesses left.
Available letters: bdefghijklmnopqstuvwxyz
Please guess a letter: e
Good guess: racecar
--------------
Congratulations, you won!
Your total score for this game is: 41
```

E. With Help: Not Enough Guesses

```
Welcome to Hangman!
I am thinking of a word that is 7 letters long.
--------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: !
Letter revealed: r
r*****r
--------------
You have 7 guesses left.
Available letters: abcdefghijklmnopqstuvwxyz
Please guess a letter: !
Letter revealed: a
ra***ar
--------------
You have 4 guesses left.
Available letters: bcdefghijklmnopqstuvwxyz
Please guess a letter: !
Letter revealed: e
ra*e*ar
--------------
You have 1 guess left.
Available letters: bcdfghijklmnopqstuvwxyz
Please guess a letter: !
Oops! Not enough guesses left: ra*e*ar
# ... snip ...
```