Hangman

Python

Goal

Write the core logic for a small Hangman game. The assignment is broken into functions so each part can be tested independently. You may also write a playable `main()` function, but the required grade is based on the functions listed below.

Starter State

Your file must define these global variables exactly:

```
WORD_BANK = {
    1: "python",
    2: "debug",
    3: "loop",
    4: "variable",
    5: "function",
}

secret_word = ""
guessed_letters = []
wrong_guesses = 0
max_wrong = 6

STATUS_SUCCESS = 0
ERROR_NOT_A_NUMBER = 1
ERROR_WORD_NOT_FOUND = 2
```

Required Functions

1. `initialize_game(choice, allowed_wrong=6)`

   Convert `choice` to a number if possible. If the number is in `WORD_BANK`, use the mapped word as `secret_word`, clear `guessed_letters`, reset `wrong_guesses` to 0, set `max_wrong`, and return `STATUS_SUCCESS`.

   If `choice` is not a number or cannot be converted to a number, return `ERROR_NOT_A_NUMBER` and do not change the current game state. If `choice` is a number but is not mapped to a word in `WORD_BANK`, return `ERROR_WORD_NOT_FOUND` and do not change the current game state.

   This function should always return an integer status code.

2. `is_valid_guess(guess)`

Return `True` only when `guess` is exactly one alphabetic character and has not already been guessed. Uppercase letters should be treated the same as lowercase letters.

3. `display_word()`

   Return the secret word with unguessed letters replaced by underscores. Separate each character with one space. For example, if the secret word is `"hangman"` and the guessed letters are `["a", "n"]`, return:

   ```
   _ a n _ _ a n
   ```

4. `process_guess(guess)`

   Handle one player guess and return one of these exact strings:

   - `"invalid"` for empty, multi-character, non-letter, or non-string guesses.
   - `"already guessed"` when the letter was already guessed.
   - `"correct"` when the letter is in the secret word.
   - `"incorrect"` when the letter is not in the secret word.

   Valid new guesses should be stored in `guessed_letters`. Incorrect valid guesses should increase `wrong_guesses` by 1. Invalid and repeated guesses should not change `guessed_letters` or `wrong_guesses`.

5. `is_game_over()`

   Return `True` if the player has guessed every letter or if `wrong_guesses` is greater than or equal to `max_wrong`. Otherwise return `False`.

6. `get_result_message()`

   Return `"You win!"` if the word is complete, `"You lose!"` if the player has used all wrong guesses, and `"Keep guessing!"` otherwise.

Debugging

You may use the grader's debug helper:

```
from codegrader.student import debug

debug("current guesses:", guessed_letters)
```

When you run your file normally, debug messages print to the error stream. During grading, they are captured separately from normal printed output.