Table of Contents
- Background
  - Wordlist file
  - Template
  - `GameState` objects
- Problem statement
- Instructions
  - `HumanPlayer` class
    - `turn()` method
  - `ComputerPlayer` class
    - `__init__()` method
    - `turn()` method
  - Other instructions
    - Length of individual lines of code
    - Docstrings
- Running your code
- Submitting your code
- Grading
- Academic integrity

Exercise: Hangman

Background

Hangman is a popular game that involves guessing a word one letter at a time. If you are unfamiliar with this game, you may want to watch this video for additional background information: https://www.youtube.com/watch?v=cGOeiOjfYPk.
We will complete a multi-player game based on hangman. Words will be chosen from a wordlist file. Each player will be allowed a certain number of bad guesses; once they reach this limit, they lose and will not be given any more turns. On a player's turn, they may guess a single letter or try to solve the puzzle by typing in an entire word or phrase. The game is not case-sensitive.
Your goal is to develop a human player and a computer player as subclasses of the abstract `Player` base class.

Wordlist file

The file `nounlist.txt` can be used as a wordlist for the game. It comes from http://www.desiquintans.com/nounlist.

Template

A template script is provided. The template contains the following classes and functions:
- `GameState` : instances of this class are passed to each player's `turn()` method to communicate the current state of the game to the player.
- `Game` : this class sets up and manages a game of hangman.
- `Player` : this is the abstract class that you will need to subclass.
- `main()` : this function reads wordlists, instantiates Player objects and a Game object, and starts a game.
- `parse_args()` : this function processes command-line arguments.

`GameState` objects

An instance of the `GameState` class is passed to each player's `turn()` method to provide important information on the current state of the game. When printed, `GameState` objects provide
- a string representation of the current state of the board, showing which letters have been guessed and which have not
- a list of characters that have been guessed that are not found in the word
- the number of bad guesses made by each character

`GameState` objects have a `match()` method that takes a string and returns `True` if the string is compatible with the current letters and blanks on the board (for example, the word "temporary" is compatible with the board `T E • • • R A R •`; the word "temerity" is not). The `match()` method does not take into account whether the string contains "bad guesses" (letters known not to be in the word).
`GameState` objects contain a number of additional potentially interesting attributes; see the class docstring for details on the purpose and structure of each of these.

Problem statement

Write two subclasses of the Player class in the provided template. One should be called `HumanPlayer` ; the other should be called `ComputerPlayer` .
For your `HumanPlayer` class, you will override the `turn()` method as described below.
For your `ComputerPlayer` class, you will override both the `__init__()` and `turn()` methods as described below.

Instructions

Please use the template to implement your solution to this homework. Make sure your program is called `hangman.py` . Follow the instructions below.

`HumanPlayer` class

Create a class called `HumanPlayer` that is a subclass of the `Player` class. Inherit the `__init__()` method from the parent class.

`turn()` method

Your `HumanPlayer` class should have a `turn()` method with two parameters:
- `self`
- a `GameState` object representing the current state of the game

This method should print the board (you can do this by printing the `GameState` object). It should prompt the user **by name** to guess a letter or solve the entire puzzle. For example:

```
Mark, guess a letter or type a word to solve the puzzle:
```

It should return the value typed by the user.

`ComputerPlayer` class

Create a class called `ComputerPlayer` that is a subclass of the `Player` class.

`__init__()` method

Your `ComputerPlayer` class should override the parent class's `__init__()` method with a method that has three parameters
- `self`
- a name for the computer player (a string)
- a vocabulary list (a list of strings); these are the words your computer player can draw from when taking turns

Store the name in an attribute called `self.name` . Store the vocabulary list in an attribute called `self.vocab` .

`turn()` method

Your `ComputerPlayer` class should have a `turn()` method with two parameters:
- `self`
- a `GameState` object representing the current state of the game

This method should either guess a letter that has not been guessed yet or guess a word to solve the puzzle. It should return the letter or word.
Note: the `choice()` function from the `random` module has been imported for you. When you pass it a list, it randomly selects one item from the list. For example, `choice(["a", "b", "c"])` will randomly select `"a"` , `"b"` , or `"c"` .
For full points, your `turn()` method should beat a ComputerPlayer who just picks an unguessed letter at random on each turn.

Other instructions

Length of individual lines of code

Please keep your lines of code to 80 characters or less. If you need help breaking up long lines of code, please see https://umd.instructure.com/courses/1299872/pages/how-to-break-up-long-lines-of-code.

Docstrings

Please write docstrings for your classes and for your `turn()` methods. Docstrings were covered in the first week's lecture videos (https://youtu.be/jHTv83PlQYw?t=1415) and revisited in the OOP lecture videos (https://youtu.be/Oq9ssywHMPg).
There's an ELMS page about them here: https://umd.instructure.com/courses/1299872/pages/docstrings.
Docstrings are not comments; they are statements. Python recognizes a string as a docstring if it is the first statement in the body of the method, function, class, or script/module it documents. Because docstrings are statements, the quotation mark at the start of the docstring must align exactly with the start of other statements in the method, function, class, or module.

▶ General instructions for class docstrings

▶ General instructions method and function docstrings

Running your code

To run your program within the VS Code built-in terminal, first make sure you have opened (in VS Code) the directory where your program is saved. If necessary, you can go to the VS Code File menu and select "Open…" on macOS or "Open Folder…" on Windows, and navigate to the directory where your program is.
Then, open the VS Code built-in terminal. Type `python3` (on macOS) or `python` (on Windows) followed by a space and the name of your program. Specify values for the name of the wordlist file and the names of all human players, separated by spaces. You may optionally add a computer player by typing `-c` , and a separate vocabulary file for your computer player by typing `-v` and the path to another wordlist file. Below are some examples of ways to invoke your program:

```
python3 nounlist.txt Sarah Owen
python3 nounlist.txt Sarah -c
python3 nounlist.txt Sarah -c -v some_other_wordlist.txt
python3 nounlist.txt -c
```

Submitting your code

Upload your `hangman.py` script to Gradescope. An autograder script will give you near-instant feedback. If you did not pass all the test cases, you can revise your code and resubmit as many times as you want until the deadline.

Grading

This assigment is worth 10 points in the exercise category, allocated as follows:
5.5 points are allocated to automatic tests of your code functionality and docstrings. 4.5 points are awarded based on the degree of completeness of your program and docstrings.

Category	Points	Notes
Automatic tests	5.5	Tests will evaluate instance attributes, return values, side effects, and docstrings of your class and its methods
Manual evaluation of code	4.5	Points deducted for incorrect, incomplete, or missing elements, including but not limited to docstrings, methods, and classes

Additionally, your participation in the exercise session is graded out of 4 points. You are expected to come to class prepared and be actively engaged in the problem-solving process throughout the exercise session.

Academic integrity

This assignment is to be done by you individually or with a partner or partners as assigned in class. Outside help of any kind (including, but not limited to, help from the internet, tutors, or classmates other than your assigned partner) is not allowed. Disseminating these instructions in whole or in part without written permission of the instructor is considered an infraction of academic integrity. Posting the instructions, or any part thereof, on the internet is considered dissemination and is strictly prohibited.

Table of Contents

- Background
  - Wordlist file
  - Template
  - `GameState` objects
- Problem statement
- Instructions
  - `HumanPlayer` class
    - `turn()` method
  - `ComputerPlayer` class
    - `__init__()` method
    - `turn()` method
  - Other instructions
    - Length of individual lines of code
    - Docstrings
- Running your code
- Submitting your code
- Grading
- Academic integrity