Back to fopp

This project will take you through the process of implementing a simplified version of the game Wheel of

course_4_project

Fortune. Here are the rules of our game:

Due: 2019-02-04 15:17:00

Score: 0 of 3 = 0.0%

```
Description: Final Project for Course 4 - Wheel of Python
```

Questions

```
    There are num_human human players and num_computer computer players.

    Every player has some amount of money ($0 at the start of the game)

    Every player has a set of prizes (none at the start of the game)

      • The goal is to guess a phrase within a category. For example:
            Category: Artist & Song

    Phrase: Whitney Houston's I Will Always Love You

    Players see the category and an obscured version of the phrase where every alphanumeric

         character in the phrase starts out as hidden (using underscores: _):
            Category: Artist & Song

    Note that case (capitalization) does not matter

      • During their turn, every player spins the wheel to determine a prize amount and:
            • If the wheel lands on a cash square, players may do one of three actions:
                   Guess any letter that hasn't been guessed by typing a letter (a-z)
                         Vowels (a, e, i, o, u) cost $250 to guess and can't be guessed if the player doesn't
                           have enough money. All other letters are "free" to guess
                         The player can guess any letter that hasn't been guessed and gets that cash
                           amount for every time that letter appears in the phrase
                         If there is a prize, the user also gets that prize (in addition to any prizes they
                           already had)
                         If the letter does appear in the phrase, the user keeps their turn. Otherwise, it's
                           the next player's turn
                         Example: The user lands on $500 and guesses 'W'
                               ■ There are three W's in the phrase, so the player wins $1500
                   Guess the complete phrase by typing a phrase (anything over one character that
                     isn't 'pass')
                         If they are correct, they win the game
                         If they are incorrect, it is the next player's turn
                   Pass their turn by entering 'pass'

    If the wheel lands on "lose a turn", the player loses their turn and the game moves on to the

               next player
            • If the wheel lands on "bankrupt", the player loses their turn and loses their money but they
               keep all of the prizes they have won so far.
      • The game continues until the entire phrase is revealed (or one player guesses the complete phrase)
  First, let's learn about a few functions and methods that we'll use along the way to do this project. There are
  no questions to answer in the next four active code windows. They are just here to introduce you to some
  functions and methods that you may not be aware of. The active code window that starts with "Part A" is
  where you are first asked to complete code.
  The time.sleep(s) function (from the time module) delays execution of the next line of code for s
  seconds. You'll find that we can build a little suspense during gameplay with some well-placed delays. The
   game can also be easier for users to understand if not everything happens instantly.
                                                       Original - 1 of 1
                               Save & Run
 1 import time
 3 for x in range(2, 6):
        print('Sleep {} seconds..'.format(x))
        time.sleep(x) # "Sleep" for x seconds
 6 print('Done!')
Sleep 2 seconds..
Sleep 3 seconds..
Sleep 4 seconds..
Sleep 5 seconds..
Done!
                                         ActiveCode (wof_ac_sleep)
  The random module includes several useful methods for generating and using random numbers, including:

    random.randint(min, max) generates a random number between min and max (inclusive)
```

random.choice(L) selects a random item from the list L

Save & Run

4 print('Random number between 1 and 10: {}'.format(rand_number))

6 letters = [letter for letter in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ']

1 import random

3 rand_number = random.randint(1, 10)

7 rand_letter = random.choice(letters)

Random number between 1 and 10: 8

Random letter: F

8 print('Random letter: {}'.format(rand_letter))

Original - 1 of 1

ActiveCode (wof_ac_rand)

There are also several string methods that we haven't gone over in detail but will use for this project:

• .upper() converts a string to uppercase (the opposite is .lower())

• .count(s) counts how many times the string s occurs inside of a larger string

Not yet graded

Not yet graded

Not yet graded

```
Original - 1 of 1
                              Save & Run
 1 myString = 'Hello, World! 123'
 3 print(myString.upper()) # HELLO, WORLD! 123
 4 print(myString.lower()) # hello, world! 123
 5 print(myString.count('1')) # 3
 7 s = 'python is pythonic'
 8 print(s.count('python')) # 2
HELLO, WORLD! 123
hello, world! 123
2
                                        ActiveCode (wof_ac_str)
  We're going to define a few useful methods for you:
      • getNumberBetween(prompt, min, max)) repeatedly asks the user for a number between min and
         max with the prompt prompt
      • spinWheel() simulates spinning the wheel and returns a dictionary with a random prize

    getRandomCategoryAndPhrase() returns a tuple with a random category and phrase for players to

        guess

    obscurePhrase(phrase, guessed) returns a tuple with a random category and phrase for players to

  Take some time to read their implementations below.
```

Original - 1 of 2

Save & Run

userinp = input(prompt) # ask the first time

7 # Repeatedly asks the user for a number between min & max (inclusive)

5 LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

8 def getNumberBetween(prompt, min, max):

1 import json

3 import time

10

2 import random

Not yet graded

```
while True:
11
12
            try:
13
                 n = int(userinp) # try casting to an integer
                 if n < min:</pre>
14
getRandomCategoryAndPhrase()
-> ('Song/Show', 'UNDER THE SEA FROM THE LITTLE MERMAID')
obscurePhrase('UNDER THE SEA FROM THE LITTLE MERMAID', ['K', 'Q', 'G', 'J', 'V', 'O', 'M', 'E', 'X',
-> ___ER _HE _E_ _ROM _HE ____E MERM___
showBoard('UNDER THE SEA FROM THE LITTLE MERMAID', '___ER _HE _E_ _ROM _HE ____E MERM___', ['K','Q'
Category: UNDER THE SEA FROM THE LITTLE MERMAID
Phrase: ___ER _HE _E_ _ROM _HE ____E MERM_
Guessed: E, G, H, J, K, M, O, Q, R, V, X, Y
Spinning the wheel 4 times (normally this would just be done once per turn)
spinWheel()
{'type': 'cash', 'text': '$950', 'value': 950, 'prize': 'A trip to Ann Arbor!'}
spinWheel()
{'type': 'cash', 'text': '$650', 'value': 650, 'prize': 'A brand new car!'}
spinWheel()
{'type': 'bankrupt', 'text': 'Bankrupt', 'prize': False}
spinWheel()
{'type': 'cash', 'text': '$650', 'value': 650, 'prize': 'A brand new car!'}
In 2 seconds, will run getNumberBetween('Testing getNumberBetween(). Enter a number between 1 and 10
                                  ActiveCode (wof_ac_other_methods)
  Part A: WOFPlayer
  We're going to start by defining a class to represent a Wheel of Fortune player, called WOFPlayer. Every
  instance of WOFPlayer has three instance variables:
      • .name: The name of the player (should be passed into the constructor)

    .prizeMoney: The amount of prize money for this player (an integer, initialized to 0)

    .prizes: The prizes this player has won so far (a list, initialized to [])

  Of these instance variables, only name should be passed into the constructor.
  It should also have the following methods (note: we will exclude self in our descriptions):

    .addMoney(amt): Add amt to self.prizeMoney

      .goBankrupt(): Set self.prizeMoney to 0
      .addPrize(prize): Append prize to self.prizes
      .__str__(): Returns the player's name and prize money in the following format:
            Steve ($1800) (for a player with instance variables .name == 'Steve' and prizeMoney ==
              1800)
  Part B: WOFHumanPlayer
  Next, we're going to define a class named WOFHumanPlayer, which should inherit from WOFPlayer (part A).
  This class is going to represent a human player. In addition to having all of the instance variables and
  methods that WOFPlayer has, WOFHumanPlayer should have an additional method:

    getMove(category, obscuredPhrase, guessed): Should ask the user to enter a move (using

         input()) and return whatever string they entered.
    .getMove() 's prompt should be:
    {name} has ${prizeMoney}
    Category: {category}
    Phrase: {obscured_phrase}
    Guessed: {guessed}
    Guess a letter, phrase, or type 'exit' or 'pass':
   For example:
```

Not yet graded

```
Steve has $200
    Category: Places
    Phrase: _L___ER N___N_L P_RK
    Guessed: B, E, K, L, N, P, R, X, Z
    Guess a letter, phrase, or type 'exit' or 'pass':
  The user can then enter:

    'exit' to exit the game

    'pass' to skip their turn

    a single character to guess that letter

    a complete phrase (a multi-character phrase other than 'exit' or 'pass') to guess that phrase

  Note that .getMove() does not need to enforce anything about the user's input; that will be done via the
  game logic that we define in the next ActiveCode window.
  Part C: WOFComputerPlayer
   Finally, we're going to define a class named WOFComputerPlayer, which should inherit from WOFPlayer
   (part A). This class is going to represent a computer player.
  Every computer player will have a difficulty instance variable. Players with a higher difficulty
  generally play "better". There are many ways to implement this. We'll do the following:
      • If there aren't any possible letters to choose (for example: if the last character is a vowel but this
         player doesn't have enough to guess a vowel), we'll 'pass'

    Otherwise, semi-randomly decide whether to make a "good" move or a "bad" move on a

         given turn (a higher difficulty should make it more likely for the player to make a "good"
         move)

    To make a "bad" move, we'll randomly decide on a possible letter.

    To make a "good" move, we'll choose a letter according to their overall frequency in the

               English language.
  In addition to having all of the instance variables and methods that WOFPlayer has, WOFComputerPlayer
  should have:
   Class variable

    SORTED_FREQUENCIES: Should be set to 'ZQXJKVBPYGFWMUCLDRHSNIOATE', which is a list of English

         characters sorted from least frequent ( 'Z' ) to most frequent ( 'E' ). We'll use this when trying to
         make a "good" move.
  Additional Instance variable

    .difficulty: The level of difficulty for this computer (should be passed as the second argument into

         the constructor after .name )
  Methods
      • .smartCoinFlip(): This method will help us decide semi-randomly whether to make a "good" or
         "bad" move. A higher difficulty should make us more likely to make a "good" move. Implement this by
         choosing a random number between 1 and 10 using random.randint(1, 10) (see above) and
        returning True if that random number is greater than self.difficulty. If the random number is
         less than or equal to self.difficulty, return False.

    .getPossibleLetters(guessed): This method should return a list of letters that can be

         guessed.

    These should be characters that are in LETTERS ( 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' ) but not in

               the guessed parameter.

    Additionally, if this player doesn't have enough prize money to guess a vowel (variable

               VOWEL_COST set to 250), then vowels (variable VOWELS set to 'AEIOU') should not be
               included

    .getMove(category, obscuredPhrase, guessed): Should return a valid move.

    Use the .getPossibleLetters(guessed) method described above.

    If there aren't any letters that can be guessed (this can happen if the only letters left to guess

               are vowels and the player doesn't have enough for vowels), return 'pass'
            • Use the .smartCoinFlip() method to decide whether to make a "good" or a "bad" move
                   • If making a "good" move ( .smartCoinFlip() returns True ), then return the most
                     frequent (highest index in .SORTED_FREQUENCIES ) possible character
                   • If making a "bad" move ( .smartCoinFlip() returns False ), then return a random
                     character from the set of possible characters (use random.choice())
                               Save & Run
                                                       Original - 1 of 4
 1 VOWEL_COST = 250
 2 LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
 3 VOWELS = 'AEIOU'
 5 # Write the WOFPlayer class definition (part A) here
 7 # Write the WOFHumanPlayer class definition (part B) here
 9 # Write the WOFComputerPlayer class definition (part C) here
10
11
                                      ActiveCode (wof_ac_wof_player)
   Result | Actual Value | Expected Value
                                                                   Notes
                                             Error: NameError: name 'WOFPlayer' is not defined
   ERROR
               None
                             None
                                         Error: NameError: name 'WOFHumanPlayer' is not defined
   ERROR
               None
                             None
                                        Error: NameError: name 'WOFComputerPlayer' is not defined
               None
   ERROR
                             None
   You passed: 0.0% of the tests
   Putting it together: Wheel of Python
  Below is the game logic for the rest of the "Wheel of Python" game. We have implemented most of the
  game logic. Start by carefully reading this code and double checking that it all makes sense. Then,
   paste your code from the previous code window in the correct places below.
   Note 1: we added the following code to ensure that the Python interpreter gives our game time to run:
    import sys
    sys.setExecutionLimit(600000)
   sys.setExecutionLimit(ms) says that we should be able to run our program for ms milliseconds before it
  gets stopped automatically.
  Note 2: As you play, you will need to keep scrolling down to follow the game.
```

Not yet graded

```
Save & Run
                                               Load History
1 # PASTE YOUR WOFPlayer CLASS (from part A) HERE
2 # PASTE YOUR WOFHumanPlayer CLASS (from part B) HERE
3 # PASTE YOUR WOFComputerPlayer CLASS (from part C) HERE
 5
6 import sys
7 sys.setExecutionLimit(600000) # let this take up to 10 minutes
8
9 import json
10 import random
11 import time
12
13 LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
14 VOWELS = 'AEIOU'
15 €
                                   ActiveCode (wof ac final)
                                          Score Me
                                                                             username: hakimihamzan10@gmail.com | Back to top
```