Project 3: Phrase Hunters

How to Approach This Project

There are a number of ways to approach and complete this project. Below, one approach is detailed in steps, but you do not have to follow this approach to the letter. You are free to explore your own solution, as long as it meets the requirements laid out in the "How you will be graded" rubric.

Part I

Step 1

Download the project files. In the project files folder, you'll see a folder named 'phrase-hunter' that includes the following files:

- app.py which will contain the main logic for your project
- another "phrasehunter" folder that contains game.py and phrase.py
- __init__.py which allows you to import from a subdirectory
- game.py will contain the Game class to manage the functioning of the game: it has methods for showing the game, handling interactions and checking for when the game is over
- phrase.py will contain the Phrase class to manage individual phrases

Step 2

Declare your classes.

- Inside game.py, declare the Game class.
- Inside phrase.py, declare the Phrase class

  Note: put `pass` in the body for now. We'll fill them out later.

Instantiate your classes.

- Inside app.py, import your two classes
- Add a dunder main inside your app.py
- Create an instance of Phrase and assign it to a variable.

- Create an instance of Game and assign it to a variable.

```
phrase = Phrase()
game = Game()
```

Test your code!

Now that you've defined your Phrase and Game classes and instantiated objects, you can add some temporary `print` statements to app.py to make sure the instances are being created.

```
print(phrase)
print(game)
```

Running app.py should give you output that looks somewhat like the following:

```
$ python app.py
<phrasehunter.phrase.Phrase object at 0x000001B630174250>
<phrasehunter.game.Game object at 0x000001B630158970>
```

The numbers at the end will be different on your system and refer to memory addresses. This proves that the objects are being created! Once you've verified that your code is working properly you can comment out (or remove) the testing code that you just created in app.py.

Step 3

Create the attributes and `__init__` methods for your classes:

- The Game class `__init__` method has only `self` as a parameter. The Game class has the following attributes:
    - `missed`: Used to track the number of missed guesses by the player. The initial value is 0 since no guesses have been made at the start of the game.
    - `phrases`: A list of Phrase objects to use with the game. For now, initialize the attribute to an empty list. In the next step you'll work on initializing this property with a list of Phrase objects.
    - `active_phrase`: This is the Phrase object that's currently in play. The initial value is `None`
    - `guesses`: A list that contains all guesses made by the user during the course of the game. Start by setting it to a list that only contains a list with a string for a space. `self.guesses = [" "]`

- The Phrase class __init__ method should receive two parameters: `self` and `phrase`. The Phrase class has the following attribute:
    - `phrase`: This is the actual phrase as a string that the Phrase object is representing. This attribute should be set to the `phrase` parameter but converted to all lower case.

Test your code!

Now that you've added an `__init__` method to your Phrase class, you can add temporary code to your app.py to make sure that everything is working as expected. Open up app.py and add the following code:

```
phrase = Phrase('Life is like a box of chocolates')

print(phrase.phrase)
```

Run app.py. You should see "life is like a box of chocolates" in all lower case letters. Once you've verified that your code is working properly you can comment out (or remove) the testing code that you just created in app.py.

Step 4

Time to create your phrases that the game will randomly choose from when showing a new phrase to the player! You will need to import your Phrase class into the Game class. There are multiple ways to create your phrases and add them to the Game class's `phrases` property:

- Option #1: Inside the Game class, create a method called `create_phrases()` that creates and returns a list of five new `Phrase` objects, and then set the `phrases` property to call that method.
- Option #2: Simply add 5 new Phrase objects directly in the empty array that was originally set as the value of the `phrases` property.

When creating a Phrase object, don't forget to pass in the actual string phrase that the Phrase object is representing. A string phrase should only include letters and spaces -- no numbers, punctuation or other special characters. A phrase must contain more than one word.

Test your code!

Now that you've added code to initialize and populate the Game class `phrases` property, let's add more temporary code to the app.py file to ensure that everything is working properly. Open up the app.py file and add the following code (note that you don't need to import Phrase in app.py anymore):

```
game = Game()

for phrase in game.phrases:
    print(phrase.phrase)
```

Run app.py. Do you see something similar to the following output (keeping in mind that your phrases are probably different)?

```
hello world
there is no trying
may the force be with you
you have to see the matrix for yourself
life is like a box of chocolates
```

Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.py file.

Step 5

Let's write the get_random_phrase() method mentioned in the last step. This method goes inside the Game class in the game.py file. This method should select and then return a random phrase from the array of phrases stored in the Game class's `phrase` property. You will need to `import random`.

Test your code!

Now that you've added the code for the get_random_phrase() method, let's add more temporary code to the app.py file to ensure that everything is working properly. Open up the app.py file and add the following code:

```
def print_phrase(phrase_object)
    print(f"The phrase is: {phrase_object.phrase}")

game = Game()
print_phrase(game.get_random_phrase())
print_phrase(game.get_random_phrase())
print_phrase(game.get_random_phrase())
print_phrase(game.get_random_phrase())
print_phrase(game.get_random_phrase())
```

Run app.py. Do you see something similar to the following output (keeping in mind again that your phrases are probably different)? You might see a phrase repeat. Running the app.py again will likely produce different results. Notice that instead of seeing a complete list of the phrases list, we're seeing a random phrase printed.

```
The phrase is: may the force be with you
The phrase is: there is no trying
The phrase is: hello world
The phrase is: there is no trying
The phrase is: you have to see the matrix for yourself
```

Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.py file.

Step 6

Go back to the `__init__` of your Game class and change the `self.active_phrase` to be the result of calling the `get_random_phrase` method.

Test your code!

Back in app.py let's create some more temporary code. This time our `active_phrase` attribute should be a random phrase each time we run app.py.

```
game = Game()
print(game.active_phrase)
print(game.active_phrase.phrase)
```

Each time you run app.py the active_phrase attribute should be set to a random phrase. The first print statement should print out that it is a Phrase object and a memory address and the second print statement should print out the `phrase` attribute of the `Phrase` object. Keeping in mind that your phrases are likely different, the output should look something like:

```
$ python app.py
<phrasehunter.phrase.Phrase object at 0x00000284F5CF13A0>
there is no trying
```

Once you've verified that the code is working properly, you can comment out (or remove) the testing code that you just added to the app.py file.

Step 7

Switch gears for a moment and head to the Phrase class inside the phrase.py file. Inside the Phrase class, create a method called `display()`. The `display` method will have two parameters: `self` and `guesses`. It will need to take the `guesses` from the Game class so that it can compare the letters that have been guessed with the letters in the `phrase` attribute.

Loop through the letters in the `phrase` attribute. If the letter was found in `self.phrase`, print the letter and then a space. Otherwise, print an underscore "_" followed by a space. To keep this from printing each letter on new line, set the `end` attribute to be a space like so:

```
print(f"{letter}", end=" ")
```

Test your code!

Back in app.py print the active phrase of the Game object. Then call the display method on the active_phrase and pass in the guesses attribute of the Game object.

```
game = Game()
print(game.active_phrase.phrase)
game.active_phrase.display(game.guesses)
```

The output should clearly indicate the separation of words and it should all be underscores and spaces at this point.

```
$ python app.py
may the force be with you

_ _ _   _ _ _   _ _ _ _ _   _ _   _ _ _ _   _ _ _
```

Once you've verified that the code is working properly, you can comment out (or remove) the testing code that you just added to the app.py file.

Step 8

Let's create a welcome message for our game which will display at the start. Inside game.py create a new method called `welcome()`. Feel free to experiment with your welcome message to make it your own. It should just be a print statement that prints a welcome message.

Test your code!

Back in app.py let's create some more temporary code to test our new welcome method. Make sure that the Game object is created and then call the welcome() method on the object.

```
game = Game()
game.welcome()
```

The output for our welcome method looks like this, but yours may differ:

```
$ python app.py

        ============================
          Welcome to Phrase Hunter
        ============================
```

Step 9

Now that we have a way to display our phrase as underscores and letters and a welcome message, let's head back to game.py and create a new method called `start()`. This method will start by calling the `welcome()` method to display the welcome, printing the number of misses (or incorrect guesses), and displaying the active phrase. Later it will also handle determining when the game should end.

Inside start call the `welcome()` method. Add a print statement to print out "Number missed: " followed by `self.missed`, then call the `display()` method on the `active_phrase` attribute. Don't forget to pass in `self.guesses` to the `display()` method.

Test your code!

Back in app.py create your Game instance. For the purposes of testing, print out the `phrase` attribute of the Phrase instance stored in `active_phrase`. Run the `start()` method on the Game object.

```
game = Game()
print(game.active_phrase.phrase)
game.start()
```

The output should look something like this, though yours may differ somewhat. It should show the welcome message, the number missed (which should be zero), and the phrase displayed as underscores and spaces:

```
$ python app.py
hello world

        ============================
          Welcome to Phrase Hunter
        ============================

Number missed: 0

_ _ _ _ _   _ _ _ _ _
```

Once you've confirmed that your code works, comment out or remove the print statement for the phrase.

Part II

Step 10

Now it's time to start adding some user interaction to the game. Let's start by creating a method to get the user's guess and store it. In game.py, create a new method named `get_guess`. This method should prompt the user for a guess and return the user's input. Inside the `start()` method, create a new variable named `user_guess` and assign it the input that is returned by the `get_guess()` method. Append `user_guess` to the `guesses` attribute.

For testing purposes, it is recommended that you print the `user_guess` after it receives the value returned by `get_guess` and then call the `display` method on the `active_phrase` attribute to make sure it is working properly.

Test your code!

Use the same code to test that you used in Step 9 without the print statement for the phrase. Running the `start()` method on the Game instance should produce something that looks similar to the following:

```
$ python app.py

        ============================
        Welcome to Phrase Hunter
        ============================


Number missed: 0

_ _ _ _ _   _ _ _ _ _
Enter a letter: o
o

_ _ _ _ o   _ o _ _ _
```

The random phrase was “hello world”, and as you can see we guessed “o” which is present twice.

Once you verify that your code is working properly, remove or comment out the second call to `display` and remove the print of `user_guess`.

Step 11

Now it’s time to check if the user got the letter incorrect. Currently, our code doesn’t have a way to increase the number missed if the user guessed incorrectly. Over in phrase.py create a method named `check_guess`. Besides `self`, it will also take a parameter for `guess`. We’re going to send in the `user_guess`. It should return `True`, if the guess was correct and `False` if the guess was incorrect.

Test your code!

Inside your `start` method in game.py after you have appended the `user_guess` to `guesses`, call the `check_guess` method on the `active_phrase` attribute and send in the `user_guess`. The following code should print “YAY” if the guess was correct and “Bummer!” if it was incorrect:

```
if self.active_phrase.check_guess(user_guess):
    print("YAY")
else:
    print("Bummer!")
```

If you guessed incorrectly, your results should look something like the following:

```
$ python app.py

        ============================
          Welcome to Phrase Hunter
        ============================

Number missed: 0

_ _ _   _ _ _   _ _ _ _ _   _ _   _ _ _ _   _ _ _
Enter a letter: z
Bummer!
```

If you guessed correctly, you should get similar results:

```
$ python app.py

        ============================
          Welcome to Phrase Hunter
        ============================

Number missed: 0

_ _ _ _ _   _ _ _ _ _
Enter a letter: e
YAY
```

Step 12

Now it's time to start adding to the `missed` attribute in the case that the user guesses incorrectly and begin the loop that will continue until the game is either won or lost. First, we'll tackle ending the game when the number missed reaches five.

First, let's remove the `else` statement that prints out "Bummer!".

Previously, we had:

```
if self.active_phrase.check_guess(user_guess):
```

We only need to do something here if it is an incorrect guess. Namely, we need to update the `missed` attribute by incrementing it by 1.

Change that to:

```
if not self.active_phrase.check_guess(user_guess):
```

Add the code inside the if statement to update the `missed` attribute.

Now we're going to add a `while` loop to keep this going as long as the number of `missed` is less than 5. Because the welcome should only be shown once, this should be added immediately below the call to the `welcome` method.

Test your code!

At this point, your game should be running almost normally. The game should end when you've guessed five times incorrectly. However, you'll notice that it doesn't end when all letters are guessed. We're going to tackle that in the next step.

This is some sample output. You should see something similar:

```
        ============================
          Welcome to Phrase Hunter
        ============================

Number missed: 0

_ _ _ _   _ _   _ _ _ _   _   _ _ _   _ _   _ _ _ _ _ _ _ _ _ _
Enter a letter: e
Number missed: 0
_ _ _ e   _ _   _ _ _ e   _   _ _ _   _ _   _ _ _ _ _ _ _ _ e _
Enter a letter: a
Number missed: 0
_ _ _ e   _ _   _ _ _ e   a   _ _ _   _ _   _ _ _ _ _ _ a _ e _
Enter a letter: h
Number missed: 0
_ _ _ e   _ _   _ _ _ e   a   _ _ _   _ _   _ h _ _ _ _ a _ e _
Enter a letter: z
Number missed: 1
_ _ _ e   _ _   _ _ _ e   a   _ _ _   _ _   _ h _ _ _ _ a _ e _
Enter a letter: v
Number missed: 2
```

Step 13

Now it's time to handle when the user wins the game! Inside the phrase.py class, create a new method called `check_complete`. The method should return `True` if all letters have been guessed and `False` if any character has not been guessed. Besides `self`, this method will also have a parameter of `guesses` and we'll need to pass in the `guesses` attribute from the Game class. Loop through the `phrase` attribute. If any letter is not present in `guesses`, return `False`. If it makes it through the entire loop without returning `False`, return `True`.

In game.py, change your `while` loop so that it runs while `missed` is less than five and calling `check_complete` returns `False`.

Test your code!

Your game should now end if you either have 5 incorrect guesses or if the phrase was completed and the user won!

Here's some sample output:

```
$ python app.py

        ============================
          Welcome to Phrase Hunter
        ============================

Number missed: 0
_ _ _ _ _   _ _ _ _ _
Enter a letter: h
Number missed: 0
h _ _ _ _   _ _ _ _ _
Enter a letter: z
Number missed: 1
h _ _ _ _   _ _ _ _ _
Enter a letter: o
Number missed: 1
h _ _ _ o   _ o _ _ _
Enter a letter: w
Number missed: 1
h _ _ _ o   w o _ _ _
Enter a letter: r
Number missed: 1
h _ _ _ o   w o r _ _
Enter a letter: n
Number missed: 2
```

```
h _ _ _ o   w o r _ _
Enter a letter: l
Number missed: 2
h _ l l o   w o r l _
Enter a letter: d
Number missed: 2
h _ l l o   w o r l d
Enter a letter: e
```

Step 14

Our app works, but it ends a little unexpectedly right now. Let's add some polish to the end so that we can either congratulate the user or tell them that they lost. Inside game.py create a new method named `game_over`. If the number `missed` is equal to the five print out a message to the user that they lost the game. Otherwise, print out a congratulatory message and tell them that they won.

Place the call to `game_over()` after the `while` loop.

Test your code!

The game should run normally and end when it's supposed to with an appropriate message to the user.

Sample output:

```
Enter a letter: f
Number missed: 1
m a y   t h e   f o _ _ e   _ e   _ i t h   y o _
Enter a letter: r
Number missed: 1
m a y   t h e   f o r _ e   _ e   _ i t h   y o _
Enter a letter: c
Number missed: 1
m a y   t h e   f o r c e   _ e   _ i t h   y o _
Enter a letter: b
```

```
Number missed: 1
m a y   t h e   f o r c e   b e   _ i t h   y o _
Enter a letter: w
Number missed: 1
m a y   t h e   f o r c e   b e   w i t h   y o _
Enter a letter: u
Congratulations! You won!
```

Congratulations! Feel free to play with the formatting, refining your phrase list and generally making this app your own. If you're wanting an additional challenge, consider tackling the "Exceeds Expectations" requirements for this project.