CS5001
Spring 2019
HW3
Assigned: January 31, 2019
Deadline: February 7, 2019 at 9:00am

To submit your solution, compress all files together into one .zip file. Login to `handins.ccs.neu.edu` with your CCIS account, click on the appropriate assignment, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

Your solution will be evaluated according to our CS5001 grading rubric. The written component accounts for 20% of your overall HW3 score; the programming component for 80%. We'll give written feedback when appropriate as well; please come to Laney's office hours with any questions about grading (email to set up a time if no current office hours work for you).

This is an introductory course, and we want to make sure that everyone has a solid understanding of the fundamentals, so we'll often rule some Python tools in or out. For this homework, do not use any built-in list functions other than **`len`**, and do not use any loops besides **`while`** and **`for...range`**.

If you apply your late token, your new deadline is **February 11th at 9:00am**. Email laneys@northeastern.edu to notify us you'll be cashing it in.

Written Component

- Filename: `written.txt`

Written #1

Suppose you're prompting the user, and you want them to enter 'Y' to continue or 'N' to stop. If they enter any other values, you prompt them again, until they get it right. What boolean expression would you use to accomplish this?

Fill in the blanks (???) in Line 2, below:

```
1  user_in = input("Enter Y/N\n")
2  while ??? :
3      user_in = input("Invalid input, enter again\n")
```

Written #2

2A

Flow of control: List the line numbers in the order in which they would be executed

```
1  i = 0
2  while i < 2:
    if i == 1:
        break
    i = i + 1
print('i is', i)
```

2B
What does the Python code in 2A print to the terminal?

2C
In the code below, how many times will Line 5 be executed?

```
1  i = 0
2  while i < 4:
3      if i == 1:
4          i = i + 2
5      i = i + 1
6  print('i is', i)
```

2D
What does the Python code in 2C print to the terminal?

Written #3
After the Python code below is run, what is the value of *sum*?

```
i = 0
sum = 0
while i < 7:
    if i % 2 == 1:
        sum += 1
    i += 1
```

Written #4
For each of the Python snippets below, what will be printed to the terminal?

4A

```
lst = [1, 2, 3]
for i in range(len(lst)):
    print(lst[i])
```

4B

```
lst = [1, 2, 3]
for i in range(len(lst)):
    print(i)
```

4C

```
lst = [1, 2, 3]
for i in range(len(lst)):
    print(len(lst) - i)
```

Programming Component
Make sure you review the Python Style Guide. A percentage of your score for every homework is for writing good, clear, readable code.

Programming #1

- Filename (starter code): `sentence.py`
- Filename (your driver): `typing.py`
- If you create another file to define your functions in, that's fine, just make sure you submit it!

It may not be the most glamorous skill, but we all need to be good at typing. Did you learn how to type in the fourth grade with that little QWERTY dude? I did, and we had to have our typing speed evaluated using the standard measure -- the number of words per minute you can type (correctly).

For this program, you'll write your own version of the words-per-minute typing test. We've gotten a few things started in the `sentence.py` file linked above, and you'll polish off the rest. We start off that file with a big list of sentences to practice typing with. Some of them come from actual typing tests we found online, some from the Khoury College website, and some from my old Facebook posts. Just a random bunch of stuff.

We've provided the following functions (think of them as black boxes; trust that they do what they say).

- **`select_sentence`.** Returns a randomly-chosen sentence, one of the ones described above. (If you call this function multiple times, it might return the same sentence more than once; that's expected and totally fine.)
- **`count_word`.** Returns the number of words in a given sentence. It cares only about spaces to separate words.
- **`count_mismatches`**. Returns the number of word-mismatches between two sentences. Punctuation and case matter. If one sentence is "Hello World!" and the other is "Hello World?", that is a mismatch count of one, because *Hello* is typed correctly but *World!* is not .

You'll use these functions to put together a program that does the following:

- Gives the user sentences to type, until they type DONE and then the test is over.
- Counts the number of seconds from when the user begins to when the test is over.
- Counts and reports:
  - The total number of words the user typed, and how many seconds it took them.
  - The words-per-minute that turns out to be.
  - The number of mistakes they made.
  - The adjusted words-per-minute, accounting for mistakes.

WPM and adjusted WPM... Suppose the user types 25 words and it takes them 30 seconds. That would be 50 words per minute. But if they made two mistakes, then we say they typed 23 words in 30 seconds, and so adjusted WPM is 46.

The number of mistakes is always subtracted from the total words typed, regardless of whether it was a misspelled word, an extra word, or a missing word.

Requirements:

- The number of words typed and number of mistakes made should should be integers throughout the program. The words per minute and adjusted words per minute, when reported on the terminal, should be rounded to the nearest whole number.
- The number of seconds should be rounded to the nearest 100th.
- The test must keep going, selecting sentence after sentence after sentence until the user types DONE.
- The user might type DONE at the very beginning and just get zeroes for everything.

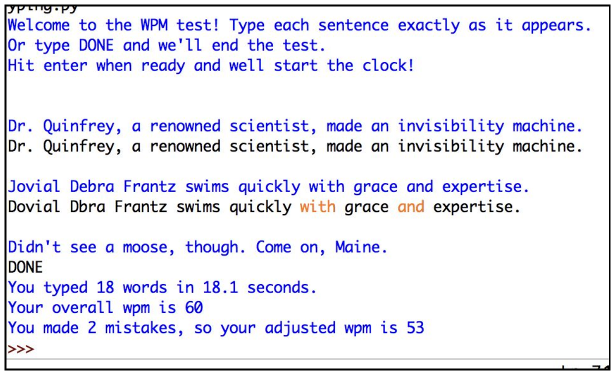
Helpful hints:

- Python has a time module that you'll probably find useful.
- You should prompt the user to just type Enter at the beginning, and start the timer right after that.
- You should stop the timer right after they type DONE (don't worry about any extra little bits of time it took them to type DONE).

AMAZING points:

- Submit a modified `sentence.py`... don't change any functions, but add your own sentences for testing.
- Include in the block comment at the very top your own results from the typing test (mine are in the example below)

Here's an example of running the program (as always, your wording can vary!):



```
Welcome to the WPM test! Type each sentence exactly as it appears.
Or type DONE and we'll end the test.
Hit enter when ready and well start the clock!


Dr. Quinfrey, a renowned scientist, made an invisibility machine.
Dr. Quinfrey, a renowned scientist, made an invisibility machine.

Jovial Debra Frantz swims quickly with grace and expertise.
Dovial Dbra Frantz swims quickly with grace and expertise.

Didn't see a moose, though. Come on, Maine.
DONE
You typed 18 words in 18.1 seconds.
Your overall wpm is 60
You made 2 mistakes, so your adjusted wpm is 53
>>>
```

Programming #2

- Filename (functions): `baseball.py`
- Filename (driver): `sox18.py`
- Filename (test suite): `test_baseball.py`

Dang, the Red Sox had a great 2018, didn't they?! As you probably know, many industrious people keep all kinds of statistics on baseball games and baseball players, and we'll do something similar for this homework. (But less intense, though. Some of those people are nuts.)

The code you'll find below initializes two lists: one containing a 'W' for each win and 'L' for each loss, and the other containing the number of runs the Sox scored in each game in the regular 2018 season. These lists are aligned with each other: The values at index 0 mean we lost that game and scored 4 runs; the values at index 1 mean we won that game and scored 1 run; and so on.

Your job with this assignment is to use those lists to answer the following questions (and print the results to the terminal):

1. What was the Red Sox win/loss record in 2018? I.e., how many entries in the first list have a 'W', and how many have an 'L'?
2. What is the average number of runs scored per game?
3. How many games did we win but scored exactly 1 run?
4. How many games did we lose but scored at least 6 runs?

Ultimately, we want to answer these four questions for the Sox '18 regular season, and your driver should use the lists provided below. But keep your functions as generic as possible!

For full credit, your program must:

- Define any functions in **`baseball.py`**, and the driver with main only in **`sox18.py.`**
- Answer all four questions above in order, in a clear, readable way. When I execute **`sox18.py`**, I see the answers to my questions right there on the terminal, and nothing else.
- Calculate and report the average as a float, rounded to the nearest 100th, but everything else should be a whole number.
- Use the **`for..range`** loop but no other loops, and no list functions besides **`len`**.
- Write a thorough test suite:
    - Test each function with at least 4 different inputs
    - Test functions with non-Red-Sox data. The functions are definitely related to baseball, but should work for any team's record from any year.
    - Print the input to be tested, expected output, and outcome (SUCCESS/FAILURE) from every test you run, and the total number of tests passed/failed.

You may assume that both lists contain good data: There is nothing other than a 'W' or 'L' in the first, and nothing other than a non-negative whole number in the second, and they are both the same length. However, in your test suite, you should consider the case of an empty list, because it's totally valid to run your code before the season starts. Make sure you don't divide by zero, which would cause an error. Instead, just return a reasonable value (like 0) if an input list is empty.

Here is the Red Sox data. There are 162 games in the regular season, so it'll be hard to figure out the final answers by hand. For testing your functions, use smaller (much smaller!) test inputs to verify that everything is working. Copy-and-paste the lists below and use them as constants above your main function:

```
RECORD = ['L', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'L', 'W', 'W',
          'W', 'W', 'W', 'W', 'W', 'W', 'L', 'L', 'L', 'W', 'W', 'L', 'L',
          'W', 'W', 'L', 'W', 'L', 'W', 'W', 'W', 'L', 'L', 'W', 'L', 'W',
          'W', 'L', 'L', 'W', 'W', 'L', 'W', 'W', 'W', 'W', 'L', 'W', 'W',
          'L', 'W', 'W', 'W', 'L', 'L', 'W', 'W', 'W', 'W', 'L', 'L', 'W',
          'L', 'W', 'W', 'W', 'W', 'L', 'L', 'W', 'L', 'L', 'W', 'W', 'L',
          'W', 'W', 'W', 'W', 'L', 'W', 'L', 'W', 'W', 'W', 'W', 'W', 'W',
          'W', 'W', 'W', 'W', 'L', 'W', 'W', 'W', 'L', 'W', 'W', 'L', 'L',
          'W', 'W', 'W', 'W', 'L', 'W', 'W', 'W', 'W', 'W', 'W', 'L', 'W',
          'W', 'W', 'W', 'W', 'L', 'W', 'W', 'L', 'L', 'L', 'W', 'W', 'L',
          'L', 'L', 'W', 'W', 'W', 'L', 'W', 'L', 'W', 'W', 'W', 'L', 'L',
          'W', 'W', 'W', 'W', 'L', 'W', 'W', 'L', 'L', 'W', 'W', 'L', 'L',
          'W', 'W', 'L', 'L', 'L', 'W']

RUNS = [4, 1, 3, 2, 7, 4, 3, 10, 8, 14, 7, 6, 7, 10, 3, 10, 9, 8, 7, 0,
        1, 3, 4, 5, 3, 6, 4, 10, 6, 5, 5, 5, 6, 6, 2, 6, 5, 3, 5, 5, 5,
        3, 6, 6, 4, 6, 5, 4, 4, 3, 6, 8, 1, 8, 8, 6, 2, 3, 5, 9, 6, 7,
        2, 0, 4, 2, 2, 6, 5, 2, 6, 0, 9, 2, 1, 9, 14, 2, 5, 9, 9, 4, 1,
        11, 1, 4, 11, 3, 10, 15, 7, 5, 8, 4, 6, 7, 6, 5, 1, 0, 9, 5, 6,
        1, 4, 10, 3, 2, 1, 15, 4, 4, 5, 10, 10, 5, 19, 5, 6, 4, 2, 4, 7,
        5, 0, 4, 3, 10, 7, 3, 1, 1, 8, 14, 9, 1, 6, 0, 8, 5, 9, 3, 3, 6,
        7, 1, 4, 0, 5, 4, 2, 1, 11, 7, 4, 3, 6, 19, 3, 6, 5, 10]
```

AMAZING points for this program:

- Make a clean distinction between general and specific: The functions defined in `baseball.py` should work with any inputs, for any team, any year, with lists of any size. The driver should use the functions you've defined to answer the four key questions.
- Make your main function very short. Like 10 lines or so, not including comments.
- Write your code such that, if I changed question 4 to be ("How many games did we lose but scored at least 7 runs?") it would require only one teeny, tiny change, and only in main. Same with a similar change to question 3.

Program #3

- Filename: `pig.py`

For this assignment, you'll implement a dice game called PIG, pitting two human players against each other. PIG is played with a standard six-sided die and proceeds as follows:

- Each player starts with zero points, and the first to 20 points wins.
- The current player chooses to roll or hold.
- If they choose roll:
    - The six-sided die is rolled
    - The value of the die is added to their round points.
    - The player goes again UNLESS...
    - ...If the value on the die is 1, then the round is over and the player has lost all the points they accumulated in the round.
- If they choose hold:
    - The points accumulated in the round are added to the player's overall score.
    - The round is over.
- When the round ends (either because the player rolled a 1 or they chose to hold), it becomes the other player's turn unless somebody has won.
- We check to see if someone won at the end of each round (which is kind of unfair, because if Player One gets 20 points on the first round, then Player Two never gets a chance, but oh well tough for them).

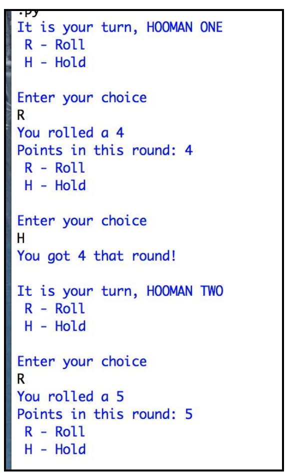
Requirements:

- Prompt each player to either R (roll) or H (hold).
- If they enter anything else, continue prompting them until they enter a correct input.
- On a roll, randomly generate a value 1-6 to represent the die.
- End the round when the die roll is value 1 (round points are lost), or the player chooses Hold (round points are added to that player's overall score).
- End the game one either player has 20 or more points and announce the winner.
- Report everything as you go along -- what the die roll was, number of points so far, etc.
- As always with your CS5001 programs, your input/output must be friendly and informative.

AMAZING points:

- Allow them to enter upper or lowercase letters. R/r for roll, and H/h for hold.
- Make your code extensible enough that we could add an arbitrary number of players with minimal coding-pain. (Consider using a list to store the player names and another list for the number of points each player has.)

A sample run of your program might look something like this (except we set the winning threshold to 10 points for the example, instead of 20, so it wouldn't go on too long).



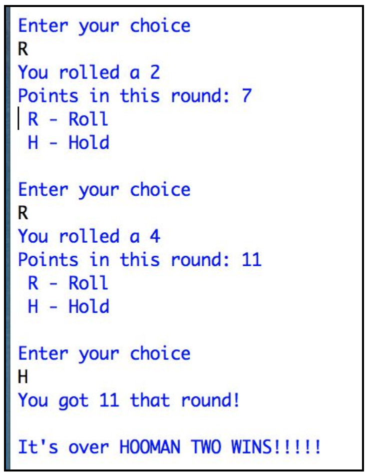
```
.py
It is your turn, HOOMAN ONE
 R - Roll
 H - Hold

Enter your choice
R
You rolled a 4
Points in this round: 4
 R - Roll
 H - Hold

Enter your choice
H
You got 4 that round!

It is your turn, HOOMAN TWO
 R - Roll
 H - Hold

Enter your choice
R
You rolled a 5
Points in this round: 5
 R - Roll
 H - Hold
```



```
Enter your choice
R
You rolled a 2
Points in this round: 7
 R - Roll
 H - Hold

Enter your choice
R
You rolled a 4
Points in this round: 11
 R - Roll
 H - Hold

Enter your choice
H
You got 11 that round!

It's over HOOMAN TWO WINS!!!!!
```