ECS 32A: Programming Assignment #2

Instructor: Aaron Kaloti

Summer Session #1 2020

Contents

1 Changelog

You should always refer to the latest version of this document.

- v.1: Initial version.
- v.2: Added autograder details.

2 Due Date

This assignment is due the night of Saturday, July 4. Gradescope will say 12:30 AM on Sunday, July 5, due to the "grace period" (as described in the syllabus). *Do not rely on the grace period for extra time; this is risky.*

Some students tend to email me very close to the deadline. This is also a bad idea. There is no guarantee that I will check my email right before the deadline.

3 General Submission Requirements

Use whichever environment/editor that you prefer, so long as it produces a Python file. Make sure that you submit the Python files with the correct names to Gradescope; the Python file names are given in each part below. These names must match *exactly*. You may submit infinitely many times to Gradescope, before the deadline.

Once the deadline occurs, whatever submission is your **active submission** will be the one that dictates your final score. By default, your active submission will be your latest submission. However, you can change which submission is your active submission, which I talked about during the Thursday (6/25) lecture.

The output of your programs needs to match the expected output ***exactly***.

4 Manual Review

There will be no manual review for points. However, I plan to have one of the TAs review some aspects of your submission and provide free advice/suggestions that you can use to improve your coding.

5 Grading Breakdown

This assignment is worth 7% of your final grade and will be worth 70 points in the autograder. Here is the breakdown of the 70 points by part:

- Part #1: 5
- Part #2: 10
- Part #3: 10
- Part #4: 5
- Part #5: 15
- Part #6: 5
- Part #7: 10
- Part #8: 10

6 Problems

For parts #1 through #5, you should not need to use any loops, just conditional statements.

6.1 Part #1

File: `part1.py`

Write a Python program that takes as input a number of inches and then prints the corresponding number of centimeters. There are 2.54 centimeters per inch.

If the user enters a negative number of inches, then instead of converting this number to centimeters, your program should print "ERROR" and nothing more.

You may assume the user will only enter an integer.

Here are some examples of how your program should work:

```
Enter number of inches: 2
Number of centimeters: 5.08
```

```
Enter number of inches: 5
Number of centimeters: 12.7
```

```
Enter number of inches: 0
Number of centimeters: 0.0
```

```
Enter number of inches: -1
ERROR
```

6.2 Part #2

File: `part2.py`

Write a Python program that prompts the user to enter three strings. The program should then tell the user the number of times that they entered the string "blah".

Below are some examples of how your program should work. Make sure that your program says "time" instead of "times", if the user only enters "blah" once. Be careful about copy/pasting the below; when you copy/paste certain marks from a PDF (e.g. single quotation marks), the formatting can sometimes be bizarre such that when pasted, the mark is not recognized as what it's supposed to be (e.g. the copy/pasted single quotation mark might be seen as some bizarre character rather than as a single quotation mark); it's a weird PDF quirk.

```
Enter first word: hi
Enter second word: there
Enter third word: blah
The word "blah" was entered 1 time. That's great.
```

```
Enter first word: bla
Enter second word: blah
Enter third word: blah
The word "blah" was entered 2 times. That's great.
```

```
Enter first word: blah
Enter second word: blah
Enter third word: blah
The word "blah" was entered 3 times. That's great.
```

In the example below, the first string entered by the user was technically "blah blah". You should not have to do anything special to handle this case, as `input(...)` already grabs the entire string that is entered by the user on the same line, even if that string contains more than one word.

```
Enter first word: blah blah
Enter second word: hi
Enter third word: there
The word "blah" was entered 0 times. That's great.
```

6.3 Part #3

File: `part3.py`

Write a Python program that prompts the user for a number of seconds and converts that number to the corresponding number of minutes and leftover seconds. If the number of minutes is 1, then "1 minute" should be printed instead of "1 minutes"; similar logic follows for if the number of leftover seconds is 1.

Hint: Use the modulo (`%`) operator. Although it is possible to do this part with a loop, you shouldn't.

You may assume the user will only enter an integer.

Here are some examples of how your program should work:

```
Enter number of seconds: 10
That is 0 minutes and 10 seconds.
```

```
Enter number of seconds: 89
That is 1 minute and 29 seconds.
```

```
Enter number of seconds: 58
That is 0 minutes and 58 seconds.
```

```
Enter number of seconds: 75
That is 1 minute and 15 seconds.
```

```
Enter number of seconds: 181
That is 3 minutes and 1 second.
```

```
Enter number of seconds: 61
That is 1 minute and 1 second.
```

6.4 Part #4

File: `part4.py`

Write a program that asks the user to enter three integers. The program should tell the user if the first integer that they entered is between the second and third integers, where the second integer is included in the range and the third integer is excluded; see the below examples for what I mean.

Below are some examples of how your program should behave.

```
Enter target number: 5
Enter start of range: 2
Enter end of range: 8
Within range!
```

```
Enter target number: 5
Enter start of range: 5
Enter end of range: 8
Within range!
```

```
Enter target number: 5
Enter start of range: 3
Enter end of range: 5
Not within range...
```

```
Enter target number: 10
Enter start of range: -3
Enter end of range: 7
Not within range...
```

```
Enter target number: 100
Enter start of range: 79
Enter end of range: 128
Within range!
```

6.5 Part #5

File: `part5.py`

Write a program that asks the user to enter a shape. If this shape is not one of "square", "rectangle", or "circle", then the program should tell the user that their choice was invalid.

Square: If the user selected the square, then the program should prompt the user for the length of the square and tell the user the area of the square. You may assume that the user will always enter a positive integer for the length.

Rectangle: If the user selected the rectangle, then the program should prompt the user for the height and width of the rectangle and tell the user the area of the rectangle. You may assume that the user will always enter positive integers for these values.

Circle: If the user selected the circle, then the program should prompt the user for the radius of the circle (which you may assume will always be a positive integer). The program should then ask the user whether they want to see the circle's circumference or the circle's area and then show the user the appropriate value. If the user enters an invalid choice here, they should be told this. To avoid having to worry about floating-point numbers, you may assume that the value of π is 3.

Below are several examples of how your program should behave.

```
Enter a shape: square
What is the length of the square? 3
The area of the square is 9.
```

```
Enter a shape: 6
Invalid shape.
```

```
Enter a shape: square
What is the length of the square? 6
The area of the square is 36.
```

```
Enter a shape: rectangle
What is the height of the rectangle? 5
What is the width of the rectangle? 4
The area of the rectangle is 20.
```

```
Enter a shape: circle
What is the radius of the circle? 2
Enter 'c' for circumference or 'a' for area: c
The circumference is 12.
```

```
Enter a shape: circle
What is the radius of the circle? 3
Enter 'c' for circumference or 'a' for area: a
The area is 27.
```

```
Enter a shape: circle
What is the radius of the circle? 4
Enter 'c' for circumference or 'a' for area: idk
Invalid choice.
```

6.6 Part #6

File: `part6.py`

Write a program that asks the user for a number N (which you may assume is always a positive integer) and then prints the message "Hello, World!" a total of $N \cdot 2 - 1$ times. After that, the program should tell the user to have a good day.

You must use a `while` loop for this part.

Below are some examples of how your program should behave.

```
Enter N: 1
Hello, World!
Have a good day.
```

```
Enter N: 3
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Have a good day.
```

```
Enter N: 4
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Have a good day.
```

6.7 Part #7

File: `part7.py`

Write a program that asks the user for an integer N and prints the N highest odd negative numbers in decreasing order, starting with -1. N will always be positive.

You must use a `while` loop for this part.

Hint: Use three variables. Some students have tried to make the counter variable serve as both a normal counter variable *and* the value to print (i.e. -1 and then -3 and then -5, etc.). Overloading *one* variable like this is *doable* but *difficult*, so I would personally recommend using two variables for these purposes. Moreover, note that in many of the loops we have seen, the counter variable has been printed; that need not always be the case (review the three essential parts of a `while` loop).

Challenge (optional): Do this part with two variables.

Below are some examples of how your program should behave.

```
Enter N: 3
-1
-3
-5
```

```
Enter N: 8
-1
-3
-5
```

```
-7
-9
-11
-13
-15
```

6.8 Part #8

File: `part8.py`

Write a program that keeps prompting the user to enter a string until the user enters the string "Done". After this, the program should print the concatenation of all strings entered by the user in reverse order. By this, I mean that the first string entered by the user should be at the end of the final printed string, *not* the beginning; similarly, the last string entered by the user (which will always be "Done") should be at the beginning of the final printed string, *not* the end.

Hint: Recall from lecture that the addition operator (`+`) can be used for concatenating two strings.

You must use a `while` loop for this part.

Below are some examples of how your program should behave.

```
Enter string: First
Enter string: Second
Enter string: Third
Enter string: Fourth
Enter string: Fifth
Enter string: Sixth
Enter string: Seventh
Enter string: Done
DoneSeventhSixthFifthFourthThirdSecondFirst
```

```
Enter string: My
Enter string: name
Enter string: is
Enter string: Aaron
Enter string: Kaloti
Enter string: Done
DoneKalotiAaronisnameMy
```

```
Enter string: My
Enter string: favorite
Enter string: number
Enter string: is
Enter string: 78
Enter string: Done
Done78isnumberfavoriteMy
```

```
Enter string: Done
Done
```

7 Autograder Details

7.1 Error Message When Fail Test Case

The autograder tries to give you a good idea of where your output differs from the expected output. Let us know if you have an issues interpreting it. Do note that for input prompts, the expected output will seem to have them on the same line; this is due to something called output redirection that you will learn about in ECS 32C or ECS 36A. For example, if you get a message about a difference in output for part #2, the input prompts may appear all on the same line like so:

```
Enter first word: Enter second word: Enter third word:
```

In reality, they are not on the same line, so do go off of the examples shown for each part in this document. The fact that the input prompts are on the same line also does not stop the autograder from telling you if you have misspellings in your input prompts, which is helpful.

7.2 Released Test Cases vs. Hidden Test Cases

You will not see if you have passed the last test case for each part until after the deadline; these are the hidden test cases, and Gradescope will not tell you whether you have gotten them correct or not until after the deadline. One purpose of this

is to promote the idea that you should test if your own code works and not depend solely on the autograder to do it. The other purpose is to make it easier to detect when students "game" the test cases (see the next subsection). That said, for this particular programming assignment, it is quite likely the case that if you got all of the visible test cases correct, you probably got the hidden ones correct as well, but the only way to know for sure is to try a variety of inputs on your code. Unfortunately, Gradescope will display a dash as your score until after the deadline, but you can still tell if you have passed all of the visible test cases if you see no test cases that are marked red for your submission.

Below are the inputs for the visible test cases.

7.2.1 Part #1

Case #1: 4

Case #2: 0

Case #3: -1

7.2.2 Part #2

Case #1:

```
blah
blah
bla
```

Case #2:

```
spiderman
batman
spiderman
```

Case #3:

```
batman
robin
blah
```

7.2.3 Part #3

Case #1: 10

Case #2: 120

Case #3: 61

7.2.4 Part #4

Case #1:

```
7
9
15
```

Case #2:

```
100
80
127
```

Case #3:

```
10
10
15
```

Case #4:

```
15
10
15
```

7.2.5 Part #5

Case #1:

```
blah
```

Case #2:

```
square
7
```

Case #3:

```
rectangle
8
7
```

Case #4:

```
circle
4
a
```

Case #5:

```
circle
20
c
```

Case #6:

```
circle
17
whatever
```

Case #7:

```
circle
7
c
```

7.2.6 Part #6

Case #1: 1

Case #2: 6

Case #3: 7

Case #4: 18

7.2.7 Part #7

Case #1: 3

Case #2: 4

Case #3: 6

Case #4: 8

7.2.8 Part #8

Case #1:

```
I
am
cool
Done
```

Case #2:

```
Dog
Cat
Snake
Dragon
Done
```

7.3 What is "Gaming" the Test Cases?

As an example, for part #7, you can see that case #1 is the integer 3. The output for this should be -1, -3, and -5. To "game" this test case is to use a conditional statement to check if the user entered a specific test case input and then print the expected test case output without having any code that solves the problem generally. Here is how that would look:

```
print("Enter N: ")
if num == 3:
    print(-1)
    print(-3)
    print(-5)
```

To be clear, doing the above is strictly prohibited, and one of the TAs will check if you do this. If you "game" the test cases, you risk getting a zero on this entire assignment and potentially even being reported to the OSSJA.