CS5001: Lab 3. Due on Friday, Sep-29-2023.

Name(s): Xujia Qin

Email(s): qin.xuj@northeastern.edu

You can work on this lab either individually or in small group of two or three students. If working in a group, include names of all the students in the submission PDF.

Getting credit for this lab. This lab handout has several empty boxes that prompt you to answer a question. As part of the lab, you are to write the answers to these questions inside the boxes/blanks. When you are finished, you should create a PDF and upload it on Canvas. If you don't finish, you have until 11:59 PM on Friday, Sep-29-2023 to submit.

What computer to use? If your primary computer is a laptop, bring it to the lab to work on, as lab is an excellent opportunity to get started with Python on your machine. You should follow the instructions on the course website. Ask a TA for help if you have problems with your installation. If you prefer, you could also use one of the machines in the lab room to work on this lab assignment.

Lab Materials. Lab materials can always be found on Canvas under the appropriate lab posting.

For today's lab, you need this handout (which is also online), plus the files Hyphenator_broken.py, Hyphenator_even.py.

1. Print statements for checking that variables have the expected values

Here is a slightly different version of the hyphenation program you saw in class.

```
# Hyphenator_broken.py
# CS5001 instructor (ad.mishra)
# Jan 2023

""" Inserts hyphens into a non-empty even-length input string as follows: The hyphen splits the
first and second halves."""

## We've added a number of so-called "debugging" print statements here


s = input('Enter an even-length string (remember to put quotes around it): ')

n = len(s)

print ('n is',n)

if n%2 == 0:     # Line A

        # s has even length
        m  =  int(n/2)
        print ('even case. m is',m first = s[0:m])
        # Line B
        print ('even case. first is',first)
        second = s[m:]
        # Line C
        print ('even case. second is',second h = first + '-' + second)
# final output
print (s,'becomes',h)
```

Notice the inclusion certain print statements that are designed to show you the contents of key variables:

- print ('n=',n)
- print ('even case.   m=',m)
- print ('even case. first=',first)
- print ('even case.   second=',second)

These are debugging print statements. (The final line print (s,'becomes',h) is not a debugging print statement, but rather, one that creates the desired final result of the program.)

In the command shell, see what the program does: run the program by typing python Hyphenator even.py. When asked for input, try 'abcdef' ; the last line of the output should say that the program turns this into 'abc-def'.

Now, let's artificially insert some errors and see how these print statements show that some variables then no longer have the "right" values in them.

Open Hyphenator_even.py in an editor.

(1). In Line B, assign s[:m-1] to first instead of s[:m]. Then, in the command shell, run Hyphenator even.py. (That is, type python Hyphenator_even.py in the command shell. Don't forget to save the file in IDLE Edit first.) Give the input 'abcdef'.

Which variable has an unexpected value, and what should the variable's value have been? Which print line tells you this? Answer in the blank space below: **(25 points)**

first has an unexpected value, the variable's value have been 'abc but now it is 'ab
The statement: print ('even case. first is',first), tells me this.

(2). Fix the mistake you made above. Now, introduce a new artificial mistake via editor: in Line C, assign s[m+1:] to second instead of s[m:]. Suppose you run Hyphenator even.py and give it input 'abcdef'.

Which variable has an unexpected value, and what should the variable's value have been? Which print line tells you this? Answer in the blank space below: **(25 points)**

second has an unexpected value, the variable's value have been def' but now it is ef'
The statement: print ('even case. second is',second), tells me this.

2. Finding errors using print statements

Now, we give you practice in adding print statements to see if there are errors in a program. This is something we ask you to do in HW1, as well. The point: learn to use print statements to figure out if your code is working or not.

Read the docstring for Hyphenator_broken.py (the comment in triple quotes) in the following program to understand what the program is supposed to do.

```
# Hyphenator_broken.py
# CS5001 instructor (ad.mishra)
# Jan 2023

""" Inserts hyphens into a non-empty odd-length input string as follows:

A hyphen is inserted on either side of the middle character.

Example: "abcde" becomes "ab-c-de"

"""


### This program intentionally has at least one error in it!

s = input('Enter an odd-length string (remember to put quotes around it): ')


n = len(s)


m = int(n/2)


first = s[0:m-1]


middle = s[m+1]


second = s[m+1:]


h = first+'-'+middle+'-'+second
```

final output

print (s,'becomes',h)

Run program Hyphenator_broken.py and give it input 'abcde' to see that it currently doesn’t work correctly.

Open the file Hyphenator_broken.py in a Python editor. Insert enough print statements so that when you run the program in the command shell, you know what lines are incorrect. Write in the blank spaces of the code above what print statements you included.

Have print statements with appropriate documents (like, print (“middle is: ”, middle) instead of just ”print (middle)”).

Run your altered program in the command shell. From the output of your print statements, explain what are the errors in the program.

Answer in the blank space below: **(50 points)**

```
# Hyphenator_broken.py
# CS5001 instructor (ad.mishra)
# Sep 2023

""" Inserts hyphens into a non-empty odd-length input string as follows:
A hyphen is inserted on either side of the middle character.

Example: "abcde" becomes "ab-c-de"

"""
### This program intentionally has at least one error in it!

s = input('Enter an odd-length string (remember to put quotes around it): ')

n = len(s)
print('Length of the string:', n)


m = int(n/2)
print('Odd case, middle index:', m)


first = s[0:m]
print('Odd case, first part:', first)


middle = s[m+1]
print('Odd case, middle character:', middle)


second = s[m+1:]
print('Odd case, second part:', second)


h = first+'-'+middle+'-'+second

# final output
print (s,'becomes',h)
```

Explanation:

The string slicing of first and middle parts are wrong.
first = s[0:m-1]
middle = s[m+1]
second = s[m+1:]

should be modifies as

first = s[0:m]
middle = s[m]
second = s[m+1:]

Because we need to take the index from 0 to index m-1(included m-1) as our first part, so first should be set as s[0:m], and the middle one as s[m], second part as s[m+1:] , which takes the index m+1 to the end of the string.

String slicing: left side index m included, but right side n excluded (e.g.[m, n]), perhaps the error comes from the misunderstanding.