Your Name:	TJ Sipin
Perm #:	3430501

This homework and its due date are posted on the course website: https://ucsb-cs8.github.io/w20. Submit a PDF file of this assignment following the guidelines posted on the website.

h08: Perkovic 4.1 (Strings, Revisited), 4.2 (Formatted Output), Perkovic 4.3 (Files), 5.3 (2D lists)

0. (5 points) Use the provided homework template (without any blank pages), filling out all requested fields, and correctly submitting a legible electronic document on Gradescope before the due date. By submitting this document you agree that this is **your own work** and that **you have read** the policies regarding Academic Integrity: https://studentconduct.sa.ucsb.edu/academic-integrity.

1. (4 pts) In Section 4.1, the author explains *escape sequences*. What is an escape sequence?
Using an escape sequence allows the coder to use certain characters without it being a part of Python functions. For example, using \' allows a single quote to be used in a string without being used as string delimiter.

How is an escape sequence interpreted by print()?
\n starts a new line, while \' or \" makes sure that the ' or " is not used as a string delimiter.

2. (4 pts) Show two ways in which you can store the string that would display `I'm reading "The Alchemist"` when printed.

`text =` 'I\'m reading "The Alchemist"'

`text =` "I'm reading \"The Alchemist\""

3. (6 pts) In Section 4.1, the author also discusses how to store multiline strings. Show **two ways** in which we can use **a single assignment** statement to store on separate lines (as shown below) **the translation** of the following famous haiku by Bashō. (Note: make sure there is *NO newline* before the first line of the haiku!)

Original Haiku	Translated Haiku
初しぐれ猿も小蓑をほしげ也 はつしぐれさるもこみのをほしげなり	the first cold shower even the monkey seems to want a little coat of straw

```
haiku = '''
the first cold shower
even the monkey seems to want
a little coat of straw
'''
```

```
haiku = "the first cold shower\neven the monkey seems to want\na little coat of straw"
```

4. (11 pts) In Section 4.1, the author discusses the slicing operator (:) and how it is used with indexing. Given the following string and the corresponding substring from it, complete the following explanation of how slicing works:

```
hamlet = "To be or not to be"
print(hamlet[3:5])
```

The expression `hamlet[3:5]` evaluates to the slice of the string `hamlet` starting at index 3____

and ending before__________ index 5______.

In order to obtain a slice that ends at the last character of a string, we must drop

the ending index____.

Illustrate what the above statement means by extracting the second "`be`" from the string `hamlet`.

`hamlet[`16:________________`]`

Write the expression that will extract the substring “To be” without explicitly specifying the starting index:

```
hamlet[:5                  ]
```

As the author shows on page 95, you can apply the slicing operator to lists. Let’s use the string’s `split()` function (p. 113) to turn the string into a list of words (by splitting the characters based on the spaces separating them).

Use the slicing operator to extract the first two and the last two values in the list. (You can use print() to check the returned values.)

```
hamlet_words = hamlet.split()

hamlet_words[0:2         ] # the first two words

hamlet_words[-2:         ] # the last two words using negative indexing
```

5. (6 pts) In Section 4.2, revisits the example of using optional arguments within the print() function. Given the following list of coordinates, in the format `[x1, y1, x2, y2]`, fill in the blanks so that the resulting code prints these values, such that they are formatted as `(xn, yn)`, with each coordinate on their own line. (Note: there is no space between parentheses and values).

```
coord = [2, 5, 6, 7, 11.2, 8.1]

for i in range(0, len(coord), 2                    ):

   print('(', coord[i     ], ', ', coord[i+1   ], ')', sep=''          )
```

Output:

```
(2, 5)
(6, 7)
(11.2, 8.1)
```

What do you need to add in the above print() statement, so that instead of outputting the result on separate lines, the lines above are separated by the tab characters when printed. (Hint: See Practice problem 4.3).

end = '\t'

6. (5 pts) In Section 4.2 (starts on p.100), the author explains how to use the `format()` function to generate the well-formatted output. Re-write the example above using the `format()` string with *positional arguments* such that the output prints the result with each coordinate on their own line.

```
coord = [2, 5, 6, 7, 11.2, 8.1]

for i in range(0, len(coord), 2):

print('({0}, {1})'.format(coord[i],coord[i+1]))
```

7. (4 pts) In Section 4.2 (starts on p.100) carefully to make sure you understand how `format()` operates. In the book, the author shows that the arguments inside the curly braces of a format string can specify how the value that mapped to the curly brace placeholder should be presented. List what the author says we can specify as part of the format string (see p. 103):

Strings and integers.

8. (11 pts) Following the example in the book on p.103, let's deconstruct the following statement

```
>>> '{0:3},{1:5}'.format(111, 222)
```

In this example, we are printing integer values 111 and 222. The format string has a placeholder for 111 with '0:3' inside the braces. The 0 refers to the first argument of the format() function, which is 111. Everything after the ':' specifies the formatting of the value. In this case, 3 indicates that width of the placeholder should be 3. Since 111 is a three-digit number, extra blank spaces are are added in front. The placeholder for 222 contains '1:5', so an extra 2 blank spaces are added in front.

9. (7 pts) What happens in this example?

```
>>> '{0:3},{1:5}'.format(1234, 56)
```

In this example, we are printing integer values 1234 and 56. The format string has a placeholder for 1234 with '0:3' inside the braces. The 1 refers to the **second** argument of the format() function, which is 56. Everything after the ':' specifies the formatting of the value. In this case, 5 indicates that the width of the placeholder should be 5.

10. (7 pts) Write down the output below, placing each character into its own box to show the exact spacing. Try to figure it out by hand before checking your answer online. If you are writing on paper and mess up the first grid, use the second.

```
>>> '{0:3},{1:5}'.format(1234, 56)
```

'	1	2	3	4	,				5	6	'								

What happens when the input is larger than the specified field width?

The width of the placeholder will match the length of the input.

11. (5 pts) Does the output change if we turn the input argument from

`'{0:3},{1:5}'.format(1234, 56)` into strings as shown below? Why?

`'{0:3},{1:5}'.format("1234", "56")`

It does, because the output of a string argument is different than the output of an integer argument. When it is a string, the output becomes left-justified, whereas when it is an integer, the output becomes right-justified.

12. (8 pts) Complete the following format strings to produce the **exact** output shown below.

```
>>> '{: 5.3}'.format(11/8) # a single blank at the beginning
' 1.38'
>>> '{5.2}'.format(11/8) # two blanks at the beginning
'  1.4'
>>> '{4.2}'.format(11/8) # a single blank at the beginning
' 1.4'
>>> '{:e}'.format(11/8)
'1.3750e+00'
```

13. (14 pts) In Section 4.3, the author explains file input / output operations. Using Table 4.5 in the book, complete the table below. **Pay attention to the different reading methods and return types**.

Command	**Explanation**
`infile.open`	Open the file "test.txt" for reading
`outfile.open`	Open the file "output.txt" for writing
`infile.read`()	Read characters from the file opened using `infile` until the end of the file is reached. Return characters read as a string
`infile.read`line()	Read file opened using `infile` until (and including) *the new line character* or until end of file, whichever is first. Return characters read as a string
`infile.read`lines()	Read file opened using `infile` until the end of the file. Return characters read as a list of lines ______
outfile.write(text)	Write a string stored in `text` to file opened using `outfile.`
infile.close()	Close file opened using `infile`
outfile.close()	Close file opened using `outfile`

14. (8 pts) In Section 5.3, the author discusses two-dimensional (2D) lists. Convert the following table into a 2D list called `myTable`, *storing rows as elements* in the list `myTable`.

10	11	12
20	21	22

Use the variables `row1` and `row2` to store the rows as elements in the `myTable`.

`row1 =` [10, 11, 12]

`row2 =` [20, 21, 22]

`myTable =` [row1, row2]

Now, store the elements directly in the `myTable` instead of using variables `row1` and `row2`.

`myTable =` [[10, 11, 12], [20, 21, 22]]

15. (6 pts) If you are given the following table that is now stored in `myTable`. Fill-in the blanks in the code below to output every element in that table within square brackets (as shown below on the right). (*Hint: first, write down the pseudocode to figure out what needs to happen first. Remember that* `print()` *outputs values one line at a time by default.*)

10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35

```
[10] [11] [12] [13] [14] [15]
[20] [21] [22] [23] [24] [25]
[30] [31] [32] [33] [34] [35]
```

```
for row in range(len(myTable)):
  for col in range(len(myTable[0])):
     print("{:2}".format(myTable[row][col]), end = " ")
 print()
```