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.

h06: 3.4 (Python Variables and Assignments), 5.1 (Decision Control and the `if` Statement), 5.4 (`while` Loop), 5.5 (More Loop Patterns), 5.6 (Additional Iteration Control Statements)

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: ttps://studentconduct.sa.ucsb.edu/academic-integrity.

1. (3 pts) In **Section 3.4**, the author talks about Python Variables. What kind of statement creates a variable in Python (i.e., what do you need to do to make a new variable exist in your code)?

You need the variable, the assignment operator (=), and the expression.
<variable> = <expression>

2. (8 pts) In **Section 5.1**, the author contrasts the one-way decision control structure, which we saw in Chapter 3, with the two-way decision that uses `if/else` statements.

```
#  one-way decision control
    if <condition>:
        <indented code block 1>

    <non-indented statement>
```

```
#  two-way decision control
    if <condition>:
        <indented code block 1>
    else:
        <indented code block 2>

    <non-indented statement>
```

What statement/block gets executed if `condition` is `True`, in the one-way decision structure?

<indented code block 1>

What statement/block gets executed if `condition` is `True`, in the two-way decision structure?

<indented code block 1>

What statement/block gets executed if `condition` is `False`, in the one-way decision structure?

<non-indented statement>

What statement/block gets executed if `condition` is `False`, in the two-way decision structure?

<indented code block 2>

3. (4 pts) What does `elif` stand for? How is it different from `else`?

elif stands for "else if."
It is different from "else" because you can have a condition after "elif" that must be true in order for the indented code to execute, whereas the code after "else" only executes if everything before it is false.

4. (10 pts) In **Section 5.1**, the author talks about an issue with multiway decision structures that does not exist with one- or two- way `if` statements. What is this issue?

The order in which the conditions appear is important. For example, say x = 13. If one condition is that x > 5, then that code will run. If another condition is that x > 10, then the code when x > 5 is going to be the only code that will run, as opposed to when x > 10.

Briefly explain how this issue can be fixed. Show a short example using an ***implicit*** structure.

You can make the conditions mutually exclusive explicitly, though this makes the code unnecessarily complicated.

Example of implicitly making the conditions mutually exclusive:

```
def f(x):
    if x > 10:
        print("> 10")
    elif x > 5:
        print("> 5")
```

5. (5 pts) The `while` loop is introduced in **Section 5.4**. In your own words briefly explain in what way the `while` loop is similar to the `if` statement.

A "while" statement is basically an "if" statement that keeps cycling through the code, executing only as long as the condition is evaluated as true.

6.1 (8 pts) In **Section 5.2**, p. 134-136 the author discusses the “Accumulator Pattern”, which is a very important topic in this course; one of the most important for you to master. So please read those two pages several times and try to understand every detail. The figure at the top of p. 135 shows the various stages of execution for the code on p. 134.

The code `mySum = mySum + num` takes the old value of `mySum`, adds `num` to it, and stores the result back in `mySum`. That code is done inside a `for` loop, `for num in myList`, after setting `mySum` initially to zero. The final value for `mySum` is 20.

Re-write this code to iterate through the numbers in `numList` and add them to `mySum` **using a `while` loop**.

```
numList = [3, 2, 7, -1, 9]
```

i = 0

mySum = 0

while i < len(numList):

    mySum = mySum + numList[i]

    i = i + 1

```
print(mySum) # Should produce 20
```

6.2 (3 pts) Remember your answer from Question **1**? Where does it apply in the above code?

It applies in the initial value of our index variable as well as mySum.

7. (5 pts) On p. 135, we see the intermediate values for `mySum`, namely 3, 5, 12, 11, and 10. Try to understand where those values come from as the loop progresses.

Now, imagine the same **`while` loop** was executed, but with the first line of code changed to `numList=[8, 3, 1, 2, 7]` (instead of `numList=[3, 2, 7, -1, 9]`.)
What would the successive intermediate values of `mySum` be in that case? List them below.

8, 11, 12, 14, 21

8.1 (3 pts) On pages 135-136, the author discusses accumulating a *product* instead of a *sum*. The accumulator variable is called `myProd` this time. In the version of the code that works properly, what is `myProd` initialized to, and why?

It should be initialized to 1 because if it is 0, then myProd will always result in 0 as anything multiplied by 0 is 0.

8.2 (8 pts) Re-write this code to iterate through the numbers in `numList` and multiply all the numbers in the list **using a `while` loop**. (Note that because the last statement in the given code is `print(myProd)`, you should store successive products in the variable called `myProd`; if you call that variable something else, `myProd` would not exist (see Question **1**) and the `print(myProd)` would result in an error.)

```
numList = [3, 2, 7, -1, 9]
myProd = 1
j = 0
while j > len(numList):
    myProd = myProd * numList[j]
    j = j + 1
print(myProd)
```

9.1 (1 pt) On p. 147, the author talks about infinite loops. How can you interrupt its execution?

Simultaneously type Ctrl + C

9.2 (3 pt) Why does the following code produce an infinite loop?

```
count = 1
while count > 0:
      print(count)
      count = count + 1
```

count was initially set to 1, thus it is always greater than 0 and the while loop will keep running.

9.3 (3 pt) Briefly explain why the following code produces an infinite loop, then fix the code to correctly output 0 1 2.

The accumulator line count = count + 1 is not within the while loop indent, thus the while loop will run forever.

Incorrect code	**Corrected code**

`count = 0`
 	
`while count < 3:`
 	
`print(count)`
 	
`count = count + 1`
	count = 0
	
	while count < 3:
	
	print (count)
	
	count = count + 1

9.4 (3 pt) Briefly explain the following code produces an infinite loop, then fix the code to correctly output only odd numbers.

The accumulator line is not indented correctly. It must be within the while loop by one indent.

Incorrect code	Corrected code

`count = 0`
 	
`while count < 6:`
 	
`  if count % 2 == 1:`
 	
`  print(count)`
 	
`  count = count + 1`
	count = 0
	
	while count < 6:
	
	if count % 2 == 1:
	
	print (count)
	
	count = count + 1

10.1 (2 pts) On p. 151, the author reminds us that "In Python, every function definition `def` statement, `if` statement, or `for` or `while` loop statement must have a body (i.e., a nonempty indented code block)." What happens if the code block is missing?

You get a syntax error.

10.2 (2 pts) How does the `pass` keyword help with the above issue? What does `pass` do?

pass is used when Python syntax requires code, but basically it does nothing: it just tells Python to pass over the def statement, if statement, or for or while loop statement.

It's also useful as a stand-in for when a code body has not been implemented yet.

10.3 (5 pts) The author says that "The `pass` statement is also useful when a code body has not yet been implemented." Usually though, the presence of the `pass` statement is an indication that the code can be re-written to eliminate the "empty block".
Using the example in the book that illustrates the usage of the `pass` statement "in a code fragment that prints the value of `n` only if the value of `n` is odd", rewrite the conditional statement so that the code does nothing for even number `n` without using the `pass` statement. (You should need only two lines of code.)

```
if n % 2 == 1:
    print(n)
```

The following exercises are *optional*. Submit them on Gradescope under hw06_bonus.
Make sure you name your file **hw06_bonus.py (note the 06 part!).**

Bonus 1) Turn the code from **Question 6.1** into a function `sum_greater`, which takes a list `myList` and a value `val` as input parameters, and returns the sum of all elements in `myList` that are greater than `val`. Return `None` if none of the values are greater than `val`. (Remember that docstrings are great to have.)
Include at least two pytest functions to verify the `sum_greater` function's accuracy.

Bonus 2) Create a function `greater_list`, which takes a list `myList` and a value `val` as input parameters, and returns a new list that contains all elements from `myList` that are greater than `val`. Return `None` if none of the values are greater than `val`.
Remember your answer from Q.**1** and a function `append()` from Section 2.3. An example from p.148 might be helpful.
Include at least two pytest functions to verify the `greater_list` function's accuracy.

Do not submit your written answers here. See instructions above.