Conceptual

1. Given the table below, fill in the correct values and type for the matching expression. If there is an error, write "Error" in the value column. (You may leave the type column blank, and you do *not* have to explain the error.)

Expression	**Value**	**Type**
`{"a": 10, "b": 20}["c"]`		
`[1, 2, 3][1] + (4, 5)[0]`		
`"10" + 6 / 2`		
`len([1, 2, 3, 4]) / 2`		
`x = set([5, 6, 7, 8])` `x.add([9, 10, 11, 12])`		

---

2. You have a large file called `element.txt` that contains information about all of the elements in the periodic table in the following format:

```
H Hydrogen 1.008 amu
He Helium 4.003 amu
…
```
(more lines not shown)
```
…
Rn Radon 222.01 amu
```

You wish to break up the longer file into smaller `.txt` files that contain information about each element and have come up with the following lines of code

```
file = open("elements.txt")
for line in file:
     element = open("info.txt", "w")
     element.write(line)
file.close()
```

a. Why would this code not work when trying to split up the longer element.txt file into smaller files for each element?

b. What would be inside the `info.txt` file as the code it is written? (write it out or describe it)

c. While running the first line of code (`file = open("elements.txt")`) you get a `FileNotFoundError`. You know for sure that you saved `elements.txt` somewhere, although you forgot exactly where. Given what you know about file I/O, why did you receive this error even though `"elements.txt"` exists on your computer?

---

3. All of the expressions in the left hand column are going to be printed, what value would be output to the terminal if printed. If evaluating the expression would result in an error, write "Error" in both the value and type columns. If you are writing a string please include a "_" where a space should be.

Expression	**Value After Evaluating**	**Type**
`100 - (27 // 2)`		
`27 % 5 - 10`		
`'Python in CSE' + 160`		
`['hello', 1, 3, [5, 6], 22][1:4:2]`		
`my_lst = [22, 29, 23][3]` `[5, 4, 7, 1, 8, 9].append(my_lst)`		

`[5, 4, 7, 1, 8, 9].sort()`		
`'Good' + 'Job!'`		

---

4. For each of the below expressions, write what the expression evaluates to and the type of that value. You should assume that variables have been declared and assigned as follows:

```
a = "cse160 is fantastic"
b = 10
c = 3.6
d = True
e = ["you're", "going", 2, "be", "amazing"]
f = [10, 7, 3, 6, 9]
g = {"is": "python", 2.0: "fun"}
```

If evaluating the expression would result in an error, write "Error" in both the value and type columns.

Expression	**Value of `output`**	**type**
`output = 8 + b * c`		
`output = a[3]`		
`output = str(b) + " " +  e[2]`		
`output = b % 2 == 0 and not d`		
`output = f.sort()`		
`output = a + 'Let's do it!'`		
`output = e[-1] + g["is"] + g[b / len(f)]`		

---

5. What is the **result** and **type** of the following expressions (if it is an Error indicate that):

	Result	**Type**	**Error? (yes/no)**
`20/10`			
`10/20`			
`1.0/4`			
`45 < 90`			

---

6. What is the last element generated by these range () calls:

	Last Element Generated
`range(5, 25, 5)`	
`range(14, 3, -2)`	
`range(13)`	

---

7. Given the following dictionary, write what each expression evaluates to. If an error is thrown, write "**Error**".

```
my_dict = {"one":1, 2:2, 'string':'with', 'withhold':-100, 4:16 }
```

a) `my_dict['1']`

b) `my_dict[6-2]`

c) `my_dict[-100]`

d) `my_dict[my_dict['string']+'hold']`

---

8. Suppose we have the following list:

```
mystery = [ [8], 4, [1, 5, 3, [2, 6]], [9, 7], 7]
```

Write the result of the following expressions. If an error is thrown, briefly describe the error.

a) `mystery[1]`	
b) `mystery[0]`	
c) `[8] in mystery`	
d) `8 in mystery`	
e) `mystery[2][1]`	
f) `mystery[4][0]`	
g) `mystery[-1]`	
h) `mystery[2:3]`	

i) Write code that modifies the original mystery list so that it contains the following (change in **bold**):

```
mystery = [ [8], 4, [1, 5, 3, [2, 6], 9], [9, 7], 7]
```

j) Write code that modifies the original mystery list so that it contains the following (change in **bold**):

```
mystery = [ [8], 0, [1, 5, 3, [2, 6]], [9, 7], 7]
```

---

9. Given the following variables that have been defined. What is the **result** and **type** of the following expressions (if it is an Error indicate that, and leave result and type blank):

```
x = 5
y = 11
z = 2.5
foo = True
bar = "false"
```

	Result	**Type**	**Error? (yes/no)**

`foo and x < y`			
`x / y`			
`y >= 0`			
`z / x`			
`bar[0]`			

---

10. Given the following dictionary, write what each expression evaluates to. If an error is thrown, write "**Error**".

```
my_dict = {"pie":[1, 2], "cake":[3], "candy":[7, 6], "salt":[4]}
```

a) `my_dict["cake"]`

b) `my_dict[3]`

c) `my_dict["candy"][0]`

d) `my_dict["pie"].append(9)`

---

11. Given the following dictionary, write what each expression evaluates to. If an error is thrown, write "**Error**".

```
my_dict = {5:"red", 2:"orange", 0:"green", 1:"purple"}
```

a) `my_dict[2]`

b) `my_dict[3]`

c) `my_dict[1][1]`

d) `my_dict["red"]`

---

12. For each of the expressions below, write what it evaluates to and the type of that value (float, int, string, or bool). You should specify any floating point values only to one decimal point (e.g., 0.5). Remember to add quotation marks if the expression evaluates to a string.

If an error is produced, write "Error" under the Evaluation column and leave the Type column blank.

Expression	Evaluation	Type
int(3 + 1.5)		
lst = [1, 3, 5, 7] lst[-2] ** 2		
"NCT " + 127		
"A" in "man" or 2 < 1		

(b) Given the following dictionary, write what each expression evaluates to. random_dict = {"purple": 1, 2: "blue", "three": 3, "hello": "world", 1: 100}

Expression	Evaluation	Type
random_dict["hello"]		
random_dict[100]		
random_dict[purple]		
random_dict["three"]		

---

13. In each of the following code snippets, how many times is the line "CSE 160" (as one string on the same line) printed?

Code	# times
for i in range(10): print("CSE 160")	
for i in range(0, 9, 3): print("CSE 160")	

for i in [1, 3, 5]: print("CSE 160") print("CSE 160")	
for i in range(5, 0): print("CSE 160")	
for i in "hello world": print("CSE 160")	

---

14. For each of the below expressions, write what the expression evaluates to and the type of that value. You should assume that variables have been declared and assigned as follows:

```
a = 11
b = "hello"
c = False
d = 10.5
e = 2
f = { "a": 1, 1: "a", "b": b }
```

Expression Value Type

```
a / e
e + a
a % 2 == 0 or not c
not (len(b) < 5 and d > e)
b[1] + str(d * e)
f["b"][2] + f[1]
```

---

15. For each expression on the left, write the value that x evaluates to and then the type of that

value. If the expression results in an Error, write "Error" in both columns. (You do not have to explain what the error is.)

For example:

***[Example]* Expression**	***[Example]* Value of x**	***[Example]* Type of x**
`x = 3`	3	int
`x = 3..0`	Error	Error

Expression Value of x	**Type of x**
`x = 10 * 3.7`	

`lst = [4, 3, 2, 1]`
 	
`x = lst.reverse()`	

`x = (67.3 * 10) + 3 > 65`
 	
`x = 2 ** 3 // 4`
 	
`x = 'Please don't make this mistake'`
 	
`a = [6, 5, 4, 3, 2, 1, 0]`
`b = "success"`
`c = b[a[-3]] + b[::4]`
`d = a[1] * a[2] * a[-3] * a[4 // 2]`
`x = str(c) + str(d)`	

---

16. For each of the below expressions, write what the expression evaluates to and the type of that value. You should assume that variables have been declared and assigned as follows: a = 11 b = "hello" c = False d = 10.5 e = 2 f = { "a": 1, 1: "a", "b": b }

Expression	**Evaluation**	**Type**
11 / 2		
2 + 11		
11 % 2 == 0 or not False		
not(len(‘hello’) < 5 and 10.5 > 2)		
‘hello’[1] + str(d * e)		

a = 10 b = ‘hello’ {"a": 1, 1: "a", "b": b}[‘b’][2] + f[1]		

---

17. For each of the below expressions, write what the expression evaluates to and the type of that value. You should assume that variables have been declared and assigned as follows:

```
a = "cse160 is fantastic"
b = 10
c = 3.6
d = True
e = ["you're", "going", 2, "be", "amazing"]
f = [10, 7, 3, 6, 9]
g = {"is": "python", 2.0: "fun"}
```

If evaluating the expression would result in an error, write "Error" in both the value and type columns.

Expression	**Value of `output`**	**type**
`output = 8 + b * c`		
`output = a[3]`		
`output = str(b) + " " +  e[2]`		
`output = b % 2 == 0 and not d`		
`output = f.sort()`		
`output = a + 'Let's do it!'`		
`output = e[-1] + g["is"] + g[b / len(f)]`		

Code Reading

18. The sub-question (a) on the next page uses the following code.

```
1     def category_grade(earned, possible):
2         earned_points = 0
3         total_possible = 0
4         for i in range(len(earned)):
5             earned_points += earned[i]
6             total_possible += possible[i]
7         return earned_points / total_possible
8
9     def calculate_total_grade(earned, possible, weights):
10        total = 0
11        for i in range(len(weights)):
12            category_grade = category_grade(earned[i], possible[i])
13            total += category_grade * (weights[i] / 100)
14        return total * 100
```

These functions are designed to calculate the total grade for a student based on earned points, possible points, and weights for different categories of assignments.

This code was initially written assuming `weights` was a list and that `earned_points` and `possible_points` were nested lists. Now, they have been converted to dictionaries with keys `"homework"`, `"participation"`, and `"exams"`, but the code has **not** yet been updated to reflect this change. Below are examples of how these structures looked as lists and then after the change to use dictionaries.

BEFORE (as lists)

```
earned = [
    [8, 9, 7],
    [4, 3],
    [25]
]

possible = [
    [10, 10, 10],
    [5, 5],
    [30]
```

AFTER (as dictionaries)

```
earned = {
    "homework": [8, 9, 7],
    "participation": [4, 3],
    "exams": [25]
}

possible = {
    "homework": [10, 10, 10],
    "participation": [5, 5],
    "exams": [30]
```

```
]

weights = [50, 20, 30]
```

```
}

weights = {
    "homework": 50,
    "participation": 20,
    "exams": 30
}
```

a) Identify which line(s) in the `category_grade` and `calculate_total_grade` functions need to be changed in order to make it work with `earned` and `possible` as dictionaries. Then, describe the necessary change, written as the code to replace the identified line(s).

Line number(s): _____
Change(s) required:

---

19. You are interested in whether ChatGPT overuses certain words (for example, "delve") in the answers it gives. Given a ChatGPT answer and a list of overused words, you decide to write a function to count the number of times it uses one of the overused words.

```
1    def count_overused(answer, overused):
2        total = 0
3        words = answer.split()
4        for word in range(len(words)):
5            for overused_word in range(len(overused)):
6                if word == overused_word:
7                    total += 1
8        return total
```

For example, if `overused = ["delve", "tapestry", "landscape", "dive"]` and `example_answer = "Let's delve into how the tapestry of the landscape was influenced by the war of 1812".`

`count_overused(example_answer, overused)` should return 3.

a. Unfortunately, that is not what is returned if this code is run. Assuming that capitalization and spelling is correct, what number will be returned instead in the given example? Explain where that number is coming from.

b. What small change (affecting 2 lines of code or less) would you implement in this function so that it would have the correct behavior? Your answer should cite specific line numbers and show the code you would replace that line with.

---

20. For this problem, you are given a file named `list_of_points.txt`. The contents of this file are shown below:

list_of_points.txt

```
Point1: 1 2 3 4
Point2: 5 6 7 8
Point3: 0 0 0 0
```

Predict the output of the following lines of code if it is given this file as input.

```
my_file = open("list_of_points.txt")
for line in my_file:
    words = line.split()
    point = []
    for coordinate in words[1:]:
        point.append(int(coordinate))
    print(words[0], point)
my_file.close()
```

---

21. You run the code below.

```
numbers = [5, 8, 13, 20, 25]
result = []
for num in numbers:
    if num % 2 == 0:
        result.append(num * 2)
    elif num > 15 or num < 8:
        result.append(num)
    else:
        if num % 5 == 0:
            result.append(num + 5)
        else:
            result.append(num - 3)
```

```
print(result)
```

What will be output to your terminal?

---

22. For all sub-parts (a through c) of this question, assume that `words` is defined as follows:

```
     words = ["Python ", "in a ", "expanse ", "dance ", "is a ",
"digital "]
```

a) What is the output of the following code snippet?

```
poem = ""

for i in [0, 4, 3, 1, 5, 2]:
    poem += words[i]

poem += "."
print(poem)
```

b) What is the output of the following code snippet?

```
for i in [0, 4, 3, 1, 5, 2]:
   if i % 2 == 0:
       print(words[i])
   else:
      line = ""
      for j in range(i):
         line += words[i]
      print(line)
```

c) Given the following list `words2`, write a short program that will find all the strings that occur in both lists, `words` and `words2`, and print them out in a list. To receive full credit, you ***must*** use range()

The `words` list is the same as it was on the previous page:

```
     words = ["Python ", "in a ", "expanse ", "dance ", "is a ",
"digital "]
```

And `words2` is defined as:

```
     words2 = ["in a ", "digital ", "space ", "Python ", "has ", " a "
, " place"]
```

The output should be: `['Python ', 'in a ', 'digital ']`

---

23. What is the output of the following code?

```
foods = ['beef', 'burger', 'sushi', 'fries', 'carrot',
'pizza']
result = []
for i in range(0, 6, 2):
    result.append(foods[i][-1])
result = result + ['z', 'is']

if (len(foods) % 2) == 0:
    result.extend('awesome')
else:
    result.insert(0, 'cool')

print(result)
```

---

24. How many times will the **print** statement be executed in the code below?

```
for x in [2, 4, 6, 8]:
 for y in range(x):
  print str(x) + "!"
```

---

25. For each of the if statements below, write the output when x = 15, x = 45, and x = 60 in the table below:

```
a)
if x < 30:
 print "line 1"
elif x < 60:
 print "line 2"
else:
 print "line 3"

b)
if x < 30:
```

```
 print "line 1"
if x < 60:
 print "line 2"
else:
 print "line 3"
```

	X = 15	X = 45	X = 60
Code a)			
Code b)			

---

26. What output is produced after running the following piece of code?

```
a = 12
p = 6

def foo(a):
 a = cat(a + 1)
 print "In foo", a, "and", p

def bar(i):
 i = i + 2
 print "In bar", i

def cat(a):
 a = a + 10
 bar(a)
 print "In cat", a
 return a


print "foo(p)=", foo(p)
print "a=", a
print "p=", p
```

27. After this code executes, what is printed?

```
list_A = [17, 3]
list_A.append(5)
list_B = list_A
list_B.append(21)
list_C = list(list_A)
list_A.append(8)
list_C.append(4)

print list_A
print list_B
print list_C
```

28. Write the output of the code below in the box here:

```
sum = 0
for x in range(1, 5):
      for y in range(x):
            sum = sum + y
print('sum:', sum)
```

---

29. The following function is supposed to take a list, and shift each element to the left by one location. The first element in the list should be placed at the last position of the list. For example, the list: `[1, 5, 4, 6, 8, 7]` should become: `[5, 4, 6, 8, 7, 1]`

```
def shift_one(lst):

      # Assumes lst contains at least one element.
      for i in range(0, len(lst) - 1):
            lst[i] = lst[i + 1]
      return lst
```

However there is a bug in the code. You should:

a) Identify and describe in plain English the bug

b) Modify the code above such that it does what is expected.

---

30. What output is produced after running the following piece of code?

```
a = [3, 1, 5]
b = a
c = b[:]
d = list(a)

a.append(9)
b.append(2)
c.append(6)
d.append(5)

print a
print b
```

```
print c
print d
```

---

31. For each of the if statements below, write the output when x = 10, x = 35, and x = 70 in the table below:

```
a)
if x > 20:
     print "line 1"
elif x > 50:
     print "line 2"
if x < 30:
     print "line 3"

b)
if x < 50:
     print "line 1"
     if x < 50:
          print "line 2"
else:
     print "line 3"
```

	X = 10	X = 35	X = 70
Code a)			
Code b)			

---

32. Write the output of the code below in the box here:

```
sum = 0
for x in range(2, 6):
     for y in range(x, 0, -1):
          sum = sum + 1
```

```
print('sum:', sum)
```

---

33. For each of the following statements, show what is printed. Put ----- if nothing is printed.

```
def foo(a):
     if (len(a) % 2 == 0):
          return a
     else:
          return a + "foo"

def bar():
     print "bar"

def cat(c):
     bar()
     c = c + "cat"
     return c

a) print bar()

b) print cat("red")

c) print foo("green")

d) print foo(cat("blue"))
```

---

34. What output is produced after running the following piece of code?

```
A = [1, 2]
B = list(A)
C = A
D = A[:]

E = A.append(3)
B.append(A)
D.append(E)
B[1] = 7
```

```
print A
print B
print C
print D
print E
```

---

35. For each of the if statements below, write the output when x = 20, x = 40, and x = 100 in the table below. If there is no output then write “NO OUTPUT”.

```
a)
if x < 30:
     print "line 1"
if x <= 40:
     print "line 2"
elif x < 100:
     print "line 3"

b)
if x > 20:
     print "line 1"
else:
     if x < 50:
          print "line 2"
     print "line 3"
```

	x = 20	x = 40	x = 100
Code a)			
Code b)			

---

36. Write the output of the code below in the box here:

```
sum = 0
```

```
for x in range(6, 0, -2):
      for y in range(x):
            sum = sum + y
print('sum:', sum)
```

---

37. What output is produced after running the following piece of code?

```
A = [1, 3, 7]
B = A
C = A[:]

A.append(C[-1])
B[2] = 5
C[1:2] = [9, 10, 11]

print A
print B
print C
```

---

38. For each of the following statements, show what is printed. If nothing is printed then write “NO OUTPUT”.

```
x = -200
def fig(x):
      if x < 100:
            return "small"
      else:
            return "large"

def pear(x):
      print "pear:", x
      return x + 7

def apple(y):
      y = pear(y)
      print "apple:", y
```

```
a) print pear(10)
b) print apple(2)
c) print fig(120)
d) print fig(pear(2))
```

---

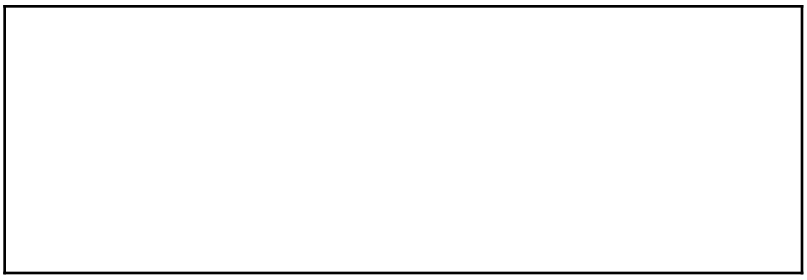
39. Given the following function, what will print out when the below code is run?
If nothing will be printed, write “Nothing” in the corresponding text box.

```
def mystery(n):
    if n < 0:
        print("Invalid input!")
    else:
        for i in range(1, n + 1):
            result = ""
            for j in range(i):
                result += "*"
            print(result)
```

(a) (1 point) mystery(-1)

(b) (1 point) mystery(0)

(c) (4 points) mystery(5)



---

40. Suppose we have the following list:

fruits = ["cherry", "strawberry", "elderberry", "raspberry"] (a) (2 points)

What is the printed output after running the following code?

```
ans = []
for i in range(len(fruits):
        if i % 2 == 0:
                ans.append(fruits[i])
        print(ans)
```

(b) (2 points) What is the printed output after running the following code?

```
ans = []
for i in range(3):
        ans.append(fruits[i][0])
        ans.extend(["160"])
        print(ans)
```

(c) (2 points) Write one line of code that modifies the original fruits list so that it contains the following:

fruits = ["cherry", "raspberry"]

Hint: list slicing will need to be used.

---

41. Each of the code snippets below has a different error. For each snippet, write one sentence naming and describing the error. Then, rewrite the code to correctly produce the indented output (indicated in the comment above the snippet).

```
A. # The expected output is: 60
height = [55, 65, 77, 56, 63, 74, 57, 60]
print(height[8])
```

```
B. # The expected output is:
# I love Python!
# I love Python!
# I love Python!
for i in range(3):
      print("I love Python!")
```

```
C. # The expected output is: I don't want to
      listen to music now. print('I don't want to
      listen to music now.')
```

---

42. For each of the following for loops, write in the box to the right how many times "**I love Python!**" is printed. Your answer should be a number.

```
for i in range(5):
      print("I love Python!")

for i in range(4, 9):
      print("I love Python!")

for i in range(11, 11):
      print("I love Python!")

for i in range(4, -9, -3):
      print("I love Python!")

for i in [1, 3, 5, 19393, -3]:
      print("I love Python!")

for i in [[1], [2, -1], [-3, 11, 64], [1, [32]], 5]:
      print("I love Python!")

for i in range(1, 19, 3):
      if i % 5 == 2 or i % 4 == 0:
            print("I love Python!")

numbers = [3, 2, 1, 5, 7, -4]

for i in numbers[2:5]:
      print("I love Python")
```

---

43. A Biology student has started learning Python to help with their research. They have written the function below.

```
def count_num_of_insects(animal_data):
    '''count_num_of_insects is designed to count the amount of
    insects in animal_data solely by counting how many have six (6)
    legs.
    Inputs: a list of integers where each number is an animal and how
    many legs it has. For example, for three animals where the first
    two have four legs and the third has 6, the input would be [4, 4,
    6].
    Outputs: an integer that shows the amount of insects in the
    Sample.
    '''
    insect_num = 0
    for animal in animal_data:
        if animal == 6:
            insect_num += 1
```

Later they call this function in their code with:

```
print(count_num_of_insects([4, 4, 6]))
```

The student has found that their code is not giving them the desired output.

A. Write one to two sentences on why the code is not producing the desired output:

B. What is the output of the above code?

C. What would they need to change in their function to give their desired output?

---

44. For each of the code snippets on the left, write the corresponding output on the right.

Code	Output

`total_sum = 0`
`for i in range(2, 8, 2):`
`total_sum = total_sum + i`
`print(total_sum)`	

```
for i in range(2):
for j in range(2):
print(i + j)
```

```
month = "January"
day = 10
output_string = ""

if day <= 10:
output_string += "Early "
if day >= 20:
output_string += "Late "
else:
output_string += "Mid "
output_string += month
print(output_string)
```

```
hours = 0
day = True
if hours < 12 and day:
hours += 12
elif hours == 12:
day = False
else:
hours -= 6
print(day)
print(hours)
```

---

45. Consider the following two functions:

```
def my_function(a, b):
result = a - b
return result
```

```
def loop(x, y):
for i in range(3):
if i % 2 == 0:
x = my_function(x, y)
else:
y = my_function(y, x)
return x + y
```

Fill in the tables below such that the code in the "Call to `loop()`" column produces the output in the "Output" column. The code in the "Call to `loop()`" column *must* use the loop function defined above.

For example:

***[Example]* Code**	***[Example]* Output**
`print("2 cubed is", 2 ** 3)`	`2 cubed is 8`

a) (3 points) Write the output produced by the code on the left:

Call to `loop()`	**Output**
`print(loop(4, -2))`	
`print(loop(-1, -2))`	

`x = 3`
`y = 2`
`result = loop(y, x)`
`print(result)`	

b) (3 points) Write a call to the function `loop()` that will produce the value on the right. (Hint: you may find it helpful to "unroll" the loop.)

Call to `loop()`	**Output**
	`8`
	`-8`

```
0
```

---

46. You are given a file named `start.txt` which contains the following text: `start.txt`

```
Knock knock
Owls say who?
Yes they do
```

a) (2 points) Write the contents of `output_a.txt` after running the following program (you may assume it is in the same directory as `start.txt`).

```
start_file = open('start.txt', 'r')

output_a = open('output_a.txt', 'w')

for line in start_file:
for word in line.split():
      output_a.write(word + '\n')
start_file.close()
output_a.close()
```

b) (3 points) Write the contents of `output_b.txt` after running the following program (you may assume it is in the same directory as `start.txt`).

```
start_file = open('start.txt', 'r')

output_b = open('output_b.txt', 'w')

for line in start_file:

output_b.write(line + '\n')

for line in start_file:
```

```
output_b.write(line + '\n')
start_file.close()
output_b.close()
```

---

47. After running this snippet of code, what will this function print?

```
def foo(n1, n2, n3, b, s, d):
     if n1 % 2 == 0 or not b:
          print("A")
     elif str(n2 * n3) == 21.0:
          print("B")
     else:
          print("C")
     if d["b"][2] + d[1] == "LA":
          print("D")
     if d["a"] == n1 - n2:
          print("E")
     else:
          print("F")

a = 11;
b = "hello";
c = False;
d = 10.5;
e = 2;
f = { "a": 1, 1: "a", "b": b }

foo(a, d, e, c, b, f)
foo(17, 21, 1, True, "hallo", {"a": 1, 1: "a", "b": "hello"}) foo(0,
       3, 7.0, False, "hallo",{"a": -3, 1: "a", "b": "hallo"})
```

---

48. For all sub-parts (a through c) of this question, assume that `words` is defined as follows:

```
     words = ["Python ", "in a ", "expanse ", "dance ", "is a ",
"digital "]
```

a) What is the output of the following code snippet?

```
poem = ""

for i in [0, 4, 3, 1, 5, 2]:
    poem += words[i]

poem += "."
print(poem)
```

b) What is the output of the following code snippet?

```
for i in [0, 4, 3, 1, 5, 2]:
   if i % 2 == 0:
       print(words[i])
   else:
      line = ""
      for j in range(i):
         line += words[i]
      print(line)
```

c) Given the following list `words2`, write a short program that will find all the strings that occur in both lists, `words` and `words2`, and print them out in a list. To receive full credit, you ***must*** use range()

The `words` list is the same as it was on the previous page:

```
words = ["Python ", "in a ", "expanse ", "dance ", "is a ", "digital "]
```

And `words2` is defined as:

```
words2 = ["in a ", "digital ", "space ", "Python ", "has ", " a " , " place"]
```

The output should be: `['Python ', 'in a ', 'digital ']`

---

49. What is the output of the following code?

```
foods = ['beef', 'burger', 'sushi', 'fries', 'carrot', 'pizza']
result = []
for i in range(0, 6, 2):
    result.append(foods[i][-1])
```

```
result = result + ['z', 'is']

if (len(foods) % 2) == 0:
    result.extend('awesome')
else:
    result.insert(0, 'cool')

print(result)
```

Code Writing

50. Doctors can use MRI (Magnetic Resonance Imaging) scans of parts of your body to diagnose certain diseases without having to perform surgery. Because our organs are 3-dimensional, these images are also 3D (unlike the 2D images that you were working with in Homework 3). If we wanted to represent this in Python, we would have to use 3D *voxels* rather than 2D pixels as shown below.

An example 2 x 2 pixel_grid:

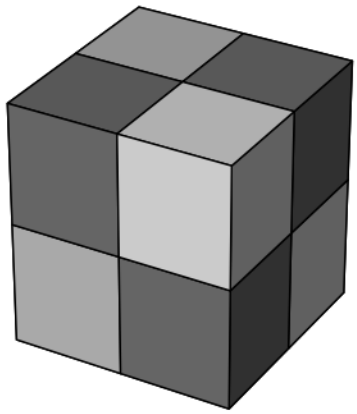
```
pixel_grid = [[1, 15],
              [20, 5]]
```

Rendered as a grid like so:

1	15
20	5

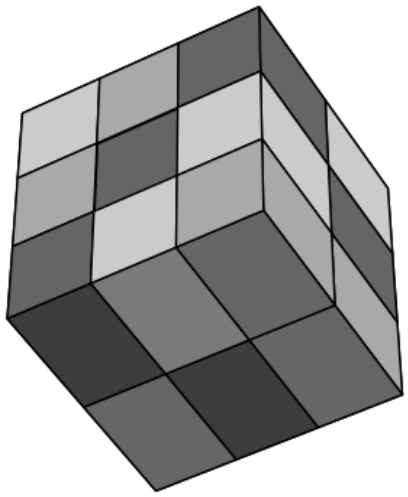
An example 2 x 2 x 2 voxel_grid:

```
voxel_grid = [
    [[3, 2],
     [2, 3]],

    [[2, 4],
     [4, 2]]
]
```



You can view these voxel grids as a series of 2D grids stacked together. For example, below is a 3 x 3 x 2 voxel grid and its rendering. (Each value in the grid for these renderings represents the color of an individual block, just as it might for a pixel in the 2D grids.)

```
voxel_grid = [
    [[3, 4, 2],
     [4, 2, 3],
     [2, 3, 4]],

    [[3, 2, 4],
     [2, 3, 2],
     [3, 4, 3]],
]
```



a) Implement a function called `remove_noise` which when given a `voxel_grid` returns a *new* voxel_grid that has 0s in place of all voxels with values < 10, with all other voxels being the same as the original. Altering the parameter `voxel_grid` will not give you full credit for this question.

```
def remove_noise(voxel_grid):
     # Write code here
```

---

51. You are given a list of lists of integers called `temperatures`, where each list contains all the recorded temperatures of a city. You are given another list called `cities` that contains the names of all of the cities that each list in `temperatures` corresponds to. For example, index 0 of `temperatures` contains the recorded temperatures for the city in index 0 of `cities`, and so on.

Write a function temperature_averaging that creates a dictionary of key-value pairs where the key is the name of the city and the value is the average recorded temperature for that city.

For example, given

```
temperatures = [[78, 92, 24], [34, 12], [40, 4, 2, 67, 8]]
cities = [“Paris”, “Tokyo”, “Seattle”]
```

Your code should return the following dictionary:

```
{“Paris” : 64.6666667, “Tokyo”: 23, “Seattle” :  38.6}
```

You should not use the built-in python function `sum`. You may assume that the lists `temperatures` and `cities` are already defined. Write your code in the provided function header below. Do not do any rounding.

```
def temperature_averaging(temperatures, cities):
```

---

52. You are given a list of barcodes. An example could be:

```
example_barcodes =          [[1, 1, 0, 0, 0, 1, 1],
                            [1, 0, 1, 0, 0, 1, 0],
                            [0, 1, 0, 1, 1, 0, 1],
                            [1, 1, 1, 0, 1, 0, 0],
                            [0, 0, 1, 1, 0, 1, 1]]
```

This barcode list is subject to change, so all code should be written without hard coding the length of `example_barcodes`. All barcodes are the same length and consist of 0s and 1s. For problems a and b, there is no need to write a function.

a. Write a code snippet to count the number of 1s in each barcode and store the counts in a list. In the example above, the expected output would be `[4, 3, 4, 4, 4]`.

Write a code snippet to sum the values at different integers in the barcodes and store them in a list. The goal is to sum each column. In the example above, the expected output would be `[3, 3, 3, 2, 2, 3, 3]`.

---

53. Given a list of lists of integers, where each list contains all the exam scores of a class, write code that writes to a file "max_scores.txt" that contains a line **Class <class number>'s maximum score is <max_score>** for each class. You may ***not*** use the `max` built-in python function. You may assume that the `class_scores` variable is already defined.

For example, given

```
class_scores = [[66, 78, 43], [33, 12, 99], [67, 87, 2]]
```

The file `max_scores.txt` should contain the following content after running your code:

```
Class 1's maximum score is 78.
Class 2's maximum score is 99.
Class 3's maximum score is 87.
```

---

54. A vet clinic wants you to create a function `average_weight()` that will find the average weight of their dogs in order to track the changes in canine obesity overtime. It will take in a list of dictionaries (`patient_data`) and return a float. You can assume that this is the only species that we care about.

For example if given the list:

```
[{'Patient': 'Lucky', 'Height': 23, 'Weight': 30, 'Species': 'Dog'},
{'Patient': 'Remi', 'Height': 30, 'Weight': 40, 'Species': 'Dog'},
{'Patient': 'Licorice', 'Height': 10, 'Weight': 15, 'Species':
'Cat'},
{'Patient': 'Shadow', 'Height': 15, 'Weight': 25, 'Species': 'Cat'}]
```

The function would return:
`35.0`

---

55. Rewrite the following nested if statements as a single if statement (may contain elif and else)?

```
if val > 120:

     print "Green!"

else:

     if val > 80:

          print "Blue!"

     else:

          if val < 32:

               print "Red!"

          else:
               print "Orange!"
```

---

56. Write a function `max_in_col(pixel_grid)` that, given a list of lists of integers (a `pixel_grid` as in HW3), creates a list of integers that includes the maximum value found in

each column of `pixel_grid`. You can assume that `pixel_grid` will always contain at least one row and one column and that the values in `pixel_grid` will be between 0 and 255 and that each row will contain the same number of columns.

Here are a few examples. After the following code is executed:

```
col_list = max_in_col(pixel_grid)
```

If `pixel_grid` contains: `col_list` will be:

```
[ [ 4, 2, 3], [ 16, 200, 12 ]
 [16, 5, 0],
 [ 3, 200, 6],
 [ 0, 10, 12] ]
```

```
[ [ 4 ], [ 16 ]
 [ 16 ],
 [ 3 ],
 [ 0 ] ]
```

```
[ [ 4, 2, 3] ] [ 4, 2, 3 ]
```

```
[ [6] ] [ 6 ]
```

a) Write TWO assert statements that can be used to test if your function is correct.

b) **Fill in your code on the next page**

```
def max_in_col(pixel_grid):
 """
 Given a list of lists of integers in the range 0 to 255,  return a
 list containing the maximum integer in each column  of the grid.
 You may assume that pixel_grid will be a list containing at  least one
 list, containing at least one value.  """
```

```
# Your code starts here
```

---

57. Write a function called `create_recip_dict` that takes no arguments and returns a dictionary that maps the integers 1 to 100 to their reciprocal as a **string**. For example the resulting commands in the interpreter should produce the output as shown:

```
>>> recip_dict = create_recip_dict()
>>> recip_dict[0]
```

(Error)

```
>>> recip_dict[1]
'1/1'
>>> recip_dict[2]
'1/2'
>>> recip_dict[100]
'1/100'
>>> recip_dict[101]
```

(Error)

MY ANSWER:

```
def create_recip_dict ():

 # Your code here
```

---

58. Write a function called `find_value` that takes a **list of lists** and a **value** as arguments and returns a **list** giving the indices of the (sub)list and position in that list where the first instance of the value was found. If the value does not exist in any of the (sub)lists, the function returns [-1, -1]. For example, given:

```
my_list = [ [2, 5], [1, 3, 4], [4], [7, 3] ]
```

the call: `find_value(my_list, 4)` would return: `[1, 2]` and

the call: `find_value(my_list, 7)` would return: `[3, 0]` and

the call: `find_value(my_list, 8)` would return: `[-1, -1]`

```
def find_value(lst, val):

 # Write your code here
```

---

59. Write a function `get_youngest_person` that takes a list of dictionaries as arguments and returns the name of the youngest person in the list. The list of dictionaries will have the following format:

```
people= [
{"name": "Alice", "age": 20},
{"name": "Bob", "age": 9},
{"name": "Dan", "age": 56}
]
```

For example, `get_youngest_person(people)` should return "Bob". If there is more than one person with the smallest age, return the name of the person who occurs first in the list. You may assume the list contains at least one person and that no one is less than 1 year old.

```
def get_youngest_person(people):
 # Write your code here
```

---

60. Write a function called `transpose` that takes a `pixel_grid` as described in Homework 3 as an argument and returns the transpose of that `pixel_grid`. This is identical to how we w ould transpose a matrix: swap the rows and columns. For example, if we had:

```
grid1 = [ [1, 2, 3],
          [4, 5, 6] ]       # a grid with 2 rows and 3 columns

grid2 = [ [1, 2, 3, 4] ]    # a grid with 1 row and 4 columns

grid3 = [ [1] ]                  # a grid with 1 row and 1 column
```

The call: `transpose(grid1)` would return:

```
[ [1, 4],
  [2, 5],
  [3, 6] ]
```

The call: `transpose(grid2)` would return:

```
[ [1],
  [2],
  [3],
  [4] ]
```

The call: `transpose(grid3)` would return: `[ [1] ]`

You may assume that the provided `pixel_grid` contains at least one row and one column.

```
def transpose(pixel_grid):   # Write your code here
```

---

61. You want to start a small flower garden at your house but don’t know the right time to plant your seeds.

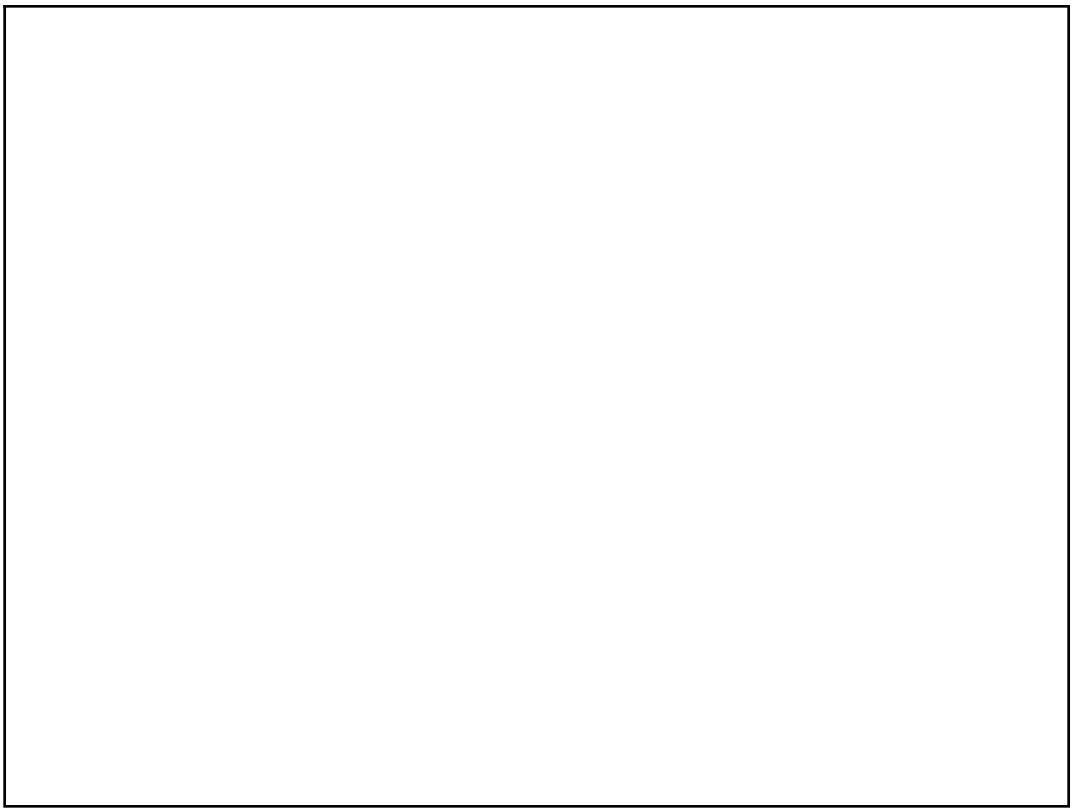
Write a function plant_time() that takes in two parameters:

- season - string which represents current season (can be only one of the values "spring", "summer", "autumn", "winter")
- temp - integer which represents temperature in Fahr

enheit

Behavior:

- If the season is "spring" or "autumn" you should print "You can plant" regardless of the temperature.
- If the season is "summer" and the temperature is between 50 (inclusive) and 80 (inclusive), you should print "You can plant", otherwise print "Do not plant".
- If the season is "winter" you should print "Do not plant".



---

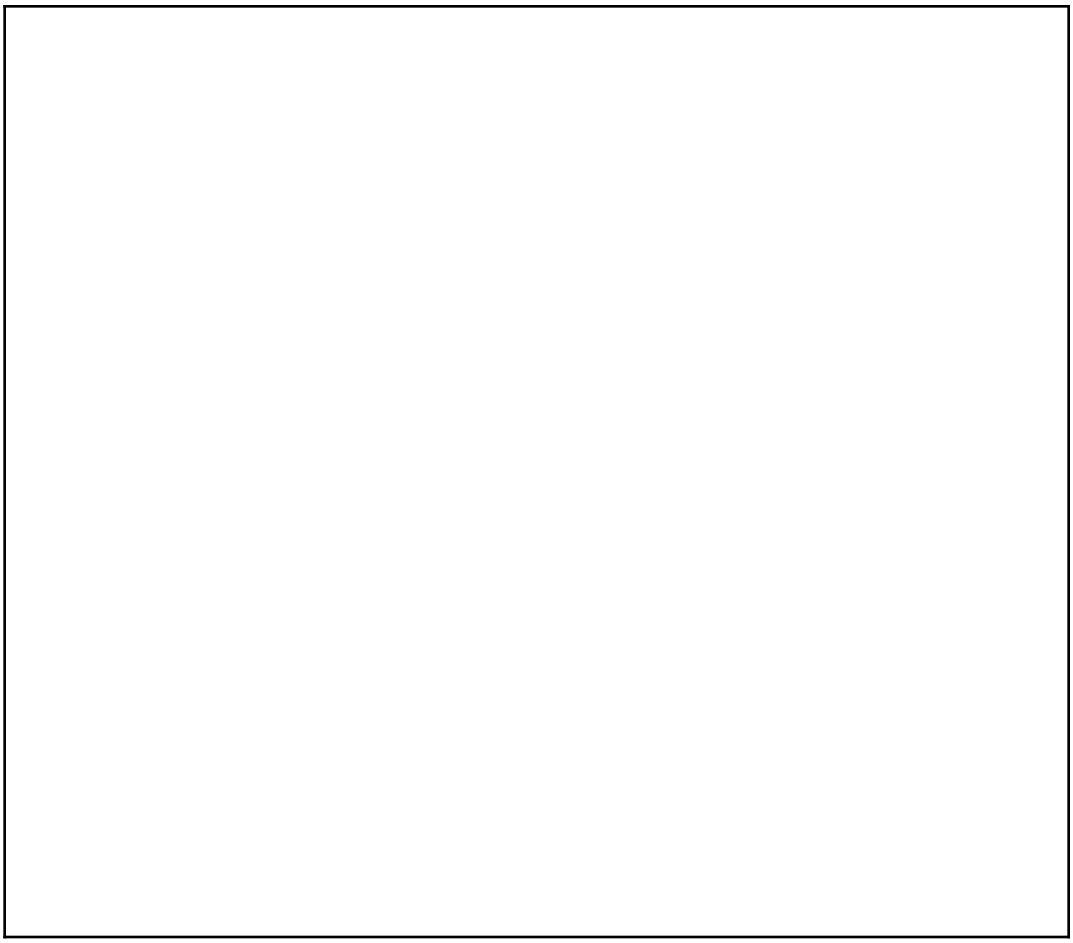
62. Given a list of strings that contains the names of TV shows tv_shows and a list of integers that contains the respective number of episodes per show episodes, write to a file "shows.txt" that contains a line <show_name> has <episode_number> episodes for each show.

For example, given:

tv_shows = ["Stranger Things", "Summer Strike", "Alice in Borderland"] episodes = [34, 12, 16]

The file shows.txt will contain the content:

Stranger Things has 34 episodes
Summer Strike has 12 episodes
Alice in Borderland has 16 episodes



---

63. Given a file named **`cool_story.txt`** (you should assume this path points to a valid file location and that the file has content), the code below aims to count the number of the common words "a", "the", and "and". Write the body of the for loop below such that **`common_words[0]`** contains the number of occurrences of "a", **`common_words[1]`** contains the number of occurrences of "the", and **`common_words[2]`** contains the number of occurrences of "and".

```
file = open("cool_story.txt", "r")
common_words = [0, 0, 0]

for line in file:
```

file.close()

---

64. Grace wants to set her alarm clock for the week. To do this, she wants you to write a function named **set_alarm** that returns the time (int) she should set her alarm clock for.

The function takes in the **day** (string) of the week and whether she is on vacation or not, **is_vacation** (bool, as in True or False). You may safely assume that **day** is *always* a valid day of the week (**"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**) and that **is_vacation** is *always* a boolean.

If it is a weekday (**"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, or **"Friday"**), she wants to set her alarm clock for **8**. On weekends she wants to set her alarm clock to **10**. But, if she is on vacation, she always wants her alarm clock to be set to **11**, regardless of the day of the week. Implement the function that returns the correct time.

Example calls and return values:

- **set_alarm("Monday", False)** should return **8**
- **set_alarm("Sunday", False)** should return **10**
- **set_alarm("Wednesday", True)** should return **11**

```
def set_alarm(day, is_vacation):
# Write your code below this line.
```

---

65. The highway patrol police has collected some data on traffic speed and needs help on analyzing it. Write a function **average_speed_over_limits(speed_data, limits)** that takes in a list of vehicle speeds (in miles/hr) and a list of speed limits (in miles/hr), then returns the average speed for vehicles that have exceeded those speed limits.

For example, **average_speed_over_limits([60, 80, 53, 68], [60, 20])** should return the list **[74, 65.25]** because

(1) the average speed of the two vehicles (at 80 miles/hr and 68 miles/hr) that exceeded 60 miles/hr is 74 miles/hr,
(2) all four vehicles have a speed exceeding 20 miles/hr and the average is 65.25 miles/hr.

If there is no vehicle data in **speed_data** or if there are no vehicles that exceed the speed limit

in the data, it should set the average speed to 0 for the corresponding limit. Your function should return a list that has an average speed corresponding to each speed limit. You can assume that the speed limits and the vehicle speeds will all be positive integers.

```
def average_speed_over_limits(speed_data, limits):
# Write your code below this line.
```

---

66. Instead of analyzing DNA (A, T, C, G) sequences like in HW2, now you are given an RNA (A, U, C, G) sequence as an input string.

Each three nucleotides in the sequence translates to an amino acid. Amino acids are the building blocks to make proteins! Use a **for loop**, **range**, and **string slicing** to "translate" the RNA sequence to the amino acids. Every three nucleotides corresponds to a single amino acid.

Specifically, your objective is to look for the following amino acids:

RNA to target Amino Acid Mapping:
"AUG" --> "Methionine"
"UGC" --> "Cysteine"
"UCU" --> "Serine"

For example:

given
`rna = "AUGCUCAUG"`
your code should print out:
`['Methionine', 'Methionine']`

given:
`rna = "ACCUUUAUGAUUUGCUACCAAUCUUUUUGCCGAUCUGCAUCUUUUGGG"`
your code should print out:
`['Methionine', 'Cysteine', 'Serine', 'Cysteine', 'Serine', 'Serine']`

---

67. Write a function `tricky_function(arr)` that modifies a given `list` in the following way: at index `i`, if the value at index `i` is divisible by 3, then the function should multiply the value at index `i` by its index; otherwise leave the value as it is.

Examples:
`tricky_function ([0, 1, 2, 3, 4, 5, 6, 7, 8]) = [0, 1, 2, 9, 4, 5, 36,`

```
       7, 8]
tricky_function ([2, 1, 21, 7, 4, 5, 4, 7, 18]) = [2, 1, 42, 21, 4, 5,
       4, 7, 144]
tricky_function ([2, 3, 2]) = [2, 3, 2]
tricky_function ([1, 1, 1, 1]) = [1, 1, 1, 1]
```

---

68. Write a function `join` that, given two lists of characters `c1` and `c2`, returns the number of pairs in `c1` and `c2` whose values are equal.

Examples:

```
join ([a,b,c,e], [a,a,e,b,c,c,d]) = 6
*[[a,a], [a,a], [b,b], [c,c], [c,c,], [e,e,]]

join ([a,a], [a,a]) = 4
*[[a,a], [a,a], [a,a], [a,a]]

join ([a,b,c], [a,b,c]) = 3
*[[a,a], [b,b], [c,c]]

join ([x,y,z], [a,a,e,b,c,c,d]) = 0
*[]
```

---

69. Write a function `max_consecutive_sum` that, given a list of integers, returns the absolute value of the largest sum between two consecutive numbers in the list. Return -1 if the length of the list is less than or equal to 1.

Examples:

```
max_consecutive_sum ([36, 45, 7]) = 81
max_consecutive_sum ([3]) = -1
max_consecutive_sum ([2, 3, 2]) = 5
max_consecutive_sum ([1, 1, 1, 1]) = 2
```