1. What will be the result of evaluating the following expression:

```
[ i + j
    for i in [x ** 2 for x in range(-2,2)]
      for j in {x ** 2 for x in range(-2,2)}
]
```

Answer:
[4, 5, 8, 1, 2, 5, 0, 1, 4, 1, 2, 5]

Feedback:
The above list comprehension expression will evaluate to a list, where every element in the list i + j, such that i is an element in the list [4,1,0,1,4] and j is an element in the set {0,1,4}. Note a set has only unique elements.

2. Consider the following dictionary that records a fruit name and its color: fruit_color = {'apple' : 'red', 'orange' : 'orange', 'strawberry' : 'red', 'banana' : 'yellow'}

We need to create a list of fruits that have the red color.

Answer:
[f for f in fruit_color if fruit_color[f] == 'red']
[k for k,v in fruit_color.items() if v == 'red']

Feedback:
There are two ways of doing this – (1) a list comprehension, where a fruit f is included the list if the fruit f is mapped to the color 'red' in the dictionalry fruit_color (i.e., fruit_color[f] = 'red'), and (2) a list comprehension, where a fruit name k has the value v in the dictionary fruit_color and the value v = 'red'. Note the different ways in which the dictionary was iterated, with and without the use of *items()*. The use of *items()* allows us to iterate over key-value pairs as opposed to just keys in a dictionary.

3. Consider a matrix with different column sizes. For example, in the matrix m = [[1,2,3],[4,5],[7,8,9],[10,11,12,13]], the first row has 3 columns, but the second row has 2 columns. If we multiply a constant number, K, with such a matrix, then we get a new matrix, where every element is multiplied by the number K. For example, if 2 is multiplied with matrix m, then we will get a new matrix [[2, 4, 6], [8, 10], [14, 16, 18], [20, 22, 24, 26]]. Notice, every element in m was multiplied by 2.

Assuming that you are given a matrix, m, and a constant K, which of the following

options will create a new matrix such that every element in the new matrix is a product of K and the corresponding element in m?

Answer:
[[K*x for x in row] for row in m]

Feedback:
A matrix with different column sizes is can be expressed in Python as a list of lists, where every list has a different length. To multiply every number in the list with a constant number k, we could write nested loops to iterate through the list of lists. However, a simpler and more elegant approach is to create a nested list comprehension (LC) expression, as shown in the solution. The nested LC shown above will be interpreted as follows:

a. *row* is a list in the list of lists
b. for each element x in the list, modify x as K*x

After evaluating this LC expression, we will have a new list of lists or matrix with every number in the matrix multiplied with a constant number K.

4. Consider a CSV file where every line is a comma-separated collection of values. The first line is the header and the subsequent lines indicate data lines. The last column of a data line must be a number. Following is an example of the contents in a file that fits the above constraints:

Name, Course, Score
Jonita, CSE337, 92
Karev, ISE337, 86
Lane, CSE337, 78
Amu, ISE337, 95

We need to compute the sum of the last column in all the lines, except the first line. Which of the following functions would give us the correct result?

Answer:
```
def getSum(fileName):
   sum = 0
  with open(fileName) as f:
      f.readline()
     for lines in f:
         n = int(lines.split(',')[-1])
         sum = sum + n
  return sum
```
Feedback:
There a number of ways of doing this. The solution above shows one way.We first open the file using the *open()* method and the *with* clause. Writing file I/O statements inside

the *with* clause will automatically close the file after we are done, saving memory and resources. Once the file is open, we get a pointer to the first line. Since file's data begins from the second line, the first line being the header, we use the *readline()* method to skip the header and move the pointer to the next line. We begin iterating over each line in the file beginning at second line. In each iteration, we extract the *Score* value in the file using the *split()* and indexing operation on the string *line*. We then add to *sum.* After all lines have been process, the file is closed and we return *sum.*

5. Consider the following snippet:

```
class A:

 def __init__(self):
 self.n = n

 def addTwice(self):
 self.n = self.n + self.n
 return self.n

class B:

 def __init__(self, n=0):
 self.n = n

 def addTwice(self):
 self.n = self.n + self.n
 return self.n

n = 10
a = A()
b = B()
c = B(n)
print(a.addTwice(), ' ', b.addTwice(), ' ', c.addTwice())
```

What will be the output?

Answer:
20 0 20

Feedback:
When an instance of class A is instantiated, the __init__() method of class A is invoked. In this method, the instance variable n is assigned to a variable n. As per the rules of scoping, Python looks for the value of n and finds that n = 10 in the *enclosed scope.* Hence, when *addTwice()* is invoked using an instance of class A, it uses *self.n = 10* and returns 20. However, an instance of class B behaves very differently. When an instance

of B is created, the __init__() method assigns n to the instance variable *self.n*. the value of the variable n is either the default value 0 or the value provided as argument when creating an instance of B.

6. Consider the following snippet:

```
personal_info = [['Jilly', [28, 150, 5.7]],
['Rony', [30, 200, 5.9]],
['Dario', [18, 157, 5.4]],
['Kim', [19, 125, 6.0]]]


personal_info_copy = personal_info.copy()
personal_info_copy[0] = ['Jilly', [25, 125, 5.7]]

print(personal_info[0])
print(personal_info_copy[0])
```

What will be the output?

Answer:
['Jilly', [28, 150, 5.7]
['Jilly', [25, 125, 5.7]]

Feedback:
In this question, we are doing a shallow copy of the list *personal_info*. In shallow copy, any changes to the first-level elements of a list are not reflected in the original and vice versa.

7. Consider the following snippet:

```
personal_info = [['Jilly', [28, 150, 5.7]],
['Rony', [30, 200, 5.9]],
['Dario', [18, 157, 5.4]],
['Kim', [19, 125, 6.0]]]


personal_info_copy = personal_info.copy()
personal_info_copy[1][1][0] = 35

print(personal_info[1])
print(personal_info_copy[1])
```

What will be the output?

Answer:
['Rony', [35, 200, 5.9]]
['Rony', [35, 200, 5.9]]

Feedback:
Unlike in the previous case, we are changing a nested element in the list. Since shallow copy makes a copy of only the first-level elements, modifying nested elements modifies both the copy and the original.

8. Consider the following code snippet:

```
def foo(n):

        def lessThanK():
print(n, 'is less than', k)
        def equalsK(n):
print(n, 'is equal to', k)
        def gtThanK():
print(n, 'is greater than', k)

if n < k:
        lessThanK()
elif n == k:
equalsK(n+1)
else:
gtThanK()

k = int(input('Enter an integer: '))
foo(10)
```

If the program is executed 3 times with user input 10, 12, and nine, respectively, then which of the following options is the correct output?

Answer:
11 is equal to 10
10 is less than 12
ValueError

Feedback:
As per the rules of scoping, for input 10, in the function *equalsK(),* n = 11 and k = 10. For input 12, *lessThanK(),* n = 10 and k = 12. For input nine, we will get a ValueError when n = 10 is compared with k = 'nine'.

9. Consider the following code snippet:

```
class Person:
 skills = []
 def __init__(self, name):
 self.name = name

 def add_skill(self, skill):
 self.skills.append(skill)

class PersonOne:
 skills = 'cooking'

 def __init__(self, name):
 self.name = name

 def change_skill(self, skill):
 self.skills = skill

p1 = Person('Frida')
p2 = Person('Kahlo')
p1.add_skill('painting')
p2.add_skill('programming')

p3 = PersonOne('Jim')
p3.change_skill('eating')
p4 = PersonOne('Pal')

print(p1.skills, ',', p2.skills, ',', Person.skills)
print(p3.skills, ',', p4.skills, ',', PersonOne.skills)
```

Answer:
[‘paintin’, ‘programming’], [‘painting’, ‘programming’], [‘painting’, ‘programming’], eating, cooking, cooking

Feedback:
In both classes, *skills* is a class variable. In class *Person*, *skills* is mutable, but in class *PersonOne*, skills are immutable. When a class variable is mutable, it is shared by all instances of the class. When a class variable is immutable, then every instance will make its own copy.