CS1026 Review/Practice Questions

1. Consider the following Python program and then answer the questions following.

```
1.        MAX_STARS = 40
2.
3.        # Read the data values from the user.
4.        values = []
5.        inputStr = input("Enter a value (blank to quit): ")
6.        while inputStr != "" :
7.           values.append(float(inputStr))
8.           inputStr = input("Enter a value (blank to quit): ")
9.
10.       # Identify the largest value.
11.       largest = max(values)
12.       # Display the bars.
13.       for i in range(0, len(values)) :
14.           print("*" * round(values[i] / largest * MAX_STARS))
```

1.a. What is a named constant in the program? **MAX_STARS**

1.b. What type of value is the user expected to enter in line 5? **string**

1.c. What kind of data structure is being used in the program? **list**

1.d. What variable references that data structure? **values**

2. The following function computes the number of digits in an integer. For example, if the parameter `n` has a value of 113, then the function should return 3. Similarly, if `n` has a value of 4, then it should return 1. The function has 4 logic mistakes; find them and correct them.

```
def digits(n) :
   if abs(n) < 10 :
      return 0                          # return 1
   else :
       num = 0
       while abs(n) > 10:               # >= 10
           num = num + 1
           n = n % 10                   # n = n // 10
       return num                       # return num+1
```

3. The following main program makes use of the function `digits` defined above. Assume that the function digits has been modified to work correctly. The main program loops repeatedly asking the user to enter an integer

or the letter "Q". Anything entered other than a "Q" or an integer is to cause an exception to be raised – a `ValueError` exception. Fill the in blanks with correct code to complete the main program.

```
def main() :
    sg = ""
    while  sg != "Q" :
        try :
            sg = input("Enter an integer (to terminate, enter Q: ")
            if sg != "Q":
                num = int(sg)
                ndig = digits(num)
                print("The number of digits in "+str(num)+" is "+str(ndig))
        except ValueError :
            print("Input error - retry.")
```

4. The following program computes the average value in each row and each column and then prints the table. It needs several functions defined.

```
def main():
    prices = [[ 1.0, 3.50, 7.50 ],
              [ 10.0, 30.50, 70.50 ],
              [ 100.0, 300.50, 700.50 ],
              [ 1000.0, 3000.50, 7000.50 ]]
    print(rowavg(prices))
    print(colavg(prices))
    printTable(prices)

def listavg(lst):

    . . . . . .

    return xavg/len(lst)


def rowavg(t):
    avg = []

    . . . . . .

    return avg


def colavg(t):
    avg = []
    ncols = len(t[0])

    . . . . . .
    return avg

def printTable(t):
    ravg = rowavg(t)
    cavg = colavg(t)
    for i in range(0,len(t)):
        for j in range(0,len(t[i])):
```

```
            print("%8.2f" % t[i][j],end="")
        print("%8.2f" % ravg[i])
    for i in range(0,len(cavg)):
        print("%8.2f" % cavg[i],end="")
    print()

main()
```

Complete the function definitions for the functions `lstavg, rowavg, colavg`.

```
def listavg(lst):
    xavg = sum(lst)
    return xavg/len(lst)


def rowavg(t):
    avg = []
    for i in range(0,len(t)):
        avg.append(listavg(t[i]))
    return avg


def colavg(t):
    avg = []
    ncols = len(t[0])
    for i in range(0,ncols):
        xavg = 0
        for j in range(0,len(t)):
            xavg += t[j][i]
        avg.append(xavg/len(t))
    return avg
```

5. Consider the following program:

```
def main() :
    a = 5
    print(doubleIt(a))
def doubleIt(x) :
    return x * 2
main()
```

What output is generated when this program is run? **10**

6. The following code segment is supposed to count the number of times that the number 5 appears in the list values ; assume that values has been assigned.

```
counter = 0
for element in values:
    if (element == 5) :
        counter = counter + 1
```

What line of code should be placed in the blank in order to achieve this goal?

7. Given the following code snippet, what is final version of the list names?

```
names = []
names.append("Amy")
names.append("Bob")
names.append("Peg")
names[1] = "Ruth"
names.pop(2)
names[0] = "Cy"
names.insert(0, "Ravi")
```

['Ravi', 'Cy', 'Ruth']

8. Consider the following Python classes and then answer the questions following.

```
#--------------------
## Represent an employee with a name and salary.
class Employee :

   def __init__(self, name, salary) :
      self._name = name
      self._salary = salary

   def __repr__(self) :
      return self._name + " has a salary of %.2f" % self._salary

#--------------------
## Represent a manager with a department.
class Manager(Employee) :

   def __init__(self, name, salary, department) :
      super().__init__(name, salary)
      self._department = department

   def __repr__(self) :
      return self._name + " has a salary of %.2f" % self._salary + \
             " and manages the " + self._department + " department"

#--------------------
## Represent an executive.
class Executive(Manager) :

   def __repr__(self) :
      return self._name + " has a salary of %.2f" % self._salary + \
             " and is the executive for the " + self._department + "
department"
```

a) What is the constructor for the class `Manager`?

```
def __init__(self, name, salary, department) :
```

b) What class is the class `Manager` a subclass of? **Employee**

c) What class is the class `Executive` a subclass of? **Manager**

d) If you create an object of the class `Manager`, what instance variables would it have?

_name, _salary, _department

e) If you create an object of the class `Executive`, what instance variables would it have?

`_name, _salary, _department`

f) What does `super()` do in the class `Manager`?

Invokes the constructor in the class Employee

g) Write a method for the class `Manager`, that will get the department name of a manager.

```
def getDept(self) :
    return self._department
```

h) Consider the following lines of code.

```
emp = Employee("John Smith", 45000.00)

print(man.getDepartment())
```

What happens when you run the code?

The first line will create an object of the Employee class – only.
The second line will cause an error.

i) Write a main method that will make use of these classes. It should a) create a Manager object for "Mickey Mouse", who manages the "Entertainment" department and has a salary of $83,000, b) create and object for "Walt Disney" who is the executive for the "Entertainment" department and has a salary of $195,000, c) print out each object.

```
aMan = Manager("Mickey Mouse", 83000.00, "Entertainment")
theExec = Executive("Walt Disney", 195000.00, "Entertainment")
print(aMan)
print(theExec)
```

j) Define a class ExecutiveAssistant for employees that work as the assistant to an executive in the company. The class definition should include the name of the executive that they work for.

```
class ExecutiveAssistant :

    def __init__(self, name, salary, department, exec) :
        self._name = name
        self._salary = salary
        self._department = department
        self._execBoss = exec
```

9. Consider the class Employee:

```
class Employee :
    def __init__(self, firstName, lastName, employeeId) :
        self._name = lastName + "," + firstName
        self._employeeId = employeeId
```

If an object is constructed as: sam = Employee("Sam", "Fisher", 54321)
What is the content of the instance variable `_name`? **Fisher,Sam**

10.Some Logic Problems

a. The following program is to read a text file and then organize the words in the text by their starting letter. This is then used to form a "dictionary" of words in the text. The program has 3 logic errors. Find and correct them. Assume that the file `inf` has been input and opened correctly.

```
words = {}
for ch in "abcdefghijklmnopqrstuvwxyz" :
   words[ch] = set()

# Read all of the lines from the file and add them to the dictionary.
for line in inf:
   parts = line.split(",")        -- should be " " - split on blank
   for word in parts :
      word = word.upper()         -- should be lower - ch above is lower
      for ch in words :
         if ch == word[0] :
            words[ch].add(ch)     -- should be add(word)


# Display the contents of the dictionary.
for ch in sorted(words) :
   print("Words beginning with %s:" % ch.upper())
   for word in sorted(words[ch]) :
      print(" ", word)
```

b. The following function builds a dictionary of the frequency of letters in a string. Assume that the string consists only of letters; i.e. all digits and punctuation have been removed. The function has 2 errors; find and correct them.

```
def charCounts(s) :
   # Count the number of occurrences of each character.
   counts = {}
   for ch in s :
      if ch not in counts :
         counts[ch] = 0                -- should be 1
      else :
         counts[ch] = counts[ch] + 1
   return s                            -- should be return counts
```

c. The following code is supposed to collect only letters (i.e. 'A' to 'Z' and 'a' to 'z' inclusively) ignoring all others. It is supposed to count the number of letters, the number of upper case letters and number of lower case letters. It contains 4 errors. Find and correct them

```
countU = 0
countL = 0
nL = 0

aset = set()
for ch in strng :
      if ch > "A" or ch < "Z" :        -- should be and
        aset.add(ch)
        countU = countL + 1            -- should be countU + 1
        nL = nL + 1
      elif ch > "a" or ch < "z" :      -- should be and
         aset.add(ch)
         countL = countU + 1           -- should be countL + 1
         nL = nL + 1
print(nL, countU, countL)
```

11. Some programming questions

1. Write a program that reads in two files and prints out, in sorted order, all words that are common in both of them. The names of the files are `input1.txt` and `input2.txt`. Assume that all words contain only lowercase and that the format of each file is comma separated words.
   e.g. `word,word2,word3,word5,word7`

```
def fileWords() :
   filename = input("Enter the name of a file: ")
   inf = open(filename, "r")
   words = set()
   for line in inf :
      parts = line.split(",")
      for word in parts :
         words.add(word)
   return words
words1 = fileWords()
words2 = fileWords()
common = words1.intersection(words2)
print("The words that are common to both files.")
```

```
for word in sorted(common) :
   print(" ", word)
```

2. Write a function `def merge(a, b)` that merges two lists, alternating elements from both lists. If one list is shorter than the other, then alternate as long as you can and then append the remaining elements from the longer list. For example,

   if `a` is 1 4 9 16 and `b` is 9 7 4 9 11

   then `merge` returns a new list containing the values

   1 9 4 7 9 4 16 9 11

```
def main() :
   # Set up sample lists.
   a = [1, 4, 9, 16]
   b = [9, 7, 4, 9, 11]

   # Demonstrate that merge works correctly.
   print("List a is", a)
   print("List b is", b)
   result = merge(a, b)
   print("The merged list is", result)

def merge(a, b) :
   result = []

   shorter = min(len(a), len(b))
   for i in range(0, shorter) :
      result.append(a[i])
      result.append(b[i])

   for i in range(shorter, len(a)) :
      result.append(a[i])
   for i in range(shorter, len(b)) :
      result.append(b[i])

   return result
# Call the main function.
main()
```