End Sem Exam

Introduction to Programming (CSE 101)- 2024

Duration: 60 mts **Total: 30 Marks**

Q1. **(1+1 marks)** Consider the following code:

```
1. def f(s):
2.     chars = {}
3.     for char in s:
4.         if char not in chars:
5.             chars[char] = 1
6.         else:
7.             chars[char] += 1
8.     d = []
9.     for char, count in chars.items():
10.           if count > 1:
11.               d.append(char)
12.    return d
```

a. What is the output of `print(f("CSE101Examisgoingon")) ?` ['E', '1', 'i', 'g', 'o', 'n'] (1 mark - for correct answer 0.5 - if there's just one omission or commission error in the answer, but the answer to Qn 1b is correct)	**b.** Give a simple one line comment explaining the functionality of the above code The program prints the list of duplicate characters in a given string. (1 mark - No partial marks)

Q2. (2 marks) Given an integer n, write **a single line statement** to form a list which is a list of digits of n. (E.g., if n is 147 then the list will be [1, 4, 7]).

Answer:

```
l = [int(str(n)[i]) for i in range(len(str(n)))]
```
or
```
l = [ int(v) for v in str(n)]
```
or
```
l= list(map(int, str(n)))
```

(2 marks, if the answer is in a single line and is correct.
1 mark, if the answer is in a single line, but has some minor syntax errors
0.5 marks, if the answer is in multiple lines, and the code executes without any logical or syntactical errors.
0 mark, for all other cases)

Q3. (3 marks) Given two lists of integers (l1 and l2) where l1 is smaller in length than l2. Write **one statement** to create a list l3 which contains the max of the corresponding numbers in l1 and l2 (when one list does not have a corresponding number, then max is the value in the other list). Note: the function max(x,y) returns the max of the two.

Answer:

```
l3 = [max(l1[i], l2[i]) for i in range(len(l1))] + [l2[i] for i in range(len(l1),len(l2))]
```
or
```
l3= [max(l2[i], l1[i]) if i<len(l1) else l2[i] for i in range(len(l2))]
```
or
```
l3 = [max(l1[i], l2[i]) for i in range(len(l1))] + l2[len(l1):]
```

(3 marks, if the answer is in a single line and is correct.
2 marks, if the answer is in a single line, but has one minor syntax error
1.5 marks, if the answer handles only the common indices i.e. `max(l1[i], l2[i]) for i in range(len(l1))`
1 mark, if the answer is in a single line, but has more than one syntax errors
0.5 marks, if the answer is in multiple lines, and the code executes without any logical or syntactical errors.
0 mark, for all other cases)

Q4. (1+1 marks) Consider the following function:

```
def g(t, count):
        for item in t:
            if isinstance(item, tuple): #True if the item is a
tuple
                g(item, count)
            else:
                if item in count:
                    count[item] += 1
                else:
                    count[item] = 1

def f(t):
    count = {}
    g(t, count)
    return count


test_tuple = (1, (2, (4, 3, 2)))
result = f(test_tuple)
print(result)
```

a. What will be the output of the above code?	**b.** Give a simple one line comment explaining the functionality of the code
{1: 1, 2: 2, 4: 1, 3: 1} (1 mark - No partial marks)	Counts the **frequency of elements from the tuple** and **returns** this as a **dictionary**

	(1 mark - No partial marks)

Q5. (4 marks) This is the "straightforward" recursive implementation of fibonacci number:

```
def fib(n):
    if n in {0, 1}:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Write a modified version of this program such that when calling fib with any n (e.g. n=5), fib for n = 1, 2, 3,4 is called exactly once (while still using recursion).

Answer:

```
cache = {0:0, 1:1}   # 1 mark - use of a global dictionary

def fib2(n):
  if n in cache:
       return cache[n]   # 1 mark - case of if n found in cache
  else:
       cache[n]=fib2(n-1)+fib2(n-2)
       return cache[n]    # 2 marks - case of if n not found in cache
```

Q6. (2+1 marks) The function below is intended to merge two dictionaries, where each key in the dictionaries stores a list of elements. For a common key in both dictionaries the values (lists) should be combined into a single list. There is, however, a bug in the code.

```
1. def merge_dicts(dict_1, dict_2):
2    new_dict = {}
3    for key in dict_1:
4      if key in dict_1 and key in dict_2:
5              new_dict[key] = dict_1[key][:] + dict_2[key][:]
6      elif key in dict_1:
7              new_dict[key] = dict_1[key][:]
8      else:
9              new_dict[key]= dict_2[key][:]
10   return new_dict

# Test data
dict_1 = {0: [1, 2], 1: [3, 4],2:[6,7]}
dict_2 = {0: [3], 1: [5]}

result = merge_dicts(dict_1, dict_2) # result is Merged
Dictionary:  #{0: [1, 2, 3], 1: [3, 4, 5],2:[6,7]}
```

a. Write a _test() function which has an assert statement for which the ***above code*** fails.	`def _test_merge_dicts():`
	`  dict_1={0:[1,2], 1:[3,4]}`
	`  dict_2={0:[1],2:[5]})`
	
	`  assert(merge_dicts(dict_1,dict_2))=={0:[1,2,1],1:[3,4],2:[5]}`
	`  # the error occurs when a different key is present in the second dictionary.`
	`  return`
	
	(2 marks - answer is in a function form, the correct case - different key present in the second dictionary - has been identified and the assert statement is also correct)
	1.5 mark - answer is not in a function form, but the rest is correct)
	1 mark - if the correct case has been identified, but the assert statement is not syntactically correct)
	0 - all other answers

b. Modify the code to fix the error (you can only change existing lines). Mention the line number(s), and the new code for each.	Code fix:
	Replacing line 3 with the below code :
	`keys=list(dict_1.keys())+list(dict_2.keys())`
	`for key in keys:`
	
	(1 mark - for the logically and syntactically correct code fix
	0.5 mark - if the fix is logically correct, but syntactically wrong - like, haven't used the .keys() method in the right way)
	
	The answer should be logically equivalent to the above, need not be exactly the same. The modified code should execute without errors and give the expected result of merging the two dictionaries correctly.

Q7. (2 + 2 marks) Given these two functions to determine the frequency of occurrences of unique numbers in a given list.

i)
```
def freq_order(lst):
    unique = []        # list of unique item
    freq =[]           # their frequencies
    for item in lst:
        if item in unique:     # already seen
            freq[unique.index(item)] += 1
        else:                  # new item
            unique.append(item)
            freq.append(1)
    return unique, freq
```

ii)
```
def freq_dict(lst):
    freq = {}        # empty dictionary
    for item in lst:
        if item in freq:       # already seen
            freq[item] += 1
        else:                  # new item
            freq[item] = 1
    return freq
```

a. Which of the above two functions (for a large input list) will be faster, and why (give the main reason in one sentence.)	The Dictionary will be faster as checking for an item in a dictionary is much faster than checking for an item in a list. (Adding an item in a dictionary and appending in a list take similar time.) (2 marks - for mentioning Dictionary and giving the correct reason 1 mark - for mentioning Dictionary, but without giving the **correct** reason)
b. Which of the above two functions (for a large input list) will use up more memory and why (give the main reason in one sentence.)	Dictionary. Dictionary uses hashing over keys, which makes accessing the value related to a key faster, but hashing requires much more space than the number of keys. (2 marks - for mentioning Dictionary and giving the correct reason of **hashing** 1 mark - for mentioning Dictionary, but without giving the **correct** reason)

Q8. (5 marks) We want to define a class `Point` to represent points in a 3-D space. The class should have methods to `shift` the point by three deltas (i.e. change) for x,y and z axis, to return the `distance` between this point and another given point, and a method which will return a suitable string when `print(p)` (where p is a Point) is called.

Define the class, and write the header of all the methods needed (you **don't need to write the body** of these methods.)

Answer:

```
class Point:   # 1 mark - correct class definition

def __init__(self, x,y,z):      # 1 mark
def shift(self, d1, d2, d3):    # 1 mark, can give 1 mark if the student has assumed
d1=d2=d3=d and has written def shift(self, d):
def distance(self, p): or def distance(self, x, y, z):    # 1 mark
def  __str__(self):             # 1 mark
```

Q9. (2 marks) What is the output of this program:

```
def myfn(f,t):
    l2 = f(t)
    l3 = [e*e for e in l2]
    return l3
def fn2(t):
    t[0], t[1] = t[0]+t[1], t[0]-t[1]
    return t

# main
lin = [1, 3]
lout = myfn(fn2,lin)
print(sum(lout))
```

Answer:

20

(2 marks - for correct answer
1 mark - if the final answer is not correct, but a correct explanation or working out of the code is shown)

Explanation (not necessary to be written as part of the answer for getting marks): The program uses a higher-order function, **`myfn`**, which takes **`fn2`** and a list **`lin`** as arguments. **`fn2`** modifies **`lin`** in-place to **`[4, -2]`**, then **`myfn`** squares these elements to get **`[16, 4]`**, and the **`sum`** function outside calculates their total, resulting in the output **`20`**.

Q10. (1+2 marks). Given this function for searching if an element x is in a sorted list. It returns the index of the searched element if it exists, or it returns -1 if the element does not exist. This function has a bug.

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:

        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
```

a. What is the bug in this function, and how will you fix it ?

Bug: If the element is not in the list, it will return None (rather than -1) (0.5 mark)

Fix: Add `return -1` after the while loop ends as below (0.5 mark)

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
```

b. Write two assert statements - one that will pass, and one that will reveal the bug.

```
assert binary_search([1, 2, 3, 4, 5], 3) == 2 # a case where the second argument is present in the list - will pass
assert binary_search([1,2,3,4,5], 6) == -1 # a case where the second argument is not present in the list - will fail as it returns None
```

```
while low <= high:

    mid = (high + low) // 2
    if arr[mid] < x:
        low = mid + 1
    elif arr[mid] > x:
        high = mid - 1
    else:
        print (mid)
        return mid
 return -1
```

========== End ==========