**General Test**

Question1.

I tried to find the solution of this problem in various way. From very simple to little more complex and recursive and brute force approach. but the give below seems the easiest to run and understand. Not very memory efficient though.

*# This function gives you the Fibonacci sequence certain value.*

**def** findfib(fn):

if fn < 1:

print("wrong number")

elif fn == 0:

return 0

elif fn == 1:

return 1

else:

return (findfib(fn-1) + findfib(fn-2))

print ("give you the certain fib number ->",findfib(10)) *#change the fn value to know the Fibonacci value in the sequence*

*################################################*

fn = [1,2]

while fn[-1] < 100:

    fn.append(fn[-1] + fn[-2])

print(fn)

evenf = []

for n in fn:

    if n % 2 ==0:

        evenf.append(n)

print(evenf)

sum =0

for n in evenf:

    sum += n

print(sum)

**output**

PS D:\TestPythonFiles> python -i fib.py

give you the certain fib number -> 55

[1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]

[2, 8, 34, 144]

188

Question 2.

They are, Hash table, Trees data structures, Dictionaries

Advantages: They are very efficient for finding, inserting and updating. Storing in hash table will not always work well as there is a chance that several elements may collide. Which is one of the main issues that has to be overcome when using hash tables. Hash functions are used to simplify the complex data in the hash table.

In contrast, the tree data structure is a collection of nodes and their main use is storing and sorting data. You cant have duplicate keys in a same way as you have duplicate key in hash table.

Dictionaries are the implantation of associative arrays in python.

Question 3.

**class** sort:

**def** \_\_init\_\_(self, data):

        self.data = data

        self.next = None

**class** list:

**def** \_\_init\_\_ (self):

        self.head =None

**def** peeklist(self):

        items = self.head

        while items:

            print(items.data, end = "->")

            items = items.next

**def** append(self,new\_data):

        new\_node = sort(new\_data)

        if self.head is None:

            self.head = new\_node

            return

        endnode = self.head

        while endnode.next:

            endnode = endnode.next

        endnode.next = new\_node

**def** mergelists(h1,h2):

    items = None

    if h1 is None:

        return h2

    if h2 is None:

        return h1

    if h1.data <= h2.data:

        items = h1

        items.next = mergelists(h1.next, h2)

    else:

        items = h2

        items.next = mergelists(h1, h2.next)

    return items

*#call the list objects in the linter and pass the values as mentioned in the question.*

if \_\_name\_\_ =="\_\_main\_\_":

list1 = list()

        list1.append(1)

        list1.append(4)

        list1.append(6)

        list2 = list()

        list2.append(2)

        list2.append(3)

        list2.append(5)

        list3= list()

        list3.head = mergelists(list1.head, list2.head)

        print("merged list",end ="")

        list3.peeklist()

**OUTPUT**

PS C:\Python34> python -i F:\linked\test\merge.py

merged list1->2->3->4->5->6->>>>

**Python Programming test**

**Question one.**

**concurrency python**

**Describe two different modules for exploiting concurrency Python, pointing out the strengths and weaknesses of each and of concurrency as a whole in Python.**

We know concurrency also by multiprocessing and parallel processing. In python uses concurrency to facilitate computers in running a program efficiently on complex distributed systems. Web servers are most simple example that can be given, for example web server is not responding to the user’s request until the previous one is completed. Concurrency modules are used where high-volume multithreaded work is required.

**Threading Module:**

The give code is one using python already placed API.

The example runs two, one is created the program and the second thread exist in the input reader class.

from threading import Thread

**class** InputReader(Thread):

**def** run(self):

self.line\_of\_text = input()

print("Enter some text and press enter: ")

thread = InputReader()

thread.start()

count = result = 1

while thread.is\_alive():

result = count \* count

count += 1

print("calculated squares up to {0} \* {0} = {1}".format(count, result))

print("while you typed '{}'".format(thread.line\_of\_text))

As soon the program executed the input is required, start () thread method is called on the object. While waiting for the input in the background thread continues to execute and count the squares in the while loop.

For example

yes

calculated squares up to 2397950 \* 2397950 = 5750159406601

while you typed 'yes'

>>>

**Multiprocessing module.**

Using thread library has its disadvantages. For more complex CPU intensive jobs and parallel computing multiprocessing library is designed.

The first 9 lines are the process IDs that are printed inside the CPUprocess instance the last line shows how much time it took for my CPU to compute 20 million interactions which was about 0.7 sec. The printed process ID are the unique number assuaged by the OS to each process.

from multiprocessing import Process, cpu\_count

import time

import os

**class** CPUprocess(Process):

**def** run(self):

print(os.getpid())

for i in range(2000000):

pass

if \_\_name\_\_ == '\_\_main\_\_':

procs = [CPUprocess() for f in range(cpu\_count())]

t = time.time()

for p in procs:

p.start()

for p in procs:

p.join()

print('work took {} seconds'.format(time.time() - t))

41972

39004

40540

21252

28376

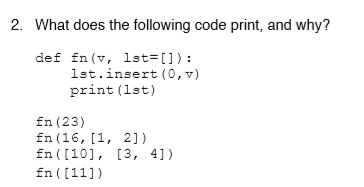
42356

7780

62520

work took 0.6459980010986328 seconds

**Question two**



The output of the function will be:

[23]

[16, 1, 2]

[[10], 3, 4]

[[11], 23]

The reason, python stores the list in by taking 2 parameters,

Index, and element. In the above function, python insert function is used to insert the element.

Lst.insert put the value of v at the 0th index of the list. When the function called 23 is returned. Other values in the output are return as result of function call and by passing those values to the function fn.

**Question three.**

**def** eveNumSq():

elist=[1,2,3,4,5,6,7,8,9,10]

sqlist = []

for i in elist:

if i % 2==0:

sqlist.append(i\*i)

listsum=sum(sqlist)

print ("even integer list is ->", sqlist, "sum of even numbers is in the given list is -> ",listsum)

eveNumSq()

PS D:\TestPythonFiles> python -i F:\linked\test\merge.py

even integer list is -> [4, 16, 36, 64, 100] sum of even numbers is in the given list is -> 220

>>>

**Java Test**

**Question 1**

**public** **class** Doubly {

**Node** head; *// head of list*

*/\* Doubly Linked list Node\*/*

**class** Node {

**int** data;

**Node** prev;

**Node** next;

*// Constructor to create a new node*

*// next and prev is by default initialized as null*

Node(**int** d) { data = d; }

}

*// Adding a node at the front of the list*

**public** **void** push(**int** new\_data)

{

*/\* 1. allocate node*

*\* 2. put in the data \*/*

**Node** new\_Node = new Node(new\_data);

*/\* 3. Make next of new node as head and previous as NULL \*/*

new\_Node.next = head;

new\_Node.prev = null;

*/\* 4. change prev of head node to new node \*/*

if (head != null)

head.prev = new\_Node;

*/\* 5. move the head to point to the new node \*/*

head = new\_Node;

}

*/\* Given a node as prev\_node, insert a new node after the given node \*/*

**public** **void** InsertAfter(**Node** prev\_Node, **int** new\_data)

{

*/\*1. check if the given prev\_node is NULL \*/*

if (prev\_Node == null) {

System.out.println("The given previous node cannot be NULL ");

return;

}

*/\* 2. allocate node*

*\* 3. put in the data \*/*

**Node** new\_node = new Node(new\_data);

*/\* 4. Make next of new node as next of prev\_node \*/*

new\_node.next = prev\_Node.next;

*/\* 5. Make the next of prev\_node as new\_node \*/*

prev\_Node.next = new\_node;

*/\* 6. Make prev\_node as previous of new\_node \*/*

new\_node.prev = prev\_Node;

*/\* 7. Change previous of new\_node's next node \*/*

if (new\_node.next != null)

new\_node.next.prev = new\_node;

}

*// Add a node at the end of the list*

**void** append(**int** new\_data)

{

*/\* 1. allocate node*

*\* 2. put in the data \*/*

**Node** new\_node = new Node(new\_data);

**Node** last = head; */\* used in step 5\*/*

*/\* 3. This new node is going to be the last node, so*

*\* make next of it as NULL\*/*

new\_node.next = null;

*/\* 4. If the Linked List is empty, then make the new*

*\* node as head \*/*

if (head == null) {

new\_node.prev = null;

head = new\_node;

return;

}

*/\* 5. Else traverse till the last node \*/*

while (last.next != null)

last = last.next;

*/\* 6. Change the next of last node \*/*

last.next = new\_node;

*/\* 7. Make last node as previous of new node \*/*

new\_node.prev = last;

}

*// This function prints contents of linked list starting from the given node*

**public** **void** printlist(**Node** node)

{

**Node** last = null;

System.out.println("Traversal in forward Direction");

while (node != null) {

System.out.print(node.data + " ");

last = node;

node = node.next;

}

System.out.println();

System.out.println("Traversal in reverse direction");

while (last != null) {

System.out.print(last.data + " ");

last = last.prev;

}

}

*/\* Drier program to test above functions\*/*

**public** **static** **void** main(**String**[] args)

{

*/\* Start with the empty list \*/*

**Doubly** dll = new Doubly();

*// Insert 6. So linked list becomes 6->NULL*

dll.append(6);

*// Insert 7 at the beginning. So linked list becomes 7->6->NULL*

dll.push(7);

*// Insert 1 at the beginning. So linked list becomes 1->7->6->NULL*

dll.push(1);

*// Insert 4 at the end. So linked list becomes 1->7->6->4->NULL*

dll.append(4);

*// Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL*

dll.InsertAfter(dll.head.next, 8);

System.out.println("Created DLL is: ");

dll.printlist(dll.head);

}

}

**Question 3.**

The finalise method is called during the garbage collection to free the system memory. Before the object is collected the runtime, system calls its finalise method. The intent is for finalise method to release system resources such as open files or socket. The method is created in protected void non-static, the method will be available to all objects before it is destroyed.