

2020 SH1 Promo Practical Solutions:

Question 1 [24]

- 1.1** • Line processing [2]
 • iteration

```
f= open("PARTICIPANTS.TXT")
participants=[]
for line in f:
    participants.append( line.strip().split(";") )
```

- 1.2** • iteration [3]
 • update counters
 • return tuple

```
def gender_count(participants):
    male_count=0
    female_count=0
    for participant in participants:
        if participant[2] == "M":
            male_count+=1
        else:
            female_count+=1
    return male_count, female_count
```

- 1.3** [4]

```
def role_statistics(participants):
    role_dict={} #[1]
    for participant in participants:
        if participant[1] not in role_dict:
            role_dict[participant[1]] = [participant]
    #[2]
        else:
            role_dict[participant[1]].append(participant)
    print(f"{'Role':<12}{'Number'}")
    for role in role_dict: #[1]
        print(f"{'role':<12}{len(role_dict[role])}")
    return role_dict
```

- 1.4** import random [10]

```
def form_random_group(participants):
    team=[]
    roles={}
    ## Group participants into roles #[5]
    for participant in participants:
        if participant[1] not in roles:
            roles[participant[1]] = [participant]
        else:
            roles[participant[1]].append(participant)
    ## Pick member for each role #[5]
    for role in ["Coder", "Manager", "Maker", "Designer",
    "Tester"]:
        if role not in roles:
            return []
        i = random.randint(0,len(roles[role])-1)
        team.append(roles[role][i])
        participants.remove(roles[role].pop(i))
    return team
```

```
1.5 f=open("GROUPS.TXT","w") [5]
    group_count=0
    while True: #[1]
        group = form_random_group(participants) #[1]
        group_count+=1 #[1]
        if group:
            f.write(f"Group {group_count}\n")
            for record in group: #[1]
                f.write(",".join(record)+"\n")
        else:
            break
    f.close()
    ## 7 groups [2]
```

Question 2 [15]

2.1 `a = int(input("Enter a: "))`
`r = int(input("Enter r: "))`
`m = int(input("Enter m: "))` [1]

`def gsum(a, r, m):`
 `if m == 1: #base case 1`
 `total = a * r` [3]
 `print(total, end = " ")`
 `return total`
 `else:`
 `item = a * r**m` [1]
 `total = item + gsum(a, r, m-1) #recursive step` [2]
 `print(item, end = " ")`
 `return total` [2]

`total = gsum(a, r, m)`
`print()`
`print(total)`

`def Fibonacci(n):`
 `results=[None for i in range(n+1)]` [1]
 `results[0], results[1] = 0,1` [1]
 `for i in range(2,n+1):` [2]
 `results[i]=results[i-1]+results[i-2]` [2]
 `return results[n]`

Question 3 (42)

Task Answers

Marks

- 3.1** for both Patient and Visitor
- Constructors [2]
 - `__str__()` [2]
 - `getters()` [2] all private attributes must have getters, -1 for 1 missing getter

```
class Patient:
    def __init__(self, name, NRIC):
        self._name = name
        self._NRIC = NRIC
    def __str__(self):
        return self._name
    def getName(self):
        return self._name
    def getNRIC(self):
        return self._NRIC

class Visitor:
    def __init__(self, name, contact):
        self._name = name
        self._contact = contact
    def __str__(self):
```

```

        return f"{self._name}:{self._contact}"
def getName(self):
    return self._name
def getContact(self):
    return self._contact

```

- 3.2**
- correct output for Patient [1]
 - correct output for Visitor[1]

2

3.3

```

class Queue:
    def __init__(self):
        self.size = 10
        self.head = -1
        self.tail = 0
        self._buffer = [None for _ in range(self.size)]

    def isFull(self):
        return self.head == self.tail

    def isEmpty(self):
        return self.head == -1

    def enqueue(self, visitor):
        if self.isFull():
            return False
        self._buffer[self.tail] = visitor
        self.tail = (self.tail + 1) % self.size
        if self.head == -1:
            self.head += 1
        return True

    def dequeue(self):
        if self.isEmpty():
            return None

        ret = self._buffer[self.head]
        self._buffer[self.head] = None
        self.head = (self.head + 1) % self.size

        if self.tail == self.head:
            self.head = -1
            self.tail = 0

        return ret

    def __str__(self):
        ret=[]
        if self.isEmpty():
            return "[]"
        cur = self.head
        while True:
            ret.append(str(self._buffer[cur]))
            cur = (cur+1)%self.size
            if cur == self.tail:
                break
        return f"[{'','.join(ret)}]"

```

- constructor() [2] 10
 - correct number of items in array initialised to None
 - initialise head, tail positions for empty queue
- enqueue()
 - check for queue full [1]
 - add new item to item queue, update head, tail [2],
- correct head, tail positions for full queue [1]
- dequeue()
 - check for empty queue [1]
 - remove item from queue by reset value to None[1]
 - check for empty queue, reset head, tail positions to empty queue[1]
- __str__()
 - Iterate from self.head to self.tail [1]

Task 3.4 • Items in queue are in FIFO order and correct format [2] 2

Task 3.5 • constructor() [2] -1 for any missing attributes or uninitialised attribute 3
 • __str__() [1]

```
class Bed:
    def __init__(self,f,w,b, visitStart, visitEnd):
        self.floor = f
        self.ward = w
        self.bedNo = b
        self.visitHourStart = visitStart
        self.visitHourEnd = visitEnd
        self.occupiedBy = None
        self.queue = Queue()
    def __str__(self):
        return f"{self.floor}-{self.ward}-{self.bedNo}:{str(self.occupiedBy)}"
```

Task 3.6 • constructor() 14

- -1 for any missing attributes or uninitialised attribute [2]
- Correct number of elements in beds array initialised[1]
- 3 loop to to create Bed objects in beds array [2]
- Different visiting hours [1]
- hash()
 - Check for valid range of floor , ward and bed in arguments [1]
 - Correct calculation of index [1]
- occupy() [1]
- showOccupancy()
 - loop to get the index and items in the beds array [1]
 - display only occupied beds [1]
- visit()
 - check for occupancy [1]
 - check for valid visitation hours [1]
 - enqueue visitor at the correct bed element [1]

```
class Hospital:
    def __init__(self, floors,wards,beds):
        self.noFloors=floors
        self.noWards=wards
```

```

        self.noBeds=beds
        self.beds = [None] * (floors*wards*beds)
        for f in range(1, floors+1):
            for w in range(1, wards+1):
                for b in range(1,beds+1):
                    if f == floors:
                        self.beds[self._hash(f,w,b)] =
Bed(f,w,b,17,19)
                    else:
                        self.beds[self._hash(f,w,b)] =
Bed(f,w,b,12,20)

        def _hash(self, floor, ward, bed):
            if 0<floor<=self.noFloors and 0<ward<=self.noWards and
0<bed<=self.noBeds :
                return (floor - 1) * (self.noWards*self.noBeds) +
(ward-1)*(self.noBeds) + (bed-1)

        def occupy(self, patient, floor, ward, bedNo):
            index=self._hash(int(floor), int(ward), int(bedNo))
            self.beds[index].occupiedBy = patient

        def showOccupancy(self):
            for i, b in enumerate(h.beds):
                print(f"{i}->{b} {b.queue}") if b.occupiedBy != None
else ""
        def visit(self,visitor, floor, ward, bed, timeStamp):
            index = self._hash(floor, ward, bed)
            bed = self.beds[index]
            if bed.occupiedBy == None:
                return False
            if timeStamp.hour >= bed.visitHourStart and
timeStamp.hour <= bed.visitHourEnd:
                bed.queue.enqueue(visitor)
                return True
            else:
                return False

```

Task • Correct reading of file [1] 5

3.7 • Correct initialisation of Hospital object [1]
• Iteration to populate beds [1]
• Correct output after 2 visitors visit a patient [2]

```

f = open("PATIENTS.CSV")
patients = f.readlines()
f.close()
noFloor, noWard, noBed = patients[0].strip().split(",")
h=Hospital(int(noFloor), int(noWard), int(noBed))
h.showOccupancy()
for p in patients[1:]:
    name, nric, floor, ward, bedNo = p.strip().split(",")
    h.occupy( Patient(name, nric), floor, ward, bedNo)
h.showOccupancy()

import datetime as dt

```

```

v1 = Visitor("Abbott", "955-505-11", 37.5)
v2 = Visitor("Norby", "955-535-82", 35.6 )
time1 = dt.datetime(2020, 1, 1, 13, 0) ## y,m,d,h,min
h.visit(v1, 3, 2, 2, time1)
h.visit(v2, 3, 2, 2, time1)
h.showOccupancy()

```

4		
4.1	<ul style="list-style-type: none"> • SQL CREATE statement for each table [2] • FOREIGN KEY REFERENCE [1] 	3
4.2	<ul style="list-style-type: none"> • Read file [1] • INSERT statement in correct order [2] 	3
4.3	<ul style="list-style-type: none"> • Jinja-template for form [1] • Css in form [1] • View function for home page [1] 	3
4.4	<ul style="list-style-type: none"> • SQL SELECT statement [1] • Process result [1] • Render template with argument [1] • Jinja template with jinja variables [1] • Css in jinja template [1] 	5