MARKING SCHEME

Class: XII Session: 2024-25

Computer Science (083)

Time allowed: 3 Hours **Maximum Marks: 70**

Q No.	SECTION A (21X1=21)	Marks
1.	False *(1 mark for correct answer)*	(1)
2.	(A) #THONPROGRAM *(1 mark for correct answer)*	(1)
3.	(A) not (True) and False *(1 mark for correct answer)*	(1)
4.	(B) ['I', 'ter', 'atio', 'al'] *(1 mark for correct answer)*	(1)
5.	ce lo *(1 mark for correct answer)*	(1)
6.	(B) False *(1 mark for correct answer)*	(1)
7.	(B) print(my_dict['apple', 'banana']) *(1 mark for correct answer)*	(1)
8.	(B) Removes the first occurrence of value x from the list *(1 mark for correct answer)*	(1)
9.	(C) 3 *(1 mark for correct answer)*	(1)
10.	file.seek(0) (OR file.seek(0,0)) *(1 mark for correct answer)*	(1)
11.	False *(1 mark for correct answer)*	(1)
12.	(C) 12#15% *(1 mark for correct answer)*	(1)
13.	Alter (or Alter Table) *(1 mark for correct answer)*	(1)

14.	(A) Details of all products whose names start with 'App' *(1 mark for correct answer)*	(1)
15.	(D) CHAR *(1 mark for correct answer)*	(1)
16.	(B) count() *(1 mark for correct answer)*	(1)
17.	(B) FTP *(1 mark for correct answer)*	(1)
18.	(B) Gateway *(1 mark for correct answer)*	(1)
19.	(B) Packet Switching *(1 mark for correct answer)*	(1)
20.	(C) A is True but R is False. *(1 mark for correct answer)*	(1)
21.	(C) A is True but R is False. *(1 mark for correct answer)*	(1)

Q No.	**SECTION B (7 X 2 =14)**	**Marks**
22.	A mutable object can be updated whereas an immutable object cannot be updated. Mutable object: [1,2] or {1:1,2:2} (Any one) Immutable object: (1,2) or '123' (Any one) *(1 mark for correct difference)* *(½ x 2 = 1 Mark for selecting correct objects)*	(2)
23.	(I) Arithmetic operators: +,- (II) Relational operators: >, >= *(½ x 4 = 2 Marks for each correct operator)*	(2)
24.	(I) A) L1.count(4) OR B) L1.sort() *(1 mark for correct answer)*	(2)

	(II) A) L1.extend(L2) OR B) L2.reverse() *(1 mark for correct answer)*	
25.	(A), (C) *(½ x 2 = 1 Mark)* Minimum and maximum possible values of the variable b: 1,6 *(½ x 2 = 1 Mark)*	(2)
26.	def swap_first_last(tup**):** if len(tup) < 2: **return tup** new_tup = (tup[-1],) + tup[1:-1] + (tup**[0],)** return new_tup result = swap_first_last((1, 2, 3, 4)) print("Swapped **tuple:", result)** (½ mark each for correcting 4 mistakes)	(2)
27.	(I) A) UNIQUE **OR** B) NOT NULL *(1 mark for correct answer)* (II) A) ALTER TABLE MOBILE DROP PRIMARY KEY; OR B) ALTER TABLE MOBILE ADD PRIMARY KEY (M_ID); *(1 mark for correct answer)*	(2)
28.	A) Advantage: Network extension is easy. Disadvantage: Failure of switch/hub results in failure of the network. *(1 mark for correct Advantage)* *(1 mark for correct Disadvantage)* **OR**	(2)

	B) SMTP: Simple Mail Transfer Protocol. SMTP is used for sending e-mails from client to server. *(1 mark for correct expansion)* *(1 mark for correct usage)*	

Q No.	SECTION C (3 X 3 = 9)	Marks

29.	(A)
	def show():
	f=open("Email.txt",'r')
	data=f.read()
	words=data.split()
	for word in words:
	if '@cmail' in word:
	print(word,end=' ')
	f.close()
	(½ mark for correct function header)
	(½ mark for correctly opening the file)
	(½ mark for correctly reading from the file)
	(½ mark for splitting the text into words)
	(1 mark for correctly displaying the desired words)
	OR
	(B)
	def display_long_words():
	with open("Words.txt", 'r') as file:
	data=file.read()
	words=data.split()
	for word in words:
	if len(word)>5:
	print(word,end=' ')
	(½ mark for correct function header)
	(½ mark for correctly opening the file)
	(½ mark for correctly reading from the file)
	(½ mark for splitting the text into words)
	(1 mark for correctly displaying the desired words)
		(3)

30. (A)

(I)

```
def push_book(BooksStack, new_book):
    BooksStack.append(new_book)
```

(II)

```
def pop_book(BooksStack):
    if not BooksStack:
        print("Underflow")
    else:
        return(BookStack.pop())
```

(III)

```
def peep(BooksStack):
    if not BooksStack:
        print("None")
    else:
        print(BookStack[-1])
```

(3x1 mark for correct function body; No marks for any function header as it was a part of the question)

OR

(B)

```
def push_even(N):
    EvenNumbers = []
    for num in N:
        if num % 2 == 0:
            EvenNumbers.append(num)
    return EvenNumbers


VALUES = []

for i in range(5):
    VALUES.append(int(input("Enter an integer: ")))

EvenNumbers = push_even(VALUES)

def pop_even():
    if not EvenNumbers:
        print("Underflow")
    else:
        print(EvenNumbers.pop())

pop_even()
```

(3)

	def Disp_even():
	  if not EvenNumbers:
	  print("None")
	  else:
	  print(EvenNumbers[-1])
	Disp_even()
	(1/2 for identifying even numbers)
	(1/2 mark for correctly adding data to stack)
	(1/2 mark for correctly poping data on the stack and 1/2 mark for checking condition)
	(1/2 mark for correctly displaying the data with none)
	(1/2 mark for function call statements)	

31.	(A) 15@ 7@ 9 OR (B) 1 #2 #3# 1 #2 #3 # 1 # *(1 mark for each correct line of output)* *(deduct ½ mark for not printing @/#)*	(3)

Q No.	SECTION D (4 X 4 = 16)	Marks
32.	(A) (I) select Product, sum(Quantity) from orders group by product having sum(Quantity)>=5; (II) select * from orders order by Price desc; (III) select distinct C_Name from orders; (IV) select sum(price) as total_price from orders where Quantity IS NULL; *(4 x 1 mark for each correct query)* **OR** (B) (I) C_Name | Total_Quantity --- |--- Jitendra | 1 Mustafa | 2 Dhwani | 1	(4)

(II)

```
O_Id | C_Name | Product        | Quantity  | Price
-----  |--------     |------------      |----------    |-------
1002 | Mustafa  | Smartphone | 2           | 10000
1003 | Dhwani   | Headphone  | 1           | 1500
```

(III)

```
O_Id | C_Name   | Product        | Quantity | Price
-----       |----------     |------------      |----------    |-------
1001      | Jitendra    | Laptop         | 1           | 12000
1002      | Mustafa    | Smartphone | 2           | 10000
1003      | Dhwani    | Headphone   | 1           | 1500
```

(IV)

```
MAX(Price)
-----------
12000
```

(4 x *1 mark for each correct output)*

33. (I)

```
def show():
    import csv
    f=open("happiness.csv",'r')
    records=csv.reader(f)
    next(records, None) #To skip the Header row
    for i in records:
        if int(i[1])>5000000:
            print(i)
    f.close()
```

(½ mark for opening in the file in right mode)
(½ mark for correctly creating the reader object)
(½ mark for correctly checking the condition)
(½ mark for correctly displaying the records)

(4)

(II)

```
def Count_records():
    import csv
    f=open("happiness.csv",'r')
    records=csv.reader(f)
    next(records, None) #To skip the Header row
    count=0
    for i in records:
        count+=1
    print(count)
    f.close()
```

	(½ mark for opening in the file in right mode) *(½ mark for correctly creating the reader object)* *(½ mark for correct use of counter)* *(½ mark for correctly displaying the counter)* **Note (**for both parts (I) and (II)**):** *(i) Ignore* **import csv** *as it may be considered the part of the complete program, and there is no need to import it in individual functions.* *(ii) Ignore next(records, None) as the file may or may not have the Header Row.*	
34.	(I) Select * from FACULTY natural join COURSES where Salary<12000; Or Select * from FACULTY, COURSES where Salary<12000 and facuty.f_id=courses.f_id; (II) Select * from courses where fees between 20000 and 50000; (III) Update courses set fees=fees+500 where CName like '%Computer%'; (IV) (A) Select FName, LName from faculty natural join courses where Came="System Design"; Or Select FName, LName from faculty, courses where Came="System Design" and facuty.f_id=courses.f_id; OR (B) Select * from FACULTY, COURSES; *(4x1 mark for each correct query)*	(4)

35.	def AddAndDisplay():
	import mysql.connector as mycon
	mydb=mycon.connect(host="localhost",user="root",
	passwd="Pencil",database="ITEMDB")
	mycur=mydb.cursor()
	no=int(input("Enter Item Number: "))
	nm=input("Enter Item Name: ")
	pr=float(input("Enter price: "))
	qty=int(input("Enter qty: "))
	query="INSERT INTO stationery VALUES ({},'{}',{},{})"
	query=query.format(no,nm,pr,qty)
	mycur.execute(query)
	mydb.commit()
	mycur.execute("select * from stationery where price>120")
	for rec in mycur:
	print(rec)
		(4)

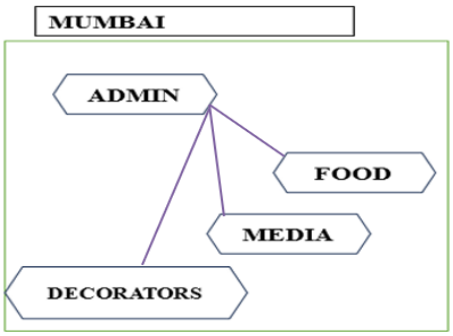
	(½ mark for correctly importing the connector object) *(½ mark for correctly creating the connection object)* *(½ mark for correctly creating the cursor object)* *(½ mark for correctly inputting the data)* *(½ mark for correct creation of first query)* *(½ mark for correctly executing the first query with commit)* *(½ mark for correctly executing the second query)* *(½ mark for correctly displaying the data)*	

Q No.	SECTION E (2 X 5 = 10)	Marks

36.	(I)
	import pickle
	
	def input_candidates():
	  candidates = []
	  n = int(input("Enter the number of candidates you want to add: "))
	  for i in range(n):
	  candidate_id = int(input("Enter Candidate ID: "))
	  candidate_name = input("Enter Candidate Name: ")
	  designation = input("Enter Designation: ")
	  experience = float(input("Enter Experience (in years): "))
	  candidates.append([candidate_id, candidate_name, designation, experience])
	  return candidates
	candidates_list = input_candidates()
	
	def append_candidate_data(candidates):
	  with open('candidates.bin', 'ab') as file:
	  for candidate in candidates:
	  pickle.dump(candidate, file)
	  print("Candidate data appended successfully.")
	
	append_candidate_data(candidates_list)
	
	(II)
	import pickle
	
	def update_senior_manager():
	  updated_candidates = []
	  try:
	  with open('candidates.bin', 'rb') as file:
	  while True:
	  try:
	  candidate = pickle.load(file)
	  if candidate[3] > 10: # If experience > 10 years
	  candidate[2] = 'Senior Manager'
	  updated_candidates.append(candidate)
	  except EOFError:
		(5)

	break # End of file reached
	except FileNotFoundError:
	print("No candidate data found. Please add candidates first.")
	return
	
	with open('candidates.bin', 'wb') as file:
	for candidate in updated_candidates:
	pickle.dump(candidate, file)
	
	print("Candidates updated to Senior Manager where applicable.")
	update_senior_manager()
	
	(III)
	
	import pickle
	
	def display_non_senior_managers():
	try:
	with open('candidates.bin', 'rb') as file:
	while True:
	try:
	candidate = pickle.load(file)
	if candidate[2] != 'Senior Manager': # Check if not Senior Manager
	print(f"Candidate ID: {candidate[0]}")
	print(f"Candidate Name: {candidate[1]}")
	print(f"Designation: {candidate[2]}")
	print(f"Experience: {candidate[3]}")
	print("----------")
	except EOFError:
	break # End of file reached
	except FileNotFoundError:
	print("No candidate data found. Please add candidates first.")
	
	display_non_senior_managers()
	
	(1/2 mark of import pickle)
	(1/2 mark for input)
	(1/2 mark for opening file in append mode and 1/2 mark for using dump)
	(1/2 mark for opening file in read mode and 1/2 mark for using load)
	(1 mark for checking the condition and updating the value)
	(1 mark for checking the condition and displaying data correctly)	

37.	(I) ADMIN Block as it has maximum number of computers. *(1 mark for correct answer)* (II) Switch *(1 mark for correct answer)* (III)	(5)



(or Any other correct layout)

Cable: Coaxial cable

(½ mark for correct layout + ½ mark for correct table type)

(IV) There is no requirement of the Repeat as the optical fibre cable used for the network can carry the data to much longer distances than within the campus.

(1 mark for correct answer)

(V) (A) a) Video Conferencing

OR

(B) LAN

(1 mark for correct answer)