2020 SH2 Prelim H2 Computing 9569 Paper 2 Marking Scheme/Solutions

Q1 [25]

|  |  |  |
| --- | --- | --- |
| **Task** | **Answers** | **Marks** |
| **1.1** | * Iterate though all the words in file [1] * Use a dictionary object to store the words using length of word as key [1] * Truncate "\n" in word [1] * Determine the shortest word and the longest word in the file.[1] * Prompt with correct range of word length [1] * Return the word from the dictionary object. [1] | 6 |
|  | import random  def getSecret(guess\_len=0): # 0 for standalone mode  f= open("WORDS.TXT","r")  words={}  for word in f:  word = word.strip()  length = len(word)  if length in words:  words[length].append(word)  else:  words[length]=[word]  min\_len = min(words.keys())  max\_len = max(words.keys())  if guess\_len == 0:  guess\_len = int(input(f"Choose the length of word to guess:[{min\_len} to {max\_len}"))  else: ## client/server mode  guess\_len = random.randint(min\_len, max\_len)  no\_words = len( words[guess\_len])  secret = words[guess\_len][ random.randint(0,no\_words-1)]  return secret | |
| **1.2** | * Generate secret word [1] * Variable to store the letters entered by the user [1] * Initialise the number of guesses [1] * Loop for the number of attempts [1] * Compare letter entered by user with the secret word [1] * Reveal the letter if correct guess [1] * Termiate if guess is correct or attempts ran out [1] | 7 |
|  | def reveal(secret, guessed):  ret=""  for ch in secret:  ret += ch if ch in guessed else "."  return ret  secret = getSecret()  num\_guess = len(secret)\*2  #guessed= cheat(secret)  guessed =[]  ret=reveal(secret, guessed)  print(f"You have {num\_guess} attempts. Good Luck!\n{ret}")  for i in range(num\_guess):  ans=input(f"[{num\_guess-i}]Guess letter:")  guessed.append(ans)  ret = reveal(secret, guessed)  print(ret)  if ret == secret:  print("WIN")  break  else:  print("LOSE..The word is", secret) | |
| **1.3** | * Calculate the number of letters to reveal [1] * Loop to select the letters to reveal[1] * Random unique indexs of letters to reveal [2] | 4 |
|  | **import** random,math ## reveal 20% of the word  **def** cheat(secret):      num\_to\_reveal **=** math.ceil(0.2 \* len(word))  **return** [ secret[i] **for** i **in** random.sample(range(len(secret)),num\_to\_reveal)] | |
| **1.4** | * Create socket,bind and listen [1] * Outer loop, accept inside Outer loop [1] * Generate secret word using a random length[1] * For/while loop [1] * Inside for/while loop, send "Win" + break, "Lose" or hidden word to client [1] | 5 |
|  | def reveal(secret, guessed):  ret=""  for ch in secret:  ret += ch if ch in guessed else "."  return ret  import socket  listen\_socket = socket.socket()  listen\_socket.bind(("127.0.0.1",9999) )  listen\_socket.listen()  while True:  active\_socket, remote\_addr =listen\_socket.accept() ## block until a connection comes in  secret = getSecret(1)  num\_guess = len(secret)\*2  guessed= cheat(secret)  ret=reveal(secret, guessed)  send\_msg = f"You have {num\_guess} attempts. Good Luck!\n{ret}"  active\_socket.sendall(send\_msg.encode())  for i in range(num\_guess):  recv\_msg = active\_socket.recv(1024).decode()  guessed.append(recv\_msg.strip())  ret = reveal(secret, guessed)  if ret == secret:  active\_socket.sendall(b"WIN\n")  print("WIN")  break  elif i == num\_guess-1:  send\_msg = f"LOSE..The word is {secret}"  print(send\_msg)  active\_socket.sendall(send\_msg.encode())  else:  send\_msg = ret.encode()  active\_socket.sendall(send\_msg)  active\_socket.close()  listen\_socket.close() | |
| **1.5** | * Connect to server ip, port [1] * While loop to show response from server till win or lose is detected [1] * Prompt and send letter to guess [1] | [3] |
|  | import socket  s = socket.socket()  s.connect(("127.0.0.1",9999))  count=0  while True:  try :  count+=1  recev\_mesg = s.recv(1024).decode()  print( recev\_mesg )  if "WIN" in recev\_mesg or "LOSE" in recev\_mesg:  break  guess = input(f"[{count}]Guess letter:")  s.sendall(guess.encode())  except:  break  s.close()  print("Client Stop") | |

Q2[ 10]

|  |  |  |
| --- | --- | --- |
|  | **Answers** | **Marks** |
| **2.1** | * Load json, iterate [1] * Remove duplicates [1] * Init ratings without a value to 0 [1] | 3 |
|  | ## Data cleaning to remove duplicates and string in rating  **import** json  f**=**open(**"RESTAURANT.JSON"**,**"r"**)  data **=** json.load(f)  names**=**[]  restaurants**=**[]  **for** restaurant **in** data:  **if** restaurant[**"name"**] **not** **in** names:          names.append(restaurant[**"name"**])          restaurant[**"rating"**] **=** **0** **if** type(restaurant[**"rating"**]) **==** str  **else** restaurant[**"rating"**]          restaurants.append(restaurant)  f.close()  # | |
| **2.2** | Helper function merge [2]   * Recursive calls [1] * Base case[1] | 4 |
|  | # Merge Sort  **def** merge(A,B):      merged**=**[]  **while** len(A) **and** len(B):  **if** A[**0**][**"rating"**] **>** B[**0**][**"rating"**]:              merged.append(A.pop(**0**))  **else**:              merged.append(B.pop(**0**))  **if** len(A) **==** **0**:          merged **+=** B  **else**:          merged **+=** A  **return** merged  **def** mergeSort(L):  **if** len(L) **<** **2**:  **return** L      mid **=** len(L)**//2**  **return** merge( mergeSort(L[:mid]), mergeSort(L[mid:]) ) | |
| **2.3** | Correct result | 1 |
|  |  |  |
| **2.4** | **import** pymongo  client **=** pymongo.MongoClient(**"localhost"**, **27017**)  sorted\_restaurants **=** mergeSort(restaurants)  client[**"Entertainment"**][**"Restaurants"**].insert\_many(sorted\_restaurants)  client.close() | 2 |
|  |  | |

Q3 (48)

|  |  |  |
| --- | --- | --- |
| **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 | 6 |
| 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, temp):  self.\_name = name  self.\_contact = contact  self.\_temperature = temp  def \_\_str\_\_(self):  return f"{self.\_name}:{self.\_temperature}"  def getTemperature(self):  return self.\_temperature  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** | * constructor() [1] * enqueue()   + create a queue for storing max age [1]   + loop to empty max age queue for age in queue that are less than age of new item [2]   + append new item to max queue and item queue [2] * dequeue()   + check for empty list [1]   + check if dequeuing item with max age [2] * peekMaxAge() [1]   + O(1) [1] * \_\_str\_\_()   + Iterate over list [1]   + Correct format[2]   NOTE: As discussed in class, the correct solution is to implement buffer using a Linked List instead of a Python List ,and keep a first and last pointers  REASON: If you are using Python list, it is not guranteed that the append() and the pop(0) are O(n) operations. | 14 |
| class Queue:  def \_\_init\_\_(self):  self.buffer=[]  self.maxQ=[]  def enqueue(self, obj):  buf\_last=len(self.buffer)-1  # if buf\_last < 0: ##empty  # self.buffer.append(obj)  # self.maxQ.append(obj)  # return  # else:  max\_last = len(self.maxQ)-1  while max\_last > -1 and self.maxQ[max\_last].getTemperature() < obj.getTemperature():  self.maxQ.pop()  max\_last = len(self.maxQ)-1  self.maxQ.append(obj)  self.buffer.append(obj)  def dequeue(self):  if len(self.buffer) == 0:  return None  ret=self.buffer.pop(0)  if ret == self.maxQ[0]:  self.maxQ.pop(0)  return  def peekMaxTemp(self):  return self.maxQ[0] if len(self.maxQ) > 0 else None  def \_\_str\_\_(self):  ret=""  for obj in self.buffer:  ret += f"{str(obj)},"    return f"[{ret[:-1]}]{self.peekMaxTemp()}" | | |
| **Task 3.4** | * Items in queue are in FIFO order [2] * Max age object is correct [1] * Outpit format is correct [1] | 4 |
| **Task 3.5** | * constructor() [2] -1 for any missing attributes or uninitialsed attribute * \_\_str\_\_() [1] | 3 |
| 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()   + -1 for any missing attributes or uninitialsed 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 corrcet bed element [1] | 14 |
| 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)  visitor.timeStamp = timeStamp  return True  else:  return False | | |
| **Task 3.7** | * Correct reading of file [1] * Correct initialisation of Hospital object [1] * Iteration to populate beds [1] * Correct output after 2 visitors visit a patient [2] | 5 |
|  | 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() |  |

Q4 15 marks

|  |  |  |
| --- | --- | --- |
| **Task** | **Answers** | **Marks** |
| **4.1** | * Tables created [1] * Location table populated [1] | **2** |
| **4.2** | * Action with correct url [1] * Name, NRIC, Contact input elements [1] * boostratp code for flask [1] * decorator for view function[1] * render template with argument[1] | **5** |
|  | checkin.html  <form action="/check\_in/{{ LocationID }}" method="POST" >  <div class="row"> {{ LocationID }}</div>  <div class="row">  <label>Name </label> <input name="Name" type="text"/>  </div>  <div class="row">  <label>NRIC </label> <input name="NRIC" type="text"/>  </div>  <div class="row">  <label> Contact Number </label> <input name="Contact" type="text" />  </div>  <div class="row">  <button type="submit" /> Check In </button>  <button type="reset" /> Cancel </button>  </div>    </form>  view function for form  @app.route("/<location\_id>")  def ret\_form(location\_id):  return render\_template("checkin.html",LocationID=location\_id) | |
| **4.3** | * decorator with function argument [1] * get curren timestamp [1] * retrieve data from request.form [1] * SQL INSERT STATEENT [1] * return url for checkout with embedded NRIC[1] | 5 |
|  | @app.route("/check\_in/<location\_id>",methods=["POST"])  def check\_in(location\_id): ## return a link with embedded check out id  try:  con = sqlite3.connect("ENTRYDB.DB")  con.row\_factory = sqlite3.Row  today = dt.datetime.now()  date = f"{today:%y%m%d}"  time = f"{today:%H%M}"  nric = request.form['NRIC']  con.execute("INSERT INTO Visitor (NRIC,LocationID,Name,Contact, Date,TimeIn) VALUES(?,?,?,?,?,?)",  ( nric ,location\_id, request.form['Name'], request.form['Contact'], date, time)  )  con.commit()    except sqlite3.Error as ex:  print(ex)  con.close()  return f"Please clicked on link to check out <a href='/check\_out/{location\_id}/{nric}'> Check Out</a>" |  |
| **4.4** | * decorator with function argument [1] * Get current time [1] * UPDATE statement [1] | 3 |
|  | @app.route("/check\_out/<location\_id>/<nric>")  def check\_out(location\_id, nric):  try:  con = sqlite3.connect("ENTRYDB.DB")  con.row\_factory = sqlite3.Row  today = dt.datetime.now()  time = f"{today:%H%M}"  con.execute("UPDATE Visitor SET TimeOut=? WHERE NRIC=? AND LocationID=?",  (time, nric ,location\_id)  )  con.commit()    except sqlite3.Error as ex:  print(ex)  con.close()  return f"{nric} checked out at {location\_id}" |  |