Task 1.1

|  |  |
| --- | --- |
|  | [1]  [1]  [1]  [1]   1. or (c) [1]      1. or (d) [1]     (c)  (d)  (a) - (d) [1] |

|  |  |
| --- | --- |
|  | [1]  [1] |

Task 1.2

|  |  |
| --- | --- |
|  | [1]  [1]  [1]  [1]  [1] |
|  | [1] |

|  |
| --- |
| **# Task 2.1, function to create hash address, 3 marks**  def HashKey(Country):  country = Country.lower() # 1 mark  total = 0  for char in country:  total += ord(char) # 1 mark for calculating total  return total % 30 # 1 mark |
| **# Task 2.2, create hash table with text file, 7 marks**  size = 30  HashTable = [''] \* size # 1 mark for initializing array  with open('COUNTRY1.txt', 'r') as f:  for line in f:  country = line.strip()  address = HashKey(country)  if HashTable[address] == '': # 1 mark  HashTable[address] = country  else:  index = address  found = False  tableFull = False    while not found and not tableFull: # 1 mark  if HashTable[index] == '': # 1 mark to insert  found = True  HashTable[index] = country  else:  index += 1  if index == size: # 1 mark for loop back  index = 0  if index == address: # 1 mark for full table  tableFull = True    if tableFull:  print("Table is full!", country, "is not added to the hash table!")  # 1 mark for 5 error message in output |
| **# Task 2.3 hash table search and test, 9 marks**  def searchCountry(HashTable, country):  address = HashKey(country)  found = False  exit = False  index = address    while not found and not exit: # 1 mark  if HashTable[index] == country: # 1 mark for found  found = True  elif HashTable[index] == '': # 1 mark for empty cell  exit = True  else:  index += 1  if index == size: # 1 mark for loop back  index = 0  if index == address: # 1 mark for not found  exit = True    if found:  print(country, "is found at address", index) # 1 mark  print()  else:  print(country, "is not found in the hash table.")  print()    # test cases, 3 mark  # locating by hash address, 'USA'found  searchCountry(HashTable, 'USA')  # locating by hash address, cell with different value, 'Spain' found  searchCountry(HashTable, 'Spain')  # locating by hash address, cell with different value, 'Vietnam' not found  searchCountry(HashTable, 'Vietnam')  Found at Hash Address: USA [29], Russia [3], UK [14], Italy [7], France [23], Germany [5], Turkey [16], Iran [6], Peru [24], Belgium [21], SaudiArabia [2], Pakistan [19], Qatar [27]  Found with collision:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Country | Hash Address | Hash Table Index | Country | Hash Address | Hash Table Index | | Spain | 29 | 0 | Switzerland | 7 | 12 | | Brazil | 14 | 15 | Sweden | 16 | 18 | | India | 7 | 8 | Portugal | 8 | 13 | | China | 5 | 9 | Belarus | 0 | 4 | | Canada | 0 | 1 | Singapore | 8 | 22 | | Mexico | 15 | 17 | Bangladesh | 13 | 25 | | Netherlands | 6 | 10 | Indonesia | 24 | 26 | | Chile | 7 | 11 | Japan | 12 | 28 | | Ecuador | 19 | 20 |  |  |  | |
| **#Task 2.4 Bubble Sort, 9 marks**  # read text file, create list of rates, and  # dictionary of key: death rate, value: [country names]  with open('COUNTRY2.txt', 'r') as f:  rateDict = {}  rateList = []  for line in f:  line = line.strip() # 1 mark for extracting data  country, confirm, death = line.split(',')    confirm = int(confirm)  death = int(death)  rate = round(death / confirm \* 100, 1) # 1 mark    if rate in rateDict: # 1 mark for if-else to create dict  rateDict[rate].append(country)  else:  rateDict[rate] = [country]    if rate not in rateList: # 1 mark for creating list  rateList.append(rate)  # sort list  size = len(rateList)  for i in range(size, 1, -1): # 1 mark for nested loop  for j in range(i - 1):  rate1 = rateList[j]  rate2 = rateList[j + 1]  if rate1 < rate2: # 1 mark for correct condition  rateList[j] = rate2  rateList[j + 1] = rate1 # 1 mark for swap    # 1 mark for generating text file with correct format  # 1 mark for countries with same rate displayed correctly  with open('RATE.txt', 'w') as f:  for rate in rateList:  line = ''  for country in rateDict[rate]:  line = country + ',' + str(rate) + '%\n'  f.write(line) |

|  |
| --- |
| **Task 3.1**  [1M] ProductCode is Primary Key for Product table  [1M] ProductCode is Foreign Key for Other 3 tables  [1M] Correct Data Types  [1M] Rest are all correct  CREATE TABLE "Product" (  "ProductCode" TEXT NOT NULL PRIMARY KEY,  "Name" TEXT,  "Type" TEXT,  "Location" TEXT,  "Price" REAL  );  CREATE TABLE "Cake" (  "ProductCode" TEXT NOT NULL,  "ServingSize" INTEGER,  "Shape" TEXT,  FOREIGN KEY("ProductCode") REFERENCES "Product"("ProductCode")  );  CREATE TABLE "Loaf" (  "ProductCode" TEXT NOT NULL,  "Weight" REAL,  FOREIGN KEY("ProductCode") REFERENCES "Product"("ProductCode")  );  CREATE TABLE "Bun" (  "ProductCode" TEXT NOT NULL,  "PiecesPerPackage" INTEGER,  FOREIGN KEY("ProductCode") REFERENCES "Product"("ProductCode")  ); |
| **Task 3.2**  import sqlite3  import csv  try:  conn = sqlite3.connect("bakery.db")  cur = conn.cursor()  with open('CAKES.TXT', newline='') as csvfile:  records = csv.reader(csvfile, delimiter=',', quotechar='"')  for row in records:  cur.execute("Insert into Product(productcode, Name, Type, Location, Price) Values(?,?,?,?,?)", (row[0],row[1], 'Cake',  row[2], float(row[3])))  cur.execute("Insert into Cake(productcode, ServingSize, Shape) Values(?,?,?)", (row[0], row[4], row[5]))  conn.commit()    with open('LOAVES.TXT', newline='') as csvfile:  records = csv.reader(csvfile, delimiter=',', quotechar='"')  for row in records:  cur.execute("Insert into Product(productcode, Name, Type, Location, Price) Values(?,?,?,?,?)", (row[0], row[1], 'Loaf', row[2], float(row[3])))  cur.execute("Insert into Loaf(productcode, Weight) Values(?,?)", (row[0],float(row[4])))  conn.commit()    with open('BUNS.TXT', newline='') as csvfile:  records = csv.reader(csvfile, delimiter=',', quotechar='"')  for row in records:  cur.execute("Insert into Product(productcode, Name, Type, Location, Price) Values(?,?,?,?,?)", (row[0],row[1], 'Bun', row[2], row[3]))  cur.execute("Insert into Bun(productcode, PiecesPerPackage) Values(?,?)", (row[0],float(row[4])))  conn.commit()  conn.close()  except Exception as err:  print('Error: %s' % (str(err)))  finally:  conn.close()  [1M – read file correctly]  [1M – Insert into Product table for all 3 types]  [1M – Insert into Bun – data inserted correctly]  [1M – Insert into Cake – data inserted correctly ]  [1M – Insert into Loaf – data inserted correctly]  [1M – Convert to correct datatype for fields that are real datatype] |
| **Task 3.3**  select p.ProductCode, p.Name,  p.Location, p.Price,  c.ServingSize from Product p inner join cake c on p.productcode = c.productcode  and c.Shape='Circle'  [1 M Correct Columns selected]  [1 M Product Inner Join Cake ]  [1 M inner join on product.productcode = cake.productcode ]  [1 M Where Shape = ‘Circle’] |
| **Task 3.4**   |  | | --- | | **### /templates/index.html**  <!DOCTYPE html>  <html>  <head><title>Bakery</title>  </head>  <body>  <form action="{{ url\_for('index') }}" method="POST">  <p>  Location: <input type="text" value="" name="location"> &nbsp;&nbsp;&nbsp;  <input type="submit">  </p>  </form>  </html>  **[2M – submit form with text and submit button]** |  |  | | --- | | **### /templates/result.html**  <!DOCTYPE html>  <html>  <head><title>Bakery</title>  </head>  <body>  <p>Listing</p>  <table>  <tr><th>Name</th><th>Type</th><th>Price</th><tr>  {% if results|length > 0 %}  {% for item in results %}  <tr>  <td>{{ item[0] }}</td><td>{{ item[1] }}</td><td>  {{ item[2] }}</td></tr>  {% endfor %}  {%else%}  <tr>  <td colspan="6">No Items</td>  </tr>  {%endif%}  </table>  </body>  </html>  **[1M – table html with correct heading]**  **[2M – for loop in template to list records]** |  |  | | --- | | **#### app.py**  import flask, os, sqlite3  from flask import render\_template, request  app = flask.Flask( \_\_name\_\_, static\_folder = './static', template\_folder = './templates')  @app.route('/', methods=['GET', 'POST'])  def index():  if request.method == 'POST':  location = request.form['location']  conn = sqlite3.connect('bakery.db')  cursor = conn.execute("select name, type, price from product where location = ? order by price asc",(location,))  all\_rows = cursor.fetchall()  cursor.close()  conn.close()  return render\_template( 'result.html', results = all\_rows)  elif request.method == 'GET':  return render\_template('index.html')  if \_\_name\_\_ == '\_\_main\_\_':  app.run()  **[1M – correct route ]**  **[1M – Get request to show index.html template]**  **[1M – Post request to show result.html template and pass data over]**  **[2M – connection to db and correct SQL to draw data, close db connection]** | |
|  |

Task 4.1

|  |  |
| --- | --- |
|  | [1]  [1]    [1]  [1]  [1]  [1]      [1] |
|  | [1]  [1]  [1]      [1]        [1]    [1]    [1] |
|  | [1]    [1]    [1]  [1]  [1]      [1] |
|  | [1]    [1] |
|  | [1]    [1]      [1] |
| [1]  [1] |  |
|  | [1]  [1]  [1] |
|  | [1] |
|  | [1]  [1] |