

Task 1.1 [2m]
<pre># Task 1.1 def charToNum(char): num = ord(char) - ord('A') #2 return num</pre>
Task 1.2 [7 m]
<pre># Task 1.2 def encrypt(text, key): # [7 marks] encrypted = [] len_key = len(key) for i in range(len(text)): #1 if text[i] == ' ': newChar = ' ' #1 #if 'A' <= text[i] <= 'Z': else: textnum = charToNum(text[i]) keynum = charToNum(key[i%len_key]) #1 newnum = (textnum + keynum) % 26 #1 newChar = chr(newnum + ord('A')) #1 encrypted.append(newChar) #1 return ''.join(encrypted) #1 print(encrypt("HELLO WORLD", "ABLE"))</pre>
Task 1.3 [3m]
<pre># Task 1.3 infile = open("TASK1.txt",'r') outfile = open("ENCRYPTED.txt",'w') lines = infile.readlines() #1 for line in lines: encrypted = encrypt(line, "JPJC") #1m outfile.write(encrypted + '\n') #1 infile.close() outfile.close() #read from encrypted text file file = open("ENCRYPTED.txt") lines = file.readlines() for line in lines: line = line.strip() print(line) file.close()</pre>

Show output from ENCRYPTED.txt file [1m]

OXAG!SAKUA!CC!WQXCG
YDY!ZJRB!IQKB!OTRSJAG
BJARAXBG!EJTCN!HXG!VDIXT!IXFJN

Task 2.1 [16m]

import random

def createland(rowsize, colsize): # [2m]

```
    grid = [[' ' for i in range(colsize)] for i in range(rowsize)] #1
    grid[0][0] = '*' #1
    return grid
```

def plantCoconuts(grid): # [4m]

```
    count=0
    while count<3: # 0, 1, 2 - 1m
        row = random.randint(0, len(grid)-1) #1
        col = random.randint(0, len(grid[0])-1)
        if grid[row][col] == ' ': #1
            grid[row][col] = 'C' #1
            count += 1
    return grid
```

def display(grid): # [1m]

```
    for row in grid:
        print(' '.join(row))
    print()
```

def move(grid, curr, direct, dist, score): # [9m]

```
    row = curr[0]
    col = curr[1]
```

plot path with *, except if has 'C' [4]

if direct == 'R':

```
    for i in range(dist):
        if grid[row][col+1+i] != 'C':
            grid[row][col+1+i] = '*'
```

elif direct == 'L':

```
    for i in range(dist):
        if grid[row][col-1-i] != 'C':
            grid[row][col-1-i] = '*'
```

elif direct == 'U':

```
    for i in range(dist):
        if grid[row-1-i][col] != 'C':
```

```

        grid[row-1-i][col] = '*'
    elif direct == 'D':
        for i in range(dist):
            if grid[row+1+i][col] != 'C':
                grid[row+1+i][col] = '*'
    # update endpoint [2]
    if direct == 'U':
        row -= dist
    elif direct == 'D':
        row += dist
    elif direct == 'L':
        col -= dist
    elif direct == 'R':
        col += dist

    # eat coconut if endpoint has 'C' [2]
    if grid[row][col] == 'C':
        grid[row][col] = 'E'
        score += 1

    print("(row, col), score =", (row, col), score) # print
    return (grid, (row, col), score) # return tuple [1]

```

Task 2.2 [10 m]

```

# main
grid = createIsland(10, 15) # create island -1m
grid = plantCoconuts(grid) # plant 3 coconut tree -1m
curr = (0,0) # initialise player starting position and score -1m
score = 0
win = False
display(grid) # display grid

for i in range(8): # game loop -1m
    # print player's current location & score -1m
    print("You are at position:", curr, ' Coconut eaten:', score)
    direct, dist = input("Enter your move (L/R/U/D distance):").split(' ') # user input -1m
    dist = int(dist)
    grid, curr, score = move(grid, curr, direct, dist, score) #1
    display(grid)
    if curr == (9,14) and score==3: #1
        win = True
        break

if win: #1
    print("You have won!")
    print("You have reached finish point and eaten 3 coconuts.")
else: #1
    print("Sorry you did not win")
    if curr != (9,14):

```

```

    print("You did not reached the finish point.")
if score<3:
    print("You have not eaten enough coconuts.")

```

Show output of game with a win or lost [1m]

Task 3.1, 3.2, 3.3, 3.4 [4m, 15m, 4m, 4m]

Task 3.1 [4m]

class Node:

```

    def __init__(self, value): #1m
        self.value = value      #1m
        self.left = None        #1m
        self.right = None       #1m

```

Task 3.2 [15m]

class BinaryTree:

```

    def __init__(self, value):
        self.root = Node(value)

```

```

    def pre_order(self, node, result):
        if node:
            result.append(node.value)
            self.pre_order(node.left, result)
            self.pre_order(node.right, result)

```

```

    def pre_order_traversal(self):
        result = []
        self.pre_order(self.root, result)
        return result

```

```

    def helper(self, node, prev): #uses a reversed pre-order traversal
        if node == None:
            return prev

```

```

        prev = self.helper(node.right, prev)
        prev = self.helper(node.left, prev)

```

```

        node.right = prev #node.next
        node.left = None
        prev = node

```

```

        return prev

```

```

    def flatten(self):
        self.helper(self.root, None)

```

Task 3.3 – [4m]

Main Program

```

tree = BinaryTree(1)

```

```
tree.root.left = Node(2)
tree.root.left.left = Node(3)
tree.root.left.right = Node(4)
tree.root.right = Node(5)
tree.root.right.right = Node(6)

print("Pre-order Traversal:", tree.pre_order_traversal())

tree.flatten()
```

Task 3.4 – [4m]

```
current = tree.root
while current:
    print(current.value, end=" -> " if current.right else "\n")
    current = current.right
```

Task 3.4 [4m]

Pre-order Traversal: [1, 2, 3, 4, 5, 6]

1 -> 2 -> 3 -> 4 -> 5 -> 6

Task 4.1 [3m]
<pre># Task 4.1 import sqlite3 conn = sqlite3.connect("Airport.db") # sql statement - 2m query = ''' CREATE TABLE `Flight` (`flightNum` TEXT, `departure` TEXT, `destination` TEXT, `departureTime` TEXT, `arrivalTime` TEXT, PRIMARY KEY(`flightNum`))''' conn.execute(query) conn.commit() conn.close() # execute and conn - 1m</pre>
Task 4.2 [5 + 1m]
<pre># Task 4.2 import sqlite3 conn = sqlite3.connect("Airport.db") # read from file - 1m infile = open("TASK4.txt",'r') lines = infile.readlines() for line in lines[1:]: line = line.strip().split(',') # strip and split data - 1m # insert into DB - 2m query = '''INSERT INTO Flight VALUES (?, ?, ?, ?, ?)''' conn.execute(query, (line[0], line[1], line[2], line[3], line[4])) conn.commit() conn.close() infile.close() # Commit, close conn, close file - 1m</pre>
Values inserted into Database [1m]

Task 4.3 [3m] # 4.3 from flask import * app = Flask(__name__) @app.route('/') def index(): return render_template("index.html") #1m app.run()
HTML file - Accept any correct implementation (Textbox, hyperlink, etc..) - 2m Menu 1.View all Arrivals 2.View all Departures 3.Query Flight
Task 4.4 [6m] @app.route('/allArrivals') def allArrivals(): conn = sqlite3.connect("Airport.db") query = "SELECT arrivalTime, departure, flightNum \ FROM Flight \ WHERE destination = 'Singapore (SIN)' \ ORDER BY arrivalTime ASC " # SQL statement 2 marks cursor = conn.execute(query) result = cursor.fetchall() # execute sql and fetch result 1m conn.close() return render_template("allArrivals.html", result = result) #1m
All Arrivals <table border=1> <tr> <th>Time</th><th>From</th><th>Flight</th> </tr> {% for flight in result %} <!-- jinja for loop and table 1m --!> <tr> <td>{{ flight[0] }}</td> <td>{{ flight[1] }}</td> <td>{{ flight[2] }}</td> </tr> <!-- jinja values/placeholders 1m --!> {% endfor %} </table>

Output in html - 1 mark**All Arrivals**

Time	From	Flight
00:00	Taipei (TPE)	SQ838
02:15	Mumbai (BOM)	SQ1040
03:30	Beijing (PEK)	SQ818
05:15	Paris (CDG)	SQ848
06:45	Seoul (ICN)	SQ1050
10:15	Tokyo (HND)	SQ1252
12:00	Sydney (SYD)	SQ212
12:45	Sydney (SYD)	SQ1454
13:00	Jakarta (CGK)	SQ303
13:45	Bangkok (BKK)	SQ505
14:15	Dubai (DXB)	SQ1020
14:15	Bangkok (BKK)	SQ1656
15:15	London (LHR)	SQ232
16:00	New Delhi (DEL)	SQ434
16:00	Frankfurt (FRA)	SQ242
16:40	Jakarta (CGK)	SQ1858
18:00	Manila (MNL)	SQ707
18:00	Seoul (ICN)	SQ414
19:45	Amsterdam (AMS)	SQ444
20:00	Perth (PER)	SQ636
20:15	Mumbai (BOM)	SQ616
21:15	Hong Kong (HKG)	SQ909
23:00	Zurich (ZRH)	SQ646

Task 4.5 [2m]

```
@app.route('/allDepartures')
```

```
def allDepartures():
```

```
    conn = sqlite3.connect("Airport.db")
```

```
    query = "SELECT departureTime, destination, flightNum \
```

```
    FROM Flight \
```

```
    WHERE departure = 'Singapore (SIN)' \
```

```
    ORDER BY departureTime ASC " # sql statement 1 m
```

```
    cursor = conn.execute(query)
```

```
    result = cursor.fetchall()
```

```
    conn.close()
```

```
    return render_template("allDepartures.html", result = result)
```

```
<h1>All Departures</h1>
```

```
<table border=1>
```

```
<tr> <th>Time</th><th>To</th><th>Flight</th>
```

```
</tr>
```

```
{% for flight in result %}
```

```
<tr> <td>{{ flight[0] }}</td>
```

```
<td>{{ flight[1] }}</td>
```

```
<td>{{ flight[2] }}</td>
```

```
</tr> <!-- jinja values/placeholders 1m --!>
```

```
{% endfor %}
```

```
</table>
```

Output in html - 1 mark**All Departures**

Time	To	Flight
04:45	Tokyo (HND)	SQ1151
05:45	London (LHR)	SQ121
06:30	Sydney (SYD)	SQ111
07:00	Dubai (DXB)	SQ919
07:15	Sydney (SYD)	SQ1353
07:30	Frankfurt (FRA)	SQ141
08:00	Kuala Lumpur (KUL)	SQ101
09:30	Jakarta (CGK)	SQ202
09:45	Bangkok (BKK)	SQ1555
10:30	New Delhi (DEL)	SQ333
11:15	Seoul (ICN)	SQ313
11:45	Amsterdam (AMS)	SQ343
12:15	Bangkok (BKK)	SQ404
12:15	Perth (PER)	SQ535
12:15	Jakarta (CGK)	SQ1757
12:30	Mumbai (BOM)	SQ515
12:45	Zurich (ZRH)	SQ545
14:30	Manila (MNL)	SQ606
17:45	Hong Kong (HKG)	SQ808
18:20	Taipei (TPE)	SQ737
19:00	Tokyo (HND)	SQ1010
20:00	Paris (CDG)	SQ747
21:30	Beijing (PEK)	SQ717
21:30	Mumbai (BOM)	SQ939
23:30	Seoul (ICN)	SQ949

Task 4.6 [5m]

```

@app.route('/queryFlight')
def queryFlight():
    return render_template("flightQuery.html")

@app.route('/query', methods=["POST"])
def query():
    num = request.form["flightNum"]
    print(num)

    conn = sqlite3.connect("Airport.db")

    query = "SELECT * \
FROM Flight \
WHERE flightNum = ? " # correct sql statement 1m

    cursor = conn.execute(query, (num,))
    result = cursor.fetchall()
    conn.close()
    return render_template("queryResult.html", result=result)

```

```
<h1>Flight Query</h1>
```

```
<form action="/query" method="POST"> <!-- Correct form and components 2m --!>
  <input type="TEXT" name="flightNum">
  <input type="Submit">
</form>
```

```
<h1>Flight Query Result</h1>
<table border=1>
<tr> <th>Flight</th><th>Departure</th><th>Arrival</th><th>Departure Time</th><th>Arrival
Time</th>
</tr>
{% for flight in result %}
<tr>  <td>{{ flight[0] }}</td>
      <td>{{ flight[1] }}</td>
      <td>{{ flight[2] }}</td>
      <td>{{ flight[3] }}</td>
      <td>{{ flight[4] }}</td> <!-- jinja values/placeholders 1m --!>
</tr>
{% endfor %}
</table>
```

Output html - 1m

Flight Query Result

Flight	Departure	Arrival	Departure Time	Arrival Time
SQ111	Singapore (SIN)	Sydney (SYD)	06:30	10:45