YISHUN INNOVA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION
Higher 2

CANDIDATE NAME			
CG		INDEX NO	

COMPUTING

9569/02

Paper 2 (Lab-based)

21 Aug 2024
3 hours

Additional Materials: Removable storage device with the following files:

- `ADDRESSES.TXT`
- `QUICKSORT.TXT` and `POSTALCODES.TXT`
- `GRID1.TXT` and `GRID2.TXT`
- `Q3 Task 3_8 Game Scenario template.ipynb`
- `module.py`
- Q4 Web App Folder (with `PURCHASE.TXT`, `LOYALTY.db`, templates for the `server.py` and `index.html` files)

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

The number of marks is given in brackets [ ] at the end of each task.
The total number of marks for this paper is **100**.

Instruction to candidates:

Your program code and output for Question 1, 2, 3 and 5 should be saved in a single `.ipynb` and downloaded as

```
Prelim_<your class>_<your name>.ipynb
```

For each of the sub-tasks, add a comment statement, at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example:

```
In [1]:   #Task 1.1
          Program Code

          Output:

In [2]:   #Task 1.2
          Program Code

          Output:

In [3]:   #Task 1.3
          Program Code

          Output:
```

1 The DJB2 String Hashing Algorithm, invented by Daniel J. Bernstein, can be used to find the hash value of a string by implementing the following steps:

(a) initialise a variable `temp` with the value `9619`

(b) for each character in the string,

- **(i)** multiply the `temp` value by `37` and add the ASCII value of the character to it to get the updated `temp` value
- **(ii)** use the updated `temp` value and repeat step **(b)(i)** for subsequent characters in the string

(c) after all the characters in the string have been processed in step **(b)**, divide the updated `temp` value by the size of the hash table and obtain the modulo, return it as the hash value for the string

For example, to find the hash value for the string with characters `'ABC'` with a hash table of size `250`:

Step	Character	ASCII value	Calculation
1	`'A'`	`65`	`temp = 9619 × 37 + 65 = 355968`
2	`'B'`	`66`	`temp = 355968 × 37 + 66 = 13170882`
3	`'C'`	`67`	`temp = 13170882 × 37 + 67 = 487322701`

```
hash value  = temp % table size
            = 487322701 % 250 = 201
```

Hence, hash value of `'ABC'` is `201.`

Task 1.1

Write program code for the function `hash(address)` that takes a string `address`, compute and return the hash value using the DJB2 String Hashing Algorithm for a hash table of size 250.

Using the above example:

```
>>> hash('ABC')        #Output: 201
```

[3]

A home renovation company intends to store all the clients' addresses in a hash table of size 250. The text file `ADDRESSES.TXT` contains the addresses of 200 clients.

Task 1.2

Write program code to:

- initialise a hash table of size 250
- read the data in the text file `ADDRESSES.TXT`
- use the function `hash(address)` written in Task 1.1 to hash each address
- use the computed hash value as the index position to insert the address into the hash table
- for addresses that are inserted into the hash table due to collisions, store them in a list named `uninserted`
- print the first 10 rows of the hash table
- count and print the number of addresses that are kept in the list `uninserted.`

[5]

Closed hashing, also known as Linear Probing, can be used to resolve collisions in a hash table.

Task 1.3

Modify the program code written in Task 1.2 to include the implementation of closed hashing (i.e. Linear Probing) to resolve the collisions.

Print the last 10 rows of the updated hash table. [2]

Task 1.4

Write program code for the function `hash_search(tbl,addr)` that searches a client's address `addr` in the hash table `tbl` and returns `True` if the address is found; Otherwise, returns `False`.

Test your program with 2 test cases. [4]

2 The following pseudocode is an implementation of the In-Place Quicksort algorithm which sorts the elements in a one-dimensional list `seq` in ascending order and returns the mutated list `seq`.

The first element in the list `seq` is assigned with index `0`.

This following pseudocode is available in the text file `QUICKSORT.TXT`

```
FUNCTION QuickSort(seq)
  QuickSortHelper(seq, 0, LENGTH(seq) - 1)
  RETURN seq
ENDFUNCTION

FUNCTION QuickSortHelper(seq, start, end)
  IF start < end
    THEN
      mid ← Partition(seq, start, end)
      QuickSortHelper(seq, start, mid - 1)
      QuickSortHelper(seq, mid + 1, end)
  ENDIF
  RETURN seq
ENDFUNCTION

FUNCTION Partition(seq, start, end)
  pivotValue ← seq[start]
  lo ← start + 1
  hi ← end
  done ← FALSE

  WHILE (done = FALSE)
    WHILE lo <= hi AND seq[lo] <= pivotValue
      <A> replace this line with your program code
    ENDWHILE

    WHILE seq[hi] >= pivotValue AND hi >= lo
      <B> replace this line with your program code
    ENDWHILE

    IF hi < lo
      THEN
        done ← TRUE
    ELSE
      temp ← seq[lo]
      seq[lo] ← seq[hi]
      seq[hi] ← temp
    ENDIF
  ENDWHILE

  <C> swap seq[start] with seq[hi]
```

```
42    RETURN hi
43  ENDFUNCTION
```

Task 2.1

Complete the pseudocode by writing the missing codes for line 24, 28 and 41. Label your answers as `<A>`, `<B>` and `<C>`.

Write Python program code for the function `quicksort(lst)` to implement the given Quicksort algorithm to arrange the elements in the list `lst` in ascending order. [4]

Task 2.2

Write program code to:

- read all the postal codes provided in the text file `POSTALCODES.TXT` and store it in a list
- use the function written in Task 2.1 to arrange the postal codes in ascending order
- print the first 5 postal codes in the sorted list. [3]

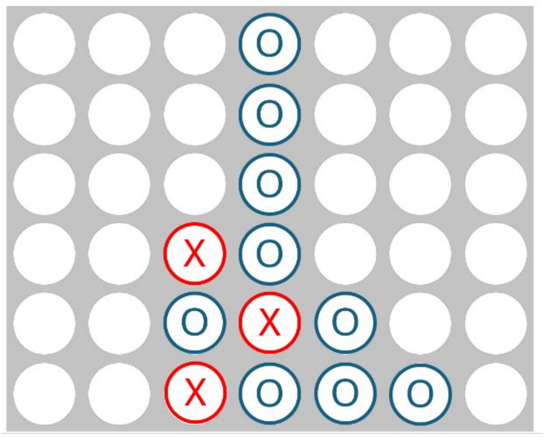
Task 2.3

Write program code for the function `insertion_sort(lst)` to sort the elements in the list `lst` using the insertion sort algorithm and return the mutated list with the elements arranged in descending order.

Use the function `insertion_sort(lst)` to arrange the postal codes read in Task 2.2 in descending order.

Write the sorted postal codes into a text file.
Save the file as `TASK2_3_<your name>.TXT` [4]

3 Connect 4 is a game played by two players. One player uses the red (`X`) tokens and the other uses blue (`O`). The game board is a vertical grid of six rows and seven columns.



The players take turns to choose a column to drop a token into and it will occupy the lowest empty position in the chosen column. The winner is the player who is the first to connect four of their own tokens in a horizontal, vertical or diagonal line.

If all tokens have been used and neither player has connected four tokens, the game ends in a draw.

Task 3.1

Write program code to read the initial state of the grid as given in the text file `GRID1.TXT` and store it in a suitable data structure named `grid1`. [2]

Task 3.2

Write program code for the function `print_it(array)` to print the grid for the array, using '`.`' to represent the empty positions.

A sample of the display of `grid1` is as follows:

```
. . . O . . .
. . . O . . .
. . . O . . .
. . X O . . .
. . O X O . .
. . X O O O .
```

[2]

Task 3.3

Write program code for the function `user_input()` to prompt the player to input the column number, from 1 to 7, which he intends to drop the token into the grid.

The program code should validate and return the player's input. [2]

Task 3.4

Write program code for the function `insertO(array, col)` to:

- accept the player's chosen column `col` and insert the player's token (`O`) to the lowest position of the chosen column
- return the mutated array with the newly inserted token (`O`).

If the column is filled, display the message "`Player (O) missed a turn.`'.

For example,

```
>>> insertO(grid1, 2)
>>> print_it(grid1)

. . . O . . .
. . . O . . .
. . . O . . .
. . X O . . .
. . O X O . .
. O X O O O .
```

```
>>> insertO(grid1, 3)
>>> print_it(grid1)

. . . O . . .
. . . O . . .
. . O O . . .
. . X O . . .
. . O X O . .
. O X O O O .
```

[4]

Task 3.5

Write program code for the function `insertX(array, col)` similar to Task 3.4 to:

- accept another player's choice of column `col` to insert the player's token (`X`) into the lowest position of the chosen column
- return the mutated array with the newly inserted token (`X`). [1]

Task 3.6

Write program code for the function `checkrow(array)` to return `True` if there are four consecutive '`O`' or '`X`' found in any row; Otherwise, return `False`.

The function should display all the elements found in the row with the four consecutive '`O`' or '`X`'. [3]

Task 3.7

Write program code for the function `checkcolumn(array)` to return `True` if there are four consecutive '`O`' or '`X`' found in any column; Otherwise, return `False`.

The function should display all the elements found in the column with the four consecutive '`O`' or '`X`'. [3]

The program code for the function `checkdiagonal(array)` has been provided in the template `Q3 Task 3_8 Game Scenario template.ipynb`. It will return `True` if there are four consecutive '`O`' or '`X`' found in any diagonal; Otherwise, it will return `False`.

Task 3.8

Write program code for a player to play against the computer, taking turns to place their tokens '`O`' and '`X`' respectively. The computer will randomly choose a column to drop its token. Your program should start with an empty grid using the text file `GRID2.TXT` and display the grid after each token has been inserted.

Terminate the game and declare the winner when there are four tokens connected.

Display an appropriate message if all the columns are filled with no winner. [8]

4 A particular bank is the appointed sole distributor for some prestigious concerts and members under the loyalty programme can access its website to book the tickets to these concerts. The bank uses a relational database to store all the data.

The database `LOYALTY.db` has the following tables:

`Member:`

- `Email` – unique email address for each member
- `Password` – password for member's account
- `MemberName` – the name of the member

`Concert:`

- `ConcertID` – unique code for the concert, for example C01
- `ConcertName` – the name of the concert
- `Artiste` – the name of the artiste
- `Date` – the date of the concert in DDMMYY format, for example, 310124
- `Price` – the cost of each concert ticket
- `Quantity` – the current number of available tickets

`Purchase:`

- `PurchaseID` – unique integer number for each purchase, auto generated
- `ConcertID` – the unique concert code
- `Email` – the unique email address of the member
- `Quantity` – the number of tickets purchased

The three tables have been created in the database and the data for tables `Concert` and `Member` have been included.

Task 4.1
Write Python program code to insert the records into the table `Purchase` using the data from the text file `PURCHASE.TXT`.

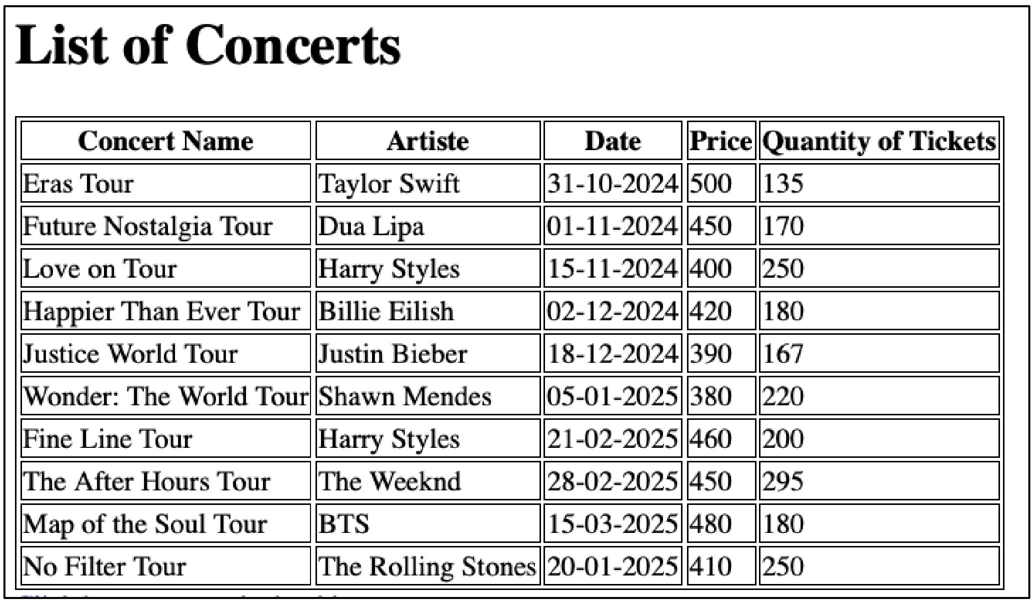
Save your program code as `Task4_1_<your name>.py`. [4]

When a user accesses the website, the following concert information will be displayed on the default webpage:

- concert name
- artiste
- date
- price of a ticket
- number of currently available tickets

Task 4.2

Write program code for the default route in the server script `server.py` to render a template `index.html` to display a table showing the concert information, similar to the one shown below:



List of Concerts

Concert Name	Artiste	Date	Price	Quantity of Tickets
Eras Tour	Taylor Swift	31-10-2024	500	135
Future Nostalgia Tour	Dua Lipa	01-11-2024	450	170
Love on Tour	Harry Styles	15-11-2024	400	250
Happier Than Ever Tour	Billie Eilish	02-12-2024	420	180
Justice World Tour	Justin Bieber	18-12-2024	390	167
Wonder: The World Tour	Shawn Mendes	05-01-2025	380	220
Fine Line Tour	Harry Styles	21-02-2025	460	200
The After Hours Tour	The Weeknd	28-02-2025	450	295
Map of the Soul Tour	BTS	15-03-2025	480	180
No Filter Tour	The Rolling Stones	20-01-2025	410	250

Save your Python program code with any additional files / subfolders in a folder named:

`Task4_2_<your name>` [5]

Run the web application and save the output of the `index.html` as:

`Task4_2_<your name>.html` [1]

Task 4.3

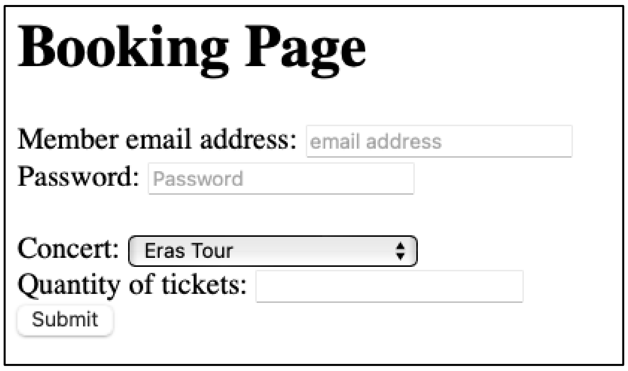
Members can purchase tickets to different concerts. For each transaction, a member needs to:

- enter his registered email address,
- enter his password,
- select one concert, and
- enter any number of tickets to purchase.

Modify the template `index.html` to include a hyperlink for the user to access the `'/booking'` route in the server script to render the template `booking.html` to display a form to allow the user to perform the following:

- enter the member's email address and password,
- select a concert from a dropdown list,
- enter the number of tickets to purchase, and
- submit the form to the `'/check'` route in the server script.

An example of the form is shown below:


Booking Page

Member email address: email address
Password: Password

Concert: Eras Tour
Quantity of tickets:
Submit

Save your Python program code with any additional files / subfolders in a folder named:

`Task4_3_<your name>` [6]

Task 4.4

Write program code for the `'/check'` route in the server script to:

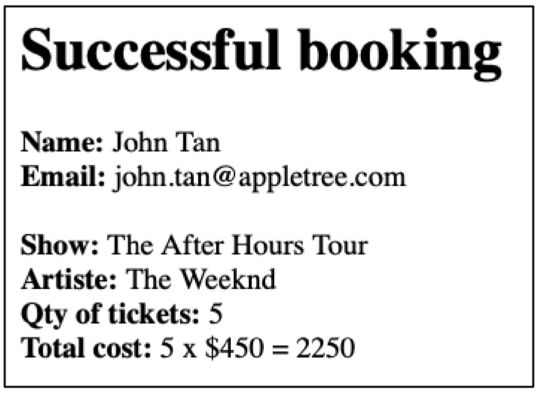
- read the member's email address, password, and booking details from the submitted form
- validate the member's email and password using the data in the table `Member`.
    - if the login information is invalid, render a template to display the error message:

      `'Invalid account.'`

If the login is successful,

- append a new record to the table `Purchase` and update the number of currently available tickets in the table `Concert`
- render a template `success.html` to display:
    - the member's name and email address
    - the concert's details, and
    - the total price for the tickets purchased.

An example is shown below:



Successful booking

Name: John Tan
Email: john.tan@appletree.com

Show: The After Hours Tour
Artiste: The Weeknd
Qty of tickets: 5
Total cost: 5 x $450 = 2250

Save your Python program code with any additional files/folders in a folder named:

`Task4_4_<your name>` [9]

5 A fast-food restaurant *Mac Burger* intends to use a linked list queue implemented with Object-Oriented Programming (OOP) to submit customer's orders to the kitchen for food preparation.

The queue consists of nodes linked from the front to the end of the queue. Each node contains the data and a pointer linking to the next node.

Task 5.1

Write program code for the class `Node` with the data and pointer attributes, and the necessary constructor, getter and setter methods. [2]

Task 5.2

Write program code for the class `Queue` with a root pointer attribute and the following methods:

- a constructor to create an empty queue with the root pointer set to `None`
- `enqueue(new_data)` to insert a node containing `new_data` at the end of the queue
- `dequeue()` to remove a node at the front of the queue and return the data stored in the removed node
- `peek()` to return the data stored in the node at the front of the queue
- `display()` to display all the data sequentially as stored in the queue.

The methods should display an appropriate message if the queue is empty. [8]

The data of an order is a tuple consisting of an Order Number and a list of items ordered.

The following orders were received and sent to the kitchen:

```
order1 = ('2342', ['Fish Burger', 'Apple Pie', 'Coke Zero'])
order2 = ('3534', ['Chicken Burger', 'French Flies'])
order3 = ('8574', ['Mocha Frappe', 'Strawberry Sundae'])
```

Task 5.3

Write program code to create an empty queue `MacQ` and store the three orders into it. Test all the methods in the class `Queue`. [5]

The Mac Burger restaurant issues reward points to the customers for their purchases, and these points will expire at the end of four weeks.

A customer has been issued with the following reward points for the three most recent purchases:

```
reward1 = (80, '2024/7/18')    #expiry: '2024/8/15'
reward2 = (100, '2024/7/26')   #expiry: '2024/8/23'
reward3 = (50, '2024/8/5')     #expiry: '2024/9/2'
```

Task 5.4

Write program code to create an empty queue `RewardQ` and store all the three rewards into it. [1]

Use the function `expirydate(string)` provided in the `module.py` to compute the expiry date based on the issue date.

For example,

```
    >>> expirydate('2024/7/18')    #output: 2024-08-15
```

Another function `convert(string)` is provided in the module to convert the date in the string to the `datetime` type.

Task 5.5

Modify the class `Queue` written in Task 5.2 to include a method `update(string)` to remove all the nodes in the queue with expired reward points (issued more than four weeks before the date specified in `string`) using the '`yyyy/mm/dd`' format. [3]

Test your program with `update('2024/8/21')` and show the remaining data in the queue `RewardQ`. [1]

Task 5.6

Modify the class `Queue` written in Task 5.2 and Task 5.5 to include a method `use(n)` to deduct `n` number of points from the available reward points, using those with earlier expiry dates first.

If there insufficient reward points available, display the following message:

'`There is insufficient reward points,` *`k`* `points not deducted.`',

where *k* is the balance number of points not deducted when the queue `RewardQ` is empty. [4]

Test your program with three consecutive deductions, `use(50)`, `use(75)` and `use(30)`, showing the data in the queue `RewardQ` after each deduction. [1]