HWA CHONG INSTITUTION

C2 PRELIMINARY EXAMINATION 2025

COMPUTING

Higher 2

25 AUG 2025 **Paper 2 (9569 / 02)** **1400 – 1700 hrs**

Additional Materials:

Electronic version of `APPOITMENT.txt` data file

Electronic version of `CUSTOMER.txt` data file

Electronic version of `STYLIST.txt` data file

Electronic version of `WISHLIST.txt` data file

Insert Quick Reference Guide

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.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

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

Instruction to candidates:

Your program code and output for each of Task 1 to 4.3 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code and output for Task 1 should be saved as:

`TASK1_<your name>_<centre number>_<index number>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

The school is exploring an online system to manage student administrative matters.

A student account is formed by 6 digits and 1 lower case letter. The last letter is the check digit, and is calculated using the following algorithm based on the 6 digits:

1. Starting from the left, the first digit has position number 1.
2. For digits at odd numbered positions, it carries a weight of the position number.
   For digits at even numbered positions, it carries a weight double the position number.
3. Calculate the sum of the products of each digit multiplied by its respective weight.
4. Calculate the remainder obtained when the sum is divided by 26.
5. Convert the remainder to the check digit using the conversion table below:

Remainder	0	1	2	…	24	25
Check Digit	a	b	c	…	y	z

For example, given the 6 digits 987654:

Step 1:

Digit	9	8	7	6	5	4
Position	1	2	3	4	5	6

Step 2:

Weight	1	4	3	8	5	12

Step 3: $9 \times 1 + 8 \times 4 + 7 \times 3 + 6 \times 8 + 5 \times 5 + 4 \times 12 = 183$

Step 4: Remainder = 1

Step 5: check digit = b

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:

Task 1.1

A full student account can be validated by calculating the check digit from the first 6 digits and comparing it to the last letter. Write a function `task1_1(account)` that determines if a student account is valid.

The function should:

- check that the parameter `account` consists of 6 digits and 1 lower case letter. Output appropriate messages if the format is invalid.
- calculate the check digit and output appropriate messages indicating if the account is valid.

Test the function fully with suitable test data.

[8]

Students must create a password for their account and the strength rating of a password is calculated based on these rules:

- Add 1 point for every 2 characters, up to a maximum of 4 points
- Add 1 point if the password contains any uppercase letters
- Add 1 point if the password contains any lowercase letters
- Add 1 point if the password contains any digits
- Add 1 point if the password contains any special characters, e.g. @, #
- Deduct 1 point if the password contains any repeated consecutive characters, e.g. 'aa', '111'

For example, 'HwaChong#25' adds 4 points based on the length, 1 point each for containing both lower- and upper-case letters, digits and special characters. The total rating is 8.

The strength of a password is considered weak if the rating is lower than 3, medium if the rating is between 3 and 5, and strong if the rating is higher than 5. For example, 'HwaChong#25' is a strong password.

Task 1.2

Write a function `task1_2(password)` that takes the parameter `password` and outputs its strength.

Test your function using the following three calls:

```
task1_2('cp')
task1_2('Password')
task1_2('HwaChong#25')
```
[8]

Passwords are encrypted before storing for safety reasons. The encryption shifts letters and digits forward by a step size but does not change the special characters. Here is an example on a step size of 9:

- The letter ‘H’ is forwarded 9 steps and encrypted as ‘Q’.
- The letter ‘c’ is forwarded 9 steps and encrypted as 'l’.
- When an uppercase letter goes beyond ‘Z’ it returns to ‘A’. For example, ‘Z’ is encrypted as ‘I’.
- When a lowercase letter goes beyond ‘z’ it returns to ‘a’. For example, ‘w’ is encrypted as ‘f’.
- The digit 0 is forwarded 9 steps and encrypted as 9.
- When a digit goes beyond 9 it returns to 0. For example, 2 is encrypted as 1.

For example, ‘HwaChong#25’ will be encrypted as 'QfjLqxwp#14'.

Task 1.3

Write a function `task1_3(password, size)` that takes the `password` and the step `size` as parameters and returns the encrypted message.

Test your function using the following code:

```
task1_3('HwaChong#25', 9) == 'QfjLqxwp#14'
```
[6]

Save your Jupyter Notebook for Task 1.

```
# Task 1.1
def task1_1(account):
    if len(account) != 7: # check if length is 7
        print('Invalid format! Length must be 7!')
    elif not account[-1].islower():  # check lowercase letter
        print('Invalid format! The last character must be a lower case letter!')
    elif not account[:-1].isdigit(): # check six digits
        print('Invalid format! First six characters must be digits!')
    else:
        total = 0
        for index in range(6):
            if index % 2 == 0: # weights differ by index
                total += int(account[index]) * (index + 1)
            else:
                total += int(account[index]) * (index + 1) * 2
        remainder = total % 26

        # convert remainder to check digit and output messages
        letters = 'abcdefghijklmnopqrstuvwxyz'
        if account[-1] == letters[remainder]:
            print('Valid account!')
        else:
            print('Invalid account! Wrong check digit!')
# invalid format
task1_1('987B65b')
task1_1('987654B')
task1_1('98765b')
# correct and wrong check digit
task1_1('987654b')
task1_1('987654c')
```

```
Invalid format! First six characters must be digits!
Invalid format! The last character must be a lower case letter!
Invalid format! Length must be 7!
Valid account!
Invalid account! Wrong check digit!
```

```
# Task 1.2
def task1_2(password):
    rating = min(4, len(password) // 2) # points by length

    upper = False
    lower = False
    digit = False
    special = False
    repeat = False
    pre_char = ''
    for char in password:
        # check if the password contains certain characters
        if char.isupper():
            upper = True
        elif char.islower():
            lower = True
        elif char.isdigit():
```

```
            digit = True
        else:
            special = True
        if char == pre_char: # check repeat
            repeat = True
        pre_char = char

    # calculate rating
    rating = rating + upper + lower + digit + special - repeat

    # output strength
    if rating < 3:
        print('Weak')
    elif rating <= 5:
        print('Medium')
    else:
        print('Strong')
task1_2('cp')
task1_2('Password')
task1_2('HwaChong#25')
```

```
Weak
Medium
Strong
```

```
# Task 1.3
def task1_3(password, size):
    encrypted = ''
    for char in password:
        if char.isdigit(): # forward digits
            new_digit = (int(char) + size) % 10
            encrypted += str(new_digit)
        elif char.isupper(): # forward uppercase
            new_char = chr((ord(char) - ord('A') + size) % 26 + ord('A'))
            encrypted += new_char
        elif char.islower(): # forward lowercase
            new_char = chr((ord(char) - ord('a') + size) % 26 + ord('a'))
            encrypted += new_char
        else: # no change to special character
            encrypted += char
    return encrypted
task1_3('HwaChong#25', 9) == 'QfjLqxwp#14'
```

2 Name your Jupyter Notebook as:

`TASK2_<your name>_<centre number>_<index number>.ipynb`

An e-commerce platform offers a discount voucher of a fixed amount when a customer spends at least a specified threshold in single purchase. The customer adds items to a wish list with each item represented by a name and a price.

The task is to implement a program that assists the customer by providing a sorted list of items from the wish list that meet or exceed the voucher's minimum spend.

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 2.1
Program Code
```

Output:

Task 2.1

Write a function `task2_1(filename)` that:

- reads a text file `filename`
- stores the records as a list of tuples (name, price)
- returns the list

Display the contents in the file with the following code:

```
print(task2_1('WISHLIST.txt'))
```
[4]

Task 2.2

Write a function `task2_2(lst)` that implements the **quicksort algorithm** which sorts `lst`, a list of tuples, in ascending order using the second element of each tuple. The function returns a sorted list.

Test and display the sorted list with the following code:

```
lst = [('a',6), ('b',2), ('c',5), ('d',7), ('e',1)]
print(task2_2(lst))
```
[8]

Task 2.3

Write a function `task2_3(lst, target)` that implements an adapted **binary search algorithm** that returns the index of the cheapest item in `lst` with a price equal to or more than `target`. You may assume that `target` is equal to or less than the largest price in `lst`.

Test and display the returned value with the following code:

```
lst = [('e',1), ('b',2), ('c',5), ('a',6), ('d',7)]
target = 3
print(task2_3(lst, target))
```

[6]

Task 2.4

Write a function `generate_Items(lst, item, threshold)` that generates a list of items from the wish list that meet or exceed a given `threshold`, based on `item` which the customer intends to buy. You may assume that `threshold` is greater than the price of `item`.

The function needs to:

- compute `target`, which is the difference between `threshold` and price of `item`
- call `task2_3()` to obtain the index of the cheapest item with price equal to or more than `target`
- return the list of items that allow the customer to meet the minimum spend [2]

Task 2.5

Write code to:

- read the file `WISHLIST.txt` to obtain a list of tuples containing the names and prices of items.
- sort the list in ascending order by the price of the items
- assign the third item in the list as the item the user intend to purchase and remove it from the list
- generate and display the list of possible items to purchase with a threshold of 140

[4]

Save your Jupyter Notebook for Task 2.

```
## Task2.1
def task2_1(filename):
    items=[]
    f = open(filename)
    for line in f:
        name, price = line.split(',')
        items.append((name, int(price.strip())))
    f.close()
    return items
```

```
## Task2.1 Test
print(task2_1('WISHLIST.txt'))
```

```
[('Wireless Mouse', 45), ('Keyboard', 60), ('Monitor', 120), ('USB
Drive', 30), ('Laptop', 150), ('Printer', 110), ('Webcam', 80)]
```

```
def partition(lst, low, high):
    pivot=lst[low]
    left = low+1
    right = high
    while left<=right:
        while left <= right and lst[left][1] <= pivot[1]:
           left += 1     ## move right
        while lst[right][1] > pivot[1]:
            right -= 1    # move left
        if left<right:
            lst[left], lst[right] = lst[right], lst[left]
    ## out of loop - left>high
    lst[low], lst[right] = lst[right], lst[low]
    return right

def quickSort(lst, low, high):
    if low < high:
        pos = partition(lst , low, high)  ## partition
        quickSort(lst, low, pos-1)
        quickSort(lst, pos+1, high)


def task2_2(lst):  ##implement a quick sort algorithm
    quickSort(lst,0,len(lst)-1) ## quickSort(A, low, high):
    return lst
```

```
## Task2.2 Test
lst=[('a',6), ('b',2), ('c',5), ('d',7), ('e',1)]
print(task2_2(lst))
```

```
## Task2.2 Test Output
[('e', 1), ('b', 2), ('c', 5), ('a', 6), ('d', 7)]
```

```
## Task2.3
def task2_3(lst, target): ## adapted binary search algorithm
    low=0
```

```
    high=len(lst) - 1
    idx=len(lst)-1    ## index to return
    while low<=high:   # loop
        mid=(low + high)//2    #assign mid
        if lst[mid][1] >= target:
            idx=mid    ## store the best index
            high = mid - 1
        else:
            low = mid + 1        #go to upper sublist
    return idx
```

```
## Task2.3 Test
lst=[('e',1), ('b',2), ('c',5), ('a',6), ('d',7)]
target=3
print(task2_3(lst, target))
```

```
## Task2.3 Test outout
2
```

```
## Task 2.4
def generate_Item(lst, item, threshold):
    target = threshold - int(item[1])
    index = task2_3(lst, target)
    nextItems = []
    for i in range(index, len(lst)):
        nextItems.append(lst[i])
    return nextItems
```

```
## Task 2.5

lst= task2_1('WISHLIST.txt')
task2_2(lst)

item = lst.pop(2)

nextItems = generate_Item(lst, item, 140)
for nItem in nextItems:
    print(f"Name: {nItem[0]}, Price: {nItem[1]}")
```

```
## Task 2.5 Test output

Name: Webcam, Price: 80
Name: Printer, Price: 110
Name: Monitor, Price: 120
Name: Laptop, Price: 150
```

3 Name your Jupyter Notebook as

`TASK3_<your name>_<centre number>_<index number>.ipynb`

A web browser maintains two stacks to track navigation history:

- A back stack stores pages visited before the current page, that the user can access via the 'back' button
- A forward stack stores pages visited after the current page, that the user can return to via the 'forward' button.

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 3.1
Program Code
```

Output:

Task 3.1

The class `Node` contains three attributes:

- `title`: title of the webpage
- `url`: URL of the webpage
- `next`: reference to the next node

The class `Stack` contains one attribute:

- `top`: reference to the top node in the stack

The class Stack contains the following methods:

- `isEmpty()`: returns `True` if stack is empty and `False` if otherwise
- `push(newnode)`: adds `newnode` to the top of the stack
- `pop()`: removes and returns the top node from the stack, returns `None` if stack is empty

Write the program code for the `Node` class and `Stack` class. [6]

Task 3.2

The class `HistoryCurator` contains three attributes:

- `b_stack`: `Stack` object representing the back stack (initially empty)
- `f_stack`: `Stack` object representing the forward stack (initially empty)
- `current`: `Node` object containing the current page (initially set to `None`)

The class `HistoryCurator` contains the following methods:

- `visit(title, url)`:
  - Pushes the current page to the back stack
  - Clears the forward stack
  - Sets new current page
- `back()`:
  - Pushes the current page to the forward stack
  - Pops top of the back stack and sets it as the current page
  - If the back stack is empty, displays an appropriate error message.
- `forward()`:
  - Pushes the current page to the back stack
  - Pops top of the forward stack and sets it as the current page
  - If the forward stack is empty, displays an appropriate error message.
- `showCurrent()`: Displays the title and URL of the current page

Write the program code for the `HistoryCurator` class. [12]

Task 3.3

Test your program with the following:

1. Create `history`, a `HistoryCurator` object
2. Visit the following pages (title and URL provided below):
   - "Home" – site.com
   - "About" – site.com/about
   - "Products" – site.com/products
   - "Contact" – site.com/contact
3. Call `showCurrent()`
4. Call `back()` twice and `forward()` once, then call `showCurrent()`
5. Visit a new page: "Careers" – site.com/careers
6. Call `forward()` once
7. Call `back()` until the back stack is empty, then show the current page. [6]

Save your Jupyter Notebook for Task 3.

```
#Task 3.1

class Node:
    def __init__(self, title, url):
        self.title = title
        self.url = url
        self.next = None

class Stack:
    def __init__(self):
        self.top = None

    def isEmpty(self):
        return self.top == None

    def push(self, newnode):
        newnode.next = self.top
        self.top = newnode

    def pop(self):
        if self.isEmpty():
            return None
        node = self.top
        self.top = self.top.next
        return node
```

```
#Task 3.2

class HistoryCurator:
    def __init__(self):
        self.b_stack = Stack()  # back history
        self.f_stack = Stack()  # forward
        self.current = None     # current page

    def visit(self, title, url):
        if self.current is not None:
            self.b_stack.push(self.current)
        self.current = Node(title, url)
        self.f_stack = Stack()  # clear forward history

    def back(self):
        if self.b_stack.isEmpty():
            print("No pages in back history.")
            return
        self.f_stack.push(self.current)
        self.current = self.b_stack.pop()

    def forward(self):
        if self.f_stack.isEmpty():
            print("No pages in forward history.")
            return
        self.b_stack.push(self.current)
```

```
        self.current = self.f_stack.pop()

    def showCurrent(self):
        if self.current:
            print(f"Current Page:   {self.current.title:10} -
{self.current.url}")
        else:
            print("No current page.")
```

```
#Task 3.3

history = HistoryCurator()

print("== Start ==")
history.visit("Home", "site.com")
history.visit("About", "site.com/about")
history.visit("Products", "site.com/products")
history.visit("Contact", "site.com/contact")
history.showCurrent()

print("\n== Back/Forward ==")
history.back()
history.back()
history.forward()
history.showCurrent()

print("\n== Visit ==")
history.visit("Careers", "site.com/careers")
history.forward()

print("\n== Back ==")
while not history.b_stack.isEmpty():
    history.back()
history.showCurrent()
```

```
== Start ==
Current Page:   Contact    - site.com/contact

== Back/Forward ==
Current Page:   Products   - site.com/products

== Visit ==
No pages in forward history.

== Back ==
Current Page:   Home       - site.com
```

4 Name your Jupyter Notebook as

`TASK4_<your name>_<centre number>_<index number>.ipynb`

A hair salon manager wants to create a suitable database to store the data of stylists, customers and appointments.

The database `Salon` will have the following tables:

`Customer`:

- `CustomerNo` – unique code for each customer, for example C2
- `CustomerName` – the name of the customer
- `Phone` – the phone number of the customer

`Stylist`:

- `StylistNo` – unique code for each stylist, for example S3
- `StylistName` – the name of the stylist
- `Price` – the price of haircut by the stylist

`Appointment`

- `ApptNo` – unique code for each appointment, for example A4
- `CustomerNo` – the code of the customer booking the appointment
- `StylistNo` – the code of the stylist booked in the appointment
- `Date` – the date of the appointment
- `Status` – the status of the appointment – 'Booked', 'Completed'

For each of the sub-tasks 4.1 to 4.3, 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 4.1
Program Code
```

Output:

Task 4.1

Write a Python program that uses SQL code to create the database `Salon` with the three tables given. Define the primary and foreign keys for each table. [4]

Task 4.2

The text files `CUSTOMER.txt`, `STYLIST.txt`, and `APPOINTMENT.txt` store the comma-separated values for each of the tables in the database. Write a Python program to read data from each file and store it in the appropriate table in the database. [4]

Task 4.3

All the appointments on 25 Aug have been completed, and Ryan Lim cancelled his appointment on 21 Sep. Write a Python program to update the database accordingly, then display all the remaining appointments. The output shows the appointment dates, customer names and stylist names in chronological order starting from the most recent date. [7]

Save your Jupyter Notebook.

Task 4.4

The manager wants to display the stylists' sales records in a web browser. Write a Python program and the necessary files to create a web application that displays a table with each stylist's name, the total number of completed appointments, and the total revenues based on the prices.

Save your Python program as:

`TASK_4_4_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders in a folder named:

`TASK_4_4_<your name>_<centre number>_<index number>`

Run the web application and save the web page output as:

`TASK_4_4_<your name>_<centre number>_<index number>.html` [9]

```
# Task 4.1

import sqlite3
conn = sqlite3.connect('Salon.db')

""" For debugging """
conn.execute("DROP TABLE IF EXISTS Customer")
conn.execute("DROP TABLE IF EXISTS Stylist")
conn.execute("DROP TABLE IF EXISTS Appointment")

# create Customer table with PK
create_Customer = """CREATE TABLE Customer (
    CustomerNo TEXT PRIMARY KEY,
    CustomerName TEXT,
    Phone TEXT);"""
conn.execute(create_Customer)

# create Stylist table with PK
create_Stylist = """CREATE TABLE Stylist (
    StylistNo TEXT PRIMARY KEY,
    StylistName TEXT,
    Price INTEGER);"""
conn.execute(create_Stylist)

# create Appointment table with PK and FK
create_Appointment = """CREATE TABLE Appointment (
    ApptNo TEXT PRIMARY KEY,
    CustomerNo TEXT,
    StylistNo TEXT,
    Date TEXT,
    Status TEXT,
    FOREIGN KEY(CustomerNo) REFERENCES Customer(CustomerNo),
    FOREIGN KEY(StylistNo) REFERENCES Stylist(StylistNo));"""
conn.execute(create_Appointment)

conn.commit()
conn.close()
```

```
# Task 4.2

conn = sqlite3.connect('Salon.db')

# insert data into Customer table
f = open('CUSTOMER.txt', 'r')
for line in f:
    CustomerNo, CustomerName, Phone = line.strip().split(',')
    conn.execute('INSERT INTO Customer (CustomerNo, CustomerName,
Phone) VALUES (?, ?, ?)', (CustomerNo, CustomerName, Phone))

f.close()

# insert data into Stylist table
```

```
f = open('STYLIST.txt', 'r')
for line in f:
    StylistNo, StylistName, Price = line.strip().split(',')
    conn.execute('INSERT INTO Stylist (StylistNo, StylistName, Price)
VALUES (?, ?, ?)', (StylistNo, StylistName, Price))
f.close()

# insert data into Appointment table
f = open('APPOINTMENT.txt', 'r')
for line in f:
    ApptNo, CustomerNo, StylistNo, Date, Status =
line.strip().split(',')
    conn.execute('INSERT INTO Appointment (ApptNo, CustomerNo,
StylistNo, Date, Status) VALUES (?, ?, ?, ?, ?)', (ApptNo,
CustomerNo, StylistNo, Date, Status))
f.close()

conn.commit()
conn.close()
```

```
# Task 4.3

conn = sqlite3.connect('Salon.db')

# update status of all appointments on 25 Aug to completed
conn.execute('UPDATE Appointment SET Status = "Completed" WHERE
Appointment.Date = "20250825" ')
conn.commit()

# find the customer number for Ryan Lim
cursor = conn.execute('SELECT CustomerNo FROM Customer WHERE
CustomerName = "Ryan Lim"')
for row in cursor:
    custNo = row[0]

# delete Ryan Lim’s appointment on Sep 21
conn.execute('DELETE FROM Appointment WHERE CustomerNo = ? and Date =
"20250921"', (custNo, ))
conn.commit()

# appointment date, customer name, stylist name in ascending order of
date
cursor = conn.execute('SELECT Appointment.Date,
Customer.CustomerName, Stylist.StylistName FROM Appointment \
INNER JOIN Customer ON Appointment.CustomerNo = Customer.CustomerNo \
INNER JOIN Stylist ON Appointment.StylistNo = Stylist.StylistNo \
WHERE Appointment.Status = "Booked" ORDER BY Appointment.Date')

print('Date\t\tCustomer Name\t\t\t\tStylist Name')
for row in cursor:
    print(f'{row [0]}\t{row [1]:<25}\t\t{row [2]}')
```

```
conn.close()
# booked only
Date       Customer Name                    Stylist Name
20250826   Aisyah Binte Rahman              Chen Xiao Ting
20250826   Clara Tan                        Amanda Lim
20250826   Jonathan Pereira                 Park Ji Hoon
20250922   Muhammad Hafiz Bin Ismail        Kim Min Ji
20250925   Priya Devi Ramasamy              Marcus Tan
```

```
# Task 4.4
from flask import Flask, render_template, request
import sqlite3

app = Flask(__name__)

@app.route('/')
def home():
    conn = sqlite3.connect('Salon.db')

    # get data for stylists with completed appointments
    data = db.execute("""
            SELECT s.StylistName, COUNT(*), SUM(s.Price)
            FROM Stylist s
            INNER JOIN Appointment a
            ON a.StylistNo = s.StylistNo
            WHERE a.Status = 'Completed'
            GROUP BY a.StylistNo;
        """).fetchall()

    # add in the stylists that do not have any completed appointments
    stylists = db.execute("SELECT StylistName FROM Stylist").fetchall()

    for stylist in stylists:
        stylistName = stylist[0]

        no_completed_appointment = True
        for row in data:
            if row[0] == stylistName:
                no_completed_appointment = False
        if no_completed_appointment:
            data.append([stylistName, 0, 0])

    conn.close()

    return render_template('home.html', results = data)

if __name__ == '__main__':
    app.run()
```

```
# Alternative SQL

    # stylist name, number of completed appointment, total revenue
    cursor = conn.execute('''SELECT StylistName, COUNT(ApptNo),
COUNT(ApptNo)*Price FROM Stylist
LEFT OUTER JOIN (SELECT StylistNo, ApptNo FROM Appointment WHERE
Status = 'Completed') AS CompletedAppointment
ON CompletedAppointment.StylistNo = Stylist.StylistNo
GROUP BY Stylist.StylistNo
ORDER BY COUNT(ApptNo) DESC''')

   results = cursor.fetchall()
```

```
<!--home.html to display the result-->

<!doctype html>
<html>
    <head>
        <title>Stylist</title>
    </head>
    <body>
        <h1>Stylist Revenue</h1>
        <table border>
            <tr>
                <th>Name</th>
                <th>Number of Appointments</th>
                <th>Total Revenue</th>
            </tr>
        {% for item in results %}
            <tr>
                <td>{{ item[0] }}</td>
                <td>{{ item[1] }}</td>
                <td>${{'%0.2f'% item[2] }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Stylist Revenue

Name	Number of Appointments	Total Revenue
Marcus Tan	2	$80.00
Amanda Lim	2	$100.00
Kim Min Ji	2	$180.00
Park Ji Hoon	1	$100.00
Zhang Wei Ming	0	$0.00
Chen Xiao Ting	0	$0.00