YISHUN INNOVA JUNIOR COLLEGE
JC 1 PROMOTIONAL EXAMINATION
in preparation for General Certificate of Education Advanced Level
Higher 2

CANDIDATE NAME

CG | INDEX NUMBER

H2 COMPUTING **9569/02**
Paper 2 (Lab-based) **25 Sep 2019**
(SAMPLE PAPER) **2 hours**

Additional Materials: Removable storage device with the following files :

- `Templates: TASK1.ipynb, TASK3.ipynb, TASK4.ipynb and Q7`
- `EXCHANGE.py, QUEUE.py, module.py`
- `data.csv`

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 question or part question.
The total number of marks for this paper is 100.

At the end of the examination, save all the program codes in the thumb drive provided.

This document consists of **13** printed pages.

 [Turn over

1 The letters in the English alphabet are either vowels (a, e, i, o, u) or consonants.

Task 1

Write a program code `count_vowels(lst)` to:

- take in a list of words and count the number of times each vowel appear
- return the vowels and the frequencies as tuple pairs as shown below.

Input word list:

```
celebrate
journey
strive
appreciate
```

Output:

```
('a', 3), ('e',7), ('i',2), ('o',1), ('u', 1)
```

Download your program code and output, using `word_list`, provided in the template `TASK1.ipynb` as the input, as

`TASK1_<your name>_<index number>.ipynb` [5]

2 In a fantasy game, a player's inventory, `my_bag`, can be represented using a dictionary as shown below:

```
my_bag = {'coin':50,'leather':23,'wood':15,'scroll':2,'ore':13}
```

When the player has successfully killed the monster `barlog`, he can pick up and possess the monster's loot.

```
barlog_loot = ['leather', 'leather', 'scroll', 'wood', 'ore']
```

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:

In [2]: #Task 2.2
        Program code
        Output:
```

Task 2.1

Write a program code for `add_loot(inventory, item_list)` that will add all the items from the monster's loot into his inventory. The program should return the player's upsized inventory.

Test your program using `my_bag` and `barlog_loot` as the inputs. [3]

Task 2.2

The items in the inventory can be exchanged for coins at the shop. The exchange rate is stored in a dictionary provided in `EXCHANGE.py`. The **key** is the item name and the **value** is the number of coins that the item can be exchanged for.

Write a program code for `change_coin(inventory)` that will return a updated inventory where all the items have been exchanged for coins.

Test your program using `my_bag` as an input. [4]

Download your program code and output for Task 2 as

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

3 In a supermarket, the grocery inventory database stores the item names in a hash table for easy search and retrieval.

The hash function uses the ASCII values of the characters in the item name to compute its hash value.

The following is an example of the computation:

`ord('i')= 105  ord('n')= 110  ord('o')= 111`

`hash('onion') = (111+110+105+111+110) % 13 = 547 % 13 = 1`

[Turn over

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:
```

```
In [2]: #Task 3.2
        Program code

        Output:
```

```
In [3]: #Task 3.3
        Program code

        Output:
```

```
In [4]: #Task 3.3
        Program code

        Output:
```

Task 3.1

Write a program code for this hash function.

Test your program using `'onion'` as an input. [2]

Task 3.2

Write a program code `hash_table(seq)` to:

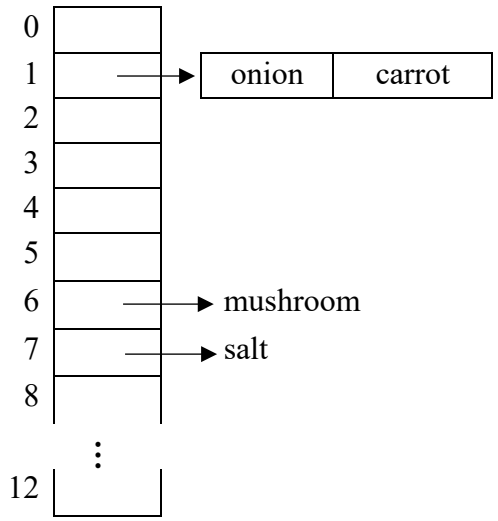
- store the sequence of item names into a hash table of size **13**. You should use the hash function, written in **Task 3.1**, to generate the indices.
- return the hash table

You may assume that there is **no** collision and the hash function will generate different values for different item names.

keys1 and keys2, provided in the template, are sequences containing the name of the items.

Test your program using `keys1`, provided in the template `TASK3.ipynb`, as the input. [4]

Separate Chaining is a strategy to handle the collisions in a hash table.



For example, the following items are to be inserted into the hash table as shown on the left:

name	hash(name)
onion	1
salt	7
mushroom	6
carrot	1

After inserting 'onion', 'salt' and 'mushroom' into their respective positions, a collision occurs when we try to insert 'carrot' as 'onion' is already stored in that location. The **Separate Chaining** strategy overcomes this by creating a list for 'onion' and 'carrot' and store this list at the location in the hash table.

Task 3.3

Modify your program code written in **Task 3.2** to use the Separate Chaining strategy to handle the collisions in the hash table.

Test your program using `keys2`, provided in the template `TASK3.ipynb`, as the input. [4]

Task 3.4

Write a program code `search(item)` to search for an item in the hash table.

It should return `True` if the item is found in the hash table and return `False` otherwise.

Test your program to search for items like '`salt`', '`stone`', '`cake`' and '`potato`'. [4]

Download your program code and output for Task 3 as

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

[Turn over

4 John observed that a car travelling at a higher speed requires a longer distance to stop when the brakes are applied. He hypothesized that the stopping distance is correlated to the speed of the car.

To test his hypothesis, he recorded 50 experimental data of the car travelling at various speeds and their corresponding stopping distances in `data.csv`.

The correlation coefficient, r, between two variables x and y determines how strongly the two variables correlate to each other and is given by:

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \ , \ r \in [-1, 1]$$

where Σ represents the sum and n is the size of the data sample.

A larger magnitude of r indicates a stronger correlation between the two variables.

A sample computation of r for a data set containing 5 observations is as follows:

x	2	4	6	8	10
y	1	2	3	4	5

$\text{sum\_x} = \sum x = 2 + 4 + 6 + 8 + 10 = 30$
$\text{sum\_y} = \sum y = 1 + 2 + 3 + 4 + 5 = 15$
$\text{sum\_x2} = \sum x^2 = 2^2 + 4^2 + 6^2 + 8^2 + 10^2 = 220$
$\text{sum\_y2} = \sum y^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$
$\text{sum\_xy} = \sum xy = (2 \times 1) + (4 \times 2) + (6 \times 3) + (8 \times 4) + (10 \times 5) = 110$

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} = \frac{5(110) - (30)(15)}{\sqrt{(5(220) - 30^2)(5(55) - 15^2)}} = \frac{100}{100} = 1$$

Since r = +1, it implies that x and y are strongly positively correlated.

The file provided are `module.py` and `data.csv`.

The `module.py` contains the following functions:

`read_csv(csv_filename)`: reads the data from a csv file and returns a tuple containing the experimental observations

`get_speed(observation)`: returns the speed in an observation

`get_stopdist(observation)`: returns the stopping distance in an observation

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 4.1
        Program code

        Output:

In [2]: #Task 4.2
        Program code

        Output:
```

Task 4.1

Using the provided template `TASK4.ipynb`, write a program code for `compute_r(table)` to determine the correlation coefficient of the speeds and stopping distances.

Test your program using the data provided in `data.csv`. [5]

On checking some physics books, John learned that the stopping distance is actually directly proportional to the square of the speed.

Task 4.2

Modify your program code written in **Task 4.1** so that it will compute the correlation coefficient between the square of the speeds and the stopping distances.

Test your program using the data provided in `data.csv`. [2]

Download your program code and output for Task 4 as

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

[Turn over

5 A Karaoke player uses a queue data structure to manage the song selections. A queue data structure can be implemented using a list.

The `QUEUE.py` module provided contains the following functions:

- `make_queue()` constructs an empty queue
- `enqueue(q,item)` appends an item to the queue, `q`
- `dequeue(q)` returns and removes the first item in `q`, returns `None` if `q` is empty
- `front(q)` returns the item at the front of `q`
- `size(q)` returns the size of the `q`

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 5.1
        Program code

        Output:
```

```
In [2]: #Task 5.2
        Program code

        Output:
```

Task 5.1

Amend the `QUEUE.py` module by adding the following functions:

- `delete(q,item)` to remove an item from queue, `q`.
- `shift(q,item)` to shift an item to the front of the queue.

Both the functions should return `None` if the item is not found in the queue. [4]

Task 5.2

Write the commands to:

- construct a new empty queue named `playlist`
- add the following songs into the `playlist`
  - ‘I have a dream’
  - ‘I want it that way’
  - ‘Perfect’
  - ‘In my feelings’
  - ‘Blank Space’
- remove the first song from the `playlist`
- display the title of the first song in the `playlist`
- delete the song ‘Blank Space’ from the `playlist`
- shift the song ‘Perfect’ to the front of the `playlist`
- output the size of the `playlist` [4]

Download your program code and output for Task 5 as

`TASK5_<your name>_<index number>.ipynb`

6 Every published book has an International Standard Book Number (ISBN). This ISBN is a 9-digit number plus a check digit which is calculated and added to the end of the number.

For example, the first 9-digit of a typical ISBN is as shown:

075154926_

A weighted-modulus scheme is used to calculate the check digit as shown in the following:

1. Each individual digit in the ISBN is multiplied by 9 fixed numbers: **10, 9, 8, 7, 6, 5, 4, 3, 2** respectively. For **075154926**, we would have

$$0\times\mathbf{10},\ 7\times\mathbf{9},\ 5\times\mathbf{8},\ 1\times\mathbf{7},\ 5\times\mathbf{6},\ 4\times\mathbf{5},\ 9\times\mathbf{4},\ 2\times\mathbf{3},\ 6\times\mathbf{2}$$

2. These products are added up as the total. The total for the above is **214**.
3. The total is then divided by 11 to obtain the remainder, **214** / **11** = **19** remainder **5**.
4. The check digit is the difference between **11** and the remainder mod 11.
   If the calculated check digit is **10**, it will be replaced with the character '**X**'

The check digit for **075154926** is **11** − **5** = **6**. Hence the complete ISBN with the check digit is:

0751549266

For example:

1. Input: '`184146208`' Output: '`184146208X`'
2. Input: '`034085045`' Output: '`0340850450`'

Task 6

Write a program code to calculate the check digit and generate the complete ISBN.

Test your program with **three** data. [6]

Download your program code and output for Task 6 as

`TASK6_<your name>_<index number>.ipynb`

7 A binary search is a technique to search for an item in an ordered dataset.

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 7.1
        Program code

        Output:
```

```
In [2]: #Task 7.2
        Program code

        Output:
```

```
In [3]: #Task 7.3
        Program code

        Output:
```

```
In [4]: #Task 7.4
        Program code

        Output:
```

```
In [5]: #Task 7.5
        Program code

        Output:
```

Task 7.1

Study the identifier table and the incomplete recursive algorithm.

Variable	Data Type	Description
ThisArray	ARRAY OF STRING	Array containing the dataset
FindItem	STRING	Item to be found
Low	INTEGER	Lowest index of the considered list
High	INTEGER	Highest index of the considered list
Middle	INTEGER	The array index for the middle position of the current list considered

```
FUNCTION BinarySearch(ThisArray, FindItem, Low, High)
    IF .............. A ..............
        RETURN 'Not found.' // if item not found
    ELSE
        // calculate new Middle value
        Middle ← ............... B ...............
        IF ThisArray[Middle] > FindItem
            RETURN BinarySearch(ThisArray, FindItem, Low, Middle - 1)
        ELSE
            IF ThisArray[Middle] < FindItem
                 ............... C ...............
            ELSE
                 RETURN Middle // found at position Middle
            ENDIF
        ENDIF
    ENDIF
ENDFUNCTION
```

Write down the missing parts of the algorithm labelled **A**, **B** and **C**. in the template TASK7.ipynb. [3]

Task 7.2

Using the algorithm provided in **Task 7.1**, write the program code for the `binary_search` function. [1]

Task 7.3

Write a program code `find_animal(seq,item)` to:

- take in a sequence and an animal name as inputs
- use the function binary_search to search whether the animal is in the sequence
- return '`Not found`' if animal name is not in sequence, otherwise, return the index position of the animal

Test your program using `MyAnimal`, provided in the template `TASK7.ipynb`, as the input sequence, '`ant`', '`seal`' and '`dinosaur`' as items. [3]

Task 7.4

Another sequence `MyAnimal2`, provided in the template `TASK7.ipynb`, contains names of animals which are unordered.

Write a program code `bubble_sort(seq)` to:

- take in a sequence
- sort the items using bubble sort algorithm

Test your program using `MyAnimal2` as the input. [4]

Task 7.5

Write an additional code such that the bubble sort program will count and display the number of comparisons made when executing the sorting process.

Test the amended bubble sort program using `MyAnimal2` as the input. [2]

Download your program code and output for Task 7 as

`TASK7_<your name>_<index number>.ipynb`