Instructions to candidates:

Your program code and output for each of Task 1 to 4 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as
`Task_1_<your name>_<NRIC number>.ipynb`.
For each sub-task, 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:
```

1 Name and save your Jupyter Notebook as

`Task1_<your name>_<NRIC number>.ipynb`.

A restaurant offers membership for its customers. As a member, the customers are issued a unique ID numbers `IDNumber` which are integers. To help manage the identification of the customers, the manager of the restaurant opted for a low-cost solution of using a one-dimensional fixed-size array to store these ID numbers. The array is first initialized with `None` elements and the elements in this array is indexed from 0 to `(Max-1)`, where `Max` is size of the array.

To determine the index `Index` in the array for the ID numbers `IDNumber`, the manager decided to use the following formula

$$\texttt{Index} \longleftarrow \texttt{IDNumber MOD Max}.$$

Task 1.1

Each line in the text file, `IDNUMBER_NAME1.CSV`, contains the ID numbers of the customers together with the customers' name.

Write program code to:

- Read ID numbers from a text file and store them in the array. For the purpose of testing the program, `Max` is to be set to the value 20. Assume different IDs will have different indices.
- Print out the contents of the array in the order in which the elements are stored in the array.

Use `IDNUMBER_NAME1.CSV` to test your program code. [7]

Task 1.2

In the midst of populating the array, the manager realised that using his formula, some ID numbers could be mapped to the same index, which would lead to customers not being able to uniquely identified.

Amend your program code so that if more than one ID number is mapped to the same index, your program will

- search sequentially from the index that has already contained an ID number for an unused location in the array,
- get the index for this unused location the array, and
- store the duplicate ID Number at this empty location.

Use `IDNUMBER_NAME2.CSV` to test your program code. [4]

Task 1.3

Add code to your Task 1.2 program.

The program is to:

- Take as input an ID number
- Perform a linear search on the array and output the index of the array where the ID number was found.

Use `IDNUMBER_NAME2.CSV` to test your program code.

Run the program three times. Use the following inputs:

80192615, 11559181 and 79626124. [7]

Save your Jupyter Notebook for Task 1.

2 Name and save your Jupyter Notebook as

`Task2_<your name>_<NRIC number>.ipynb`.

A program is to be written to implement a product inventory system of an online book store using object-oriented programming (OOP).

The inventory system is to be implemented using a list.

Each book has a unique id, title and price.

The base class will be called `Book` and is designed as follows:

Book
id : STRING title : STRING price : FLOAT

constructor(id: STRING, t: STRING, p: FLOAT)
set_id(id : STRING)
set_title(s: STRING)
set_price(p: FLOAT)
get_id(): STRING
get_title(): STRING
get_price(): FLOAT
summary()

The `summary()` method returns the id, title and price as a single string.

Task 2.1

Write program code to define the class `Book`. [5]

The books in the inventory can be sorted either alphabetically by title or numerically by price, depending on the user's preference.

Once the preference is decided, a book to be added to the list is compared to the books already in the list to determine its correct position in the list. If the list is empty, it is added to the beginning of the list.

The comparison will use an additional member method,

`compare_with(bk: Book, option: STRING): INTEGER`

This function compares the instance (the item in the list) and the `Book` object passed to it, returning one of the three values:

- -1 if the instance is before the given `Book`
- 0 if the two are equal
- 1 if the instance is after the given `Book`

where `option` is either `p`, for sorting according to price, or `a`, for sorting according to alphabetical order of the title.

Task 2.2

There are four books defined in the text file `TASK2_2.TXT`.

Write program code to:

- create a `Book` object for each of the books defined in the text file
- create an empty list of `Book` objects
- receive user input of ordering preferences: `t` for title, `p` for price
- implement the addtional member method `compare_with(bk: Book, option: STRING): INTEGER`
- add each of the four objects in the text file `TASK2_2.TXT` to its appropriate place in the list based on user preferences
- print out the list contents using the `summary()` method. [13]

An electronic book has a default number of pages while an audio book has a duration.

The `ElectronicBook` and `AudioBook` classes inherit from the Book class, extending them to have a default number of pages and duration, designed as follows:

ElectronicBook : Book
pages : INTEGER

constructor(id: STRING, t: STRING, p: FLOAT, pg: INTEGER)
set_pages(pg : INTEGER)
get_pages() : INTEGER

AudioBook : Book
duration : INTEGER

constructor(id: STRING, t: STRING, p: FLOAT, d: INTEGER)
set_duration(d : INTEGER)
get_duration(): INTEGER

The ElectronicBook and Audio Book classes should extend the `compare_with()` method to ensure that :

- the electronic books are found before the audio books in the list,
- the books are first ordered by ascending title or price, depends on user preference,
- the `summary()` method should also be extended to return the pages or duration and the return values of the base `summary()` method.

Task 2.3

There are nine objects defined in the text file `TASK2_3.TXT`.

Amend your program code to:

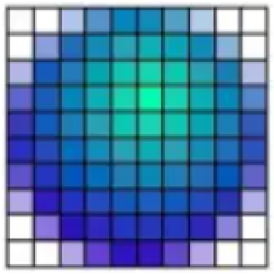
- implement the `ElectronicBook` and `AudioBook` class based on the specification given in the table above,
- implement the extended `compare_with()` method
- implement the extended `summary()` method
- ensure all nine objects in the text file `TASK2_3.TXT` are added to the list
- print out the list contents using the `summary()` method. [12]

Save your Jupyter Notebook for Task 2.

3 Name and save your Jupyter Notebook as

`Task3_<your name>_<NRIC number>.ipynb.`

A picture element, or a **pixel**, is one of the small squares that make up an image on a computer screen. As such, every digital image can be thought of being made up of pixels. The following is an example of an image of the size 10 by 10 pixels. The first 10 refers to the number of rows in the image and the latter 10 refers to the number of columns in the image.



A pixel holds the information of a color and the colors are often represented in a string starting with a hex `#` character followed by 3 pairs of hexadecimal digits. We will call this the **hexadecimal representation** of the pixel. For example, the color white is `#FFFFF` and the color silver is `#C0C0C0`.

Task 3.1

Write a program code with the following specification:

- input a hexadecimal number as a string
- validate the input
- without using the internal Python converter function, calculate the denary value of the hexadecimal number input
- output the denary value. [10]

Each pair of the hexadecimal digits actually represents the intensity of 3 primary colors, red, green and blue, that make up the color in the pixel. As such, instead of having the hexadecimal representation of the pixel, each pixel can be represented by a list of 3 denary values that represents the intensity of the colours, red, green and blue respectively, e.g., the color `#FFC080` can be represented as the list `[255,192,128]`. We will call this representation of the pixel the **RGB representation**.

Task 3.2

A 16 by 13 pixels image is stored as hexadecimal form in the text file `TASK3_2.CSV`, where each line represents a row in the image.

Write a program code with the following specification:

- read the hexadecimal values from the file `TASK3_2.CSV` and store them in a suitable format
- using the program code in **Task 3.1**,
  - convert the image from its hexadecimal pixels representation ito the RGB presentation,
  - write the new presentation into the file `MY_RGB_IMAGE.TXT`. [5]

Conversely, sometimes pixels are given in their RGB representation instead and for graphic designers, hexadecimal values are often preferred.

Task 3.3

A 16 by 13 pixels image is stored its RGB representation in the text file `TASK3_3.CSV`, where each line represents a row in the image.

Write an additional program code with the following specification:

- read the rgb values from the file `TASK3_3.CSV` and store them in a suitable format
- without using the internal Python converter function, convert the denary values in the RGB representation into the hexadecimal representation
- write the new presentation into the file `MY_HEX_IMAGE.TXT`. [12]

Save your Jupyter Notebook for Task 3.

4 Name and save your Jupyter Notebook as

`Task4_<your name>_<NRIC number>.ipynb.`

Monte Carlo simulations is an approach of using a computer's random ability to generate random samples and thus, use the randomness to approximate solutions to problems that might be deterministic in principle.

In probability theory, a **sample space** Ω of an experiment is the *set* of all possible outcomes or results of that experiment and an **event** X is a *subset* of the sample space. For example,

- In the experiment of tossing a double-sided coin, the sample space is $\{\text{Head}, \text{Tail}\}$ and $\{\text{Tail}\}$ is an event.
- In rolling a standard 6-sided dice, the sample space is $\{1, 2, 3, 4, 5, 6\}$ and $\{2, 4, 6\}$ is an event.

The probability of an event X is written as $P(X)$ and is defined by

$$P(X) = \frac{\text{number of elements in } X}{\text{number of elements in } \Omega}.$$

Thus, the value of $P(X)$ is always between 0 and 1.

Task 4.1

30 fair coins, meaning probabilities of getting Heads and Tails are equal i.e $P\left(\text{Head}\right) = P\left(\text{Tail}\right) = \frac{1}{2}$, are flipped and the outcome are recorded. Your task is to approximate the probability of getting exactly 15 heads in those 30 flips.

Write a program code with the following specification:

- generate 100 lists that each represents a random 30 flips of fair coins. If the flip is tails, use 0 to represent it in the list, otherwise, if the flip is heads, use 1.
- find the number of lists that contains exactly 15 heads.
- prints the approximation of the probability of getting exactly 15 heads in when 30 fair coins are flipped. [5]

Task 4.2

We will generalize the experiment by allowing user to specify:

- the probability p of getting Head for each coin flip. This can be achieved with the following pseudocode.

```
FUNCTION MyRandom(p:  FLOAT):
    IF RANDOM(0,1) < p:
    //RANDOM(0,1) returns a FLOAT value between 0 and 1 inclusive
    THEN
        RETURN 1
    ELSE
        RETURN 0
ENDFUNCTION
```

- the number of lists n to be generated,
- number of coin flips c represented by each list,
- number of heads h in the c coin flips.

Write a program code with the following specification:

- the probability p of getting Head for each coin flip is determined by the function `MyRandom` defined in the pseudocode.
- generate n lists that each represents a random c flips of the coins. If the flip is tails, use 0 to represent it in the list, otherwise, if the flip is heads, use 1.
- find the number of lists that contains exactly h heads.
- prints the approximation of the probability of getting exactly h heads in when the c coins are flipped. [4]

Task 4.3

The next task is to display data for a frequency distribution for the simulation above and then output to the screen a horizontal bar chart. The data is generated by modifying your program in Task 4.2.

Write a program code with the following specification:

- create an empty list `data`
- generate n lists that each represents a random c flips of the coins
- for each $0 \leq h \leq c$:
  - find the number of lists n_h that contains exactly h heads,
  - store (h, n_h) as a tuple and add it into the list `data`. As such `data` is the frequency distribution.
- print out the horizontal bar chart similar to the one shown in the following example.

For example, in a run of the program with $n = 195$, $p = \frac{1}{2}$ and $c = 5$, one such possible pairs of values are

$$(0, 13), (1, 21), (2, 45), (3, 56), (4, 39), (5, 11),$$

and the associated horizontal bar chart looks like:

```
++++++++++++++++++++++++++++++++++++++++
Frequency distribution
++++++++++++++++++++++++++++++++++++++++
0 @@@@@@@@@@@@@
1 @@@@@@@@@@@@@@@@@@@@@
2 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
3 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
4 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
5 @@@@@@@@@@@
```

Test your program with $n = 200$, $p = 0.7$ and $c = 10$. [10]

Save your Jupyter Notebook for Task 4.