1 Task 1.1

Write program code for a function `JumpSearch(arr, x, m)` that implements an **iterative** Jump Search. The function returns the integer index where `x` is found in `arr` or `-1` if `x` is not found. You may assume there are no duplicate values stored in `arr`.

Test your function with the following values:

Case	Identifier	Values
1	`arr`	`[ 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 999 ]`
	`x`	`55`
	`m`	`4`
2	`arr`	`[ 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 999 ]`
	`x`	`1000`
	`m`	`4`

10]

Mark	Description
10	`def JumpSearch(arr, x, m):`
	`    #initialise index [1]`
	`    index = 0`
	`    while index < len(arr) and arr[index] < x: #jump [3]`
	`        index += m`
	`    for i in range(index-4, index): #linear search [3]`
	`        if arr[i] == x: #found`
	`            return i`
	`    return -1 #not found [1]`
	`#Main Program`
	`arr = [ 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 999 ]`
	`print(JumpSearch(arr, 55, 4)) #[1]`
	`print(JumpSearch(arr, 1000, 4)) #[1]`

Task 1.2

Write program code for a function `JumpSearchRecursive(arr, x, m, currentLastIndex)` that implements a **recursive** Jump Search. The function returns the integer index where `x` is found in `arr` or `-1` if `x` is not found. You may assume there are no duplicate values stored in `arr`.
The linear search performed, after the interval is found, may be iterative.

Test your function with the two test cases from Task 1.1. [6]

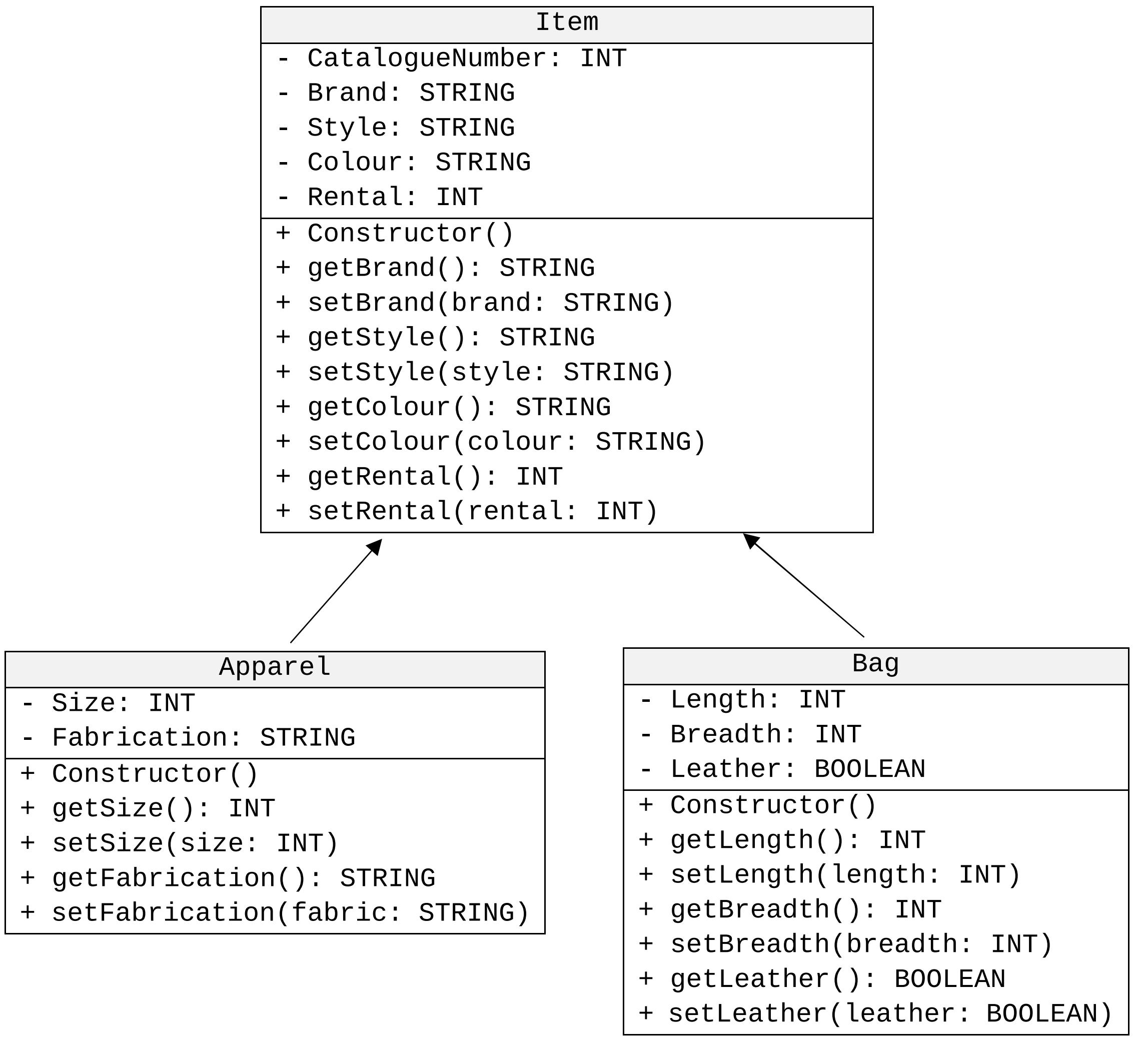
Download your program code for Task 1 as
`TASK1_<your class>_<your name>.ipynb`

Mark	Description

6	
	```
	def JumpSearchRecursive(arr, x, m, currentLastIndex): #[1]
	    #recursive jump [3]
	    #award one mark per condition for general case
	    #award one mark for general recursive call
	    if currentLastIndex < len(arr) and arr[currentLastIndex] < x:
	        return JumpSearchRecursive(arr, x, m, currentLastIndex+m)
	    else:
	    #no mark for linear search
	        for i in range(currentLastIndex-4, currentLastIndex):
	            if arr[i] == x: #found
	                return i

	    return -1 #not found [1]

	#Main Program
	arr = [ 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 999 ]

	print(JumpSearchRecursive(arr, 55, 4, 0)) #[1] for both; no 0.5 mark
	print(JumpSearchRecursive(arr, 1000, 4, 0))
	```

2 Task 2.1

Write program code to define the classes `Item`, `Apparel` and `Bag`. `Item` is the parent class of `Apparel` and `Bag` classes.



[8]

Mark	Description
1	Item class constructor is correct (attributes and data types).
1	Item class get methods are correct.
1	Item class set methods are correct.
1	Apparel and Bag classes inherits from Item class.
1	Apparel constructor method is correct.
1	Apparel get and set methods are correct.
1	Bag constructor method is correct.
1	Bag get and set methods are correct.

Task 2.2

The size of the hash table array is 100. In addition, the hashing algorithm that has been used is `CatalogueNumber MOD 100`. **Linear probing** is implemented to handle collisions.

A flat file `RECORDS.csv` stores the current rental records. Write the program code to:

- Initialise `hashtable[0:99]`.
- For each record in the flat file, instantiate an appropriate object (of either `Apparel` or `Bag` ADT).
- The rental information is stored in a tuple. The first element in the tuple is the object instantiated and the second element in the tuple is the customer's email address. For example: `(Apparel object, “mundepi@gmail.com”)`.
- Insert the tuple into `hashtable`.

[8]

Mark	Description
1	Read data from csv file correctly.
1	Close file.
1	Initialise hash table correctly (index 0 to 99).
	For each record:
1	Selection statement to instantiate either Apparel or Bag object.
1	Use set methods to assign data from file to object.
1	Initialise tuple with the two elements described.
1	Calculate index using hashing algorithm.
1	Insert tuple into hash table using linear probing.

Task 2.3

Display in neat columns, the index, catalogue number and customer email stored from index 0 to 10 of `hashtable`. [2]

Mark	Description
1	Correct data output.
1	Neat columns with 10 rows displayed.

Task 2.4

Hence, write program code for a function `HashTableSearch(hashtable, CatalogueNumber)` that returns the email address of the customer renting the item associated with `CatalogueNumber` , or “`Not Found`” if the item is not in `hashtable`. [8]

Mark	Description
1	Calculate index using hashing algorithm.
1	Element 0 of tuple to retrieve object.
1	Use get method to retrieve value of CatalogueNumber of the object.
1	Compare with the value from input argument.
1	Element 1 of tuple to retrieve email address.
1	Return email address if it matches.
1	Linear probing if it does not match.
1	Return “Not Found” when the index does not store a tuple or has an empty tuple.

Task 2.5

Test `HashTableSearch` function with the following `CatalogueNumber`:

i. 1399

ii. 1220 [2]

Mark	Description
1	Correct output TBC
1	Correct output TBC

Download your program code for Task 2 as
`TASK2_<your class>_<your name>.ipynb`

3 Style Theory decides to manage its rental using a NoSQL database instead. Information about the items is stored in the JSON file `RENTAL.JSON`.

The following fields are recorded:

- catalogue number,
- brand,
- category (i.e.: apparel or bag),
- daily rental fee,
- size (for apparel only),
- customer’s email
- start date, and
- end date.

Task 3.1
Write program code to import the information from the JSON file into a MongoDB database. Save the information under the `Rental` collection in the `StyleTheory` database. Ensure that the collection only stores the information from the JSON file. [2]

Mark	Description
1	Create StyleTheory database with Rental collection.
1	Import data from JSON to database.

Save your program code as
`TASK3_<your class>_<your name>.py`

Task 3.2
Write program code for a user to insert information of a new rental or update the end date of an existing rental. Display the following user menu for the user:

```
Style Theory
Option 1 – Insert new rental
Option 2 – Update existing rental
Option 3 – Quit
Enter your option (1, 2, or 3):
```

If Option 1 is entered, get user input for all relevant fields and insert the new document to the collection. Size data input is applicable for apparel only.

Otherwise, if Option 2 is entered, get user input for the catalogue number, start date and new end date and update the end date of the existing document.

Run your program to insert the following documents:

Field	Document 1	Document 2
Catalogue Number	1400	1375
Brand	Trioon	Gucci
Category	Apparel	Bag
Daily Rental Fee	20	50
Size	8	
Customer Email	liu@gmail.com	liu@gmail.com
Start Date	2022-01-10	2022-01-10
End Date	2022-01-14	2022-01-14

Run your program to update the following document:

Field	Document 1	Document 2
Catalogue Number	1371	1266
Start Date	2022-01-03	2022-01-03
End Date	2022-01-05	2022-01-05

[10]

Mark	Description
1	Loop until user enters 3.
1	Display menu output as per sample.
	Option 1
1	User does not input size data for bag.
1	Insert document to Rental collection.
	Option 2
1	get user input for the catalogue number, start date and new end date
1	update the end date of the existing document
2	Insert two documents correctly.
2	Update two documents correctly.

```
TASK3_<your class>_<your name>.py
```

Task 3.3
Write a procedure `display_all` that will display all the information in the `Rental` collection. Output `"NA"` for size if the document does not contain size field. Include an additional column to display the total amount payable (i.e.: daily rental fee multiplied by the number of days rented).

You may refer to the following Python code to calculate the difference in the number of days between two dates:

```
from datetime import date
f_date = date(2014, 7, 2)
l_date = date(2014, 7, 11)
delta = l_date - f_date
print(delta.days)
```

Run the `display_all` procedure. [8]

Mark	Description
1	Connect to database and collection.
1	Find all documents in that collection.
1	Display all fields for ach document neatly.
1	Selection to output “NA” in size for bags.
1	Split the date using ‘-‘ delimiter and pass the values to date method.
1	Calculate the difference in number of days.
1	Calculate total amount payable.
1	Correct output for all documents.

Add your program code to
`TASK3_<your class>_<your name>.py`

4 Style Theory's competitor JP Fashion uses a relational SQL database to manage its customers and rental details.

The table descriptions are as follow:

```
Customer(Email, FirstName, LastName, ContactNumber, DOB,
Address)
Product(CatalogueNumber, Category, Brand, Size, Fee)
CustomerRental(ID, Email*, StartDate, EndDate)
ProductRental(ID*, CatalogueNumber*, Returned)
```

Note: * denotes foreign key

Task 4.1

Create an SQL file called `TASK4_1_<your class>_<your name>.sql` with the SQL code to create database `JPFashion.db` with the four tables. [7]

Mark	Description
1	Correct sequence of table creation. Customer/ Product followed by CustomerRental and ProductRental.
1	Correct SQL code to create Customer table.
1	Correct SQL code to create Product table.
1	Correct SQL code to create CustomerRental table.
1	Correct SQL code to create ProductRental table.
1	CustomerRental table references to Customer table.
1	ProductRental table references to Product and CustomerRental tables.

Save your SQL code as
`TASK4_1_<your class>_<your name>.sql`

Task 4.2
The text files, `CUSTOMER.TXT`, `PRODUCT.TXT`, `CUSTOMERRENTAL.TXT` and `PRODUCTRENTAL.TXT` contain data for each table.

Write program code to read the records from the four text files and insert all information from the files into the `JPFashion.db`. [7]

Mark	Description
1	Connect to `JPFashion.db.`
4	Insert data from four text files.
1	In the correct sequence.
1	Close connection.

Save your program code as
`TASK4_2_<your class>_<your name>.py`

Task 4.3
Customers may create a new account using JP Fashion’s web application homepage.

STEP 1: User to enter email address in a form.
STEP 2: Prompt user to re-enter email address if it exists in Customer table. Otherwise, get user to enter his/ her first name, last name, contact number, date of birth and address).
STEP 3: Display message “`New account created successfully`”.

[12]

Write the code for a Python program as well as the necessary HTML files. Name the homepage `index.html`. And use `5678` as the port number.

Mark	Description
1	HTML file index.html for home page.
1	Text field to enter email address.
1	Submit button.
2	POST email address to server and query if it exists in Customer table.
1	HTML page to prompt user to re-enter email address if exists.
1	Repeat POST
1	HTML page to get other data if does not exist.
2	POST data to server and insert new record to Customer table.
1	HTML file to display message when new account has been created successfully.
1	Connect to local host with correct port number.

Save your program code as
`TASK4_3_<your class>_<your name>.py`

Task 4.4
Staff may visit `rentalsdue.html` to filter the rental information by end date and display the results on the web browser.

Write additional Python code and the necessary files to create a web application that:

- receives end date from a HTML form, then
- creates and returns a HTML document that enables the web browser to display the rentals due that day and have not been returned yet.

The output should include the following columns:

- Rental ID,
- catalogue number,
- email, and
- contact number. [10]

Mark	Description
1	HTML file `rentalsdue.html` created.
1	Text field to enter end date with submit button.
3	POST end date to server and perform JOIN query. End date matches CustomerRental, Returned in ProductRental is False.
1	Render template with data to be displayed.
3	Rentals due that day and have not been returned are displayed in table format (rows and columns). Loop program construct.
1	Output is correct with `‘2022-08-09’` as end date.

Run the web application and input end date as `‘2022-08-09’` in the webpage.

Save the output of the program as
`TASK4_4_<your class>_<your name>. html`

END OF PAPER