2023 JC2 Prelim Paper 2 - Suggested Marking Scheme

Qn	Task
1	**Task 1.1**
	Correct validation to ensure the input value is - a string - made up of 11 characters
	Correct validation to ensure the first 4 characters of the input value are made up of alphabets
	Correct validation to ensure the last 7 characters of the input value are made up of digits
	Return True if input value passes the validation
	Return False if input value fails the validation
	Task 1.2
	Correct test case implementation - input value: string containing only integers - output: False
	Correct test case implementation - input: string containing only letters - output: False
	Correct test case implementation - input: a valid string - output: True
	Correct test case implementation - input: string of incorrect length - output: False
	Task 1.3
	Correct mapping of letters to numbers
	Correct calculation of the check digit (incl. if check digit is equal to 10, return 0)
	Return check digit
	Penalties
	Task 1.4
	Correct function call to Task 1.1 to validate the input value
	Correct function call to Task 1.3 to calculate the check digit
	Correct comparison of calculated check digit against last character of code and return of boolean value
	Correct output for print(task1_4('PONU2079674')): True
	Correct output for print(task1_4('XONU2079674')): False
	Task 1.5
	Correct test cases: The same character should be changed in both test cases.
	Correct output for the test cases: True (marks awarded **ONLY** if test cases are correct)
2	**Task 2.1**
	File opened in appropriate mode & file handle closed appropriately (by calling **f.close()**)

	Data correctly read into an appropriate data structure
	Use of **insertion sort algorithm** - iteration through unsorted elements (to be inserted into sorted elements)
	- correct comparison (including appropriate key), to achieve descending order
	- correct determination of insertion position
	- correct execution for moving target element to insertion position
	Correct display of sorted data - **Facility** followed by **Number of Bookings**
	Task 2.2
	Use of **quick sort algorithm**
	- Base case handled appropriately
	- Correct selection of pivot (and exclusion from subarray)
	- Array is partitioned into **appropriate** subarrays
	- Subarrays are recursively sorted
	- Subarrays concatenated appropriately
	Correct display of sorted data
	Task 2.3
	Implement comparison logic for **Insertion Sort**
	Implement comparison logic for **Quick Sort**
	Display correct comparisons for both sort algorithms
	Care taken to ensure fairness of comparison (ie. Same unsorted array used for both functions, since the array may have been modified in-place by one of the functions)
	Determine which sort algorithm **is faster and** output an **appropriate conclusion**
3	**Task 3.1**
	Correct class declaration syntax: **Process**
	Constructor correctly sets **name**, **remaining**
	Method **execute()** correctly updates **remaining** attribute
	Method **is_completed()** returns boolean correctly
	Task 3.2
	Constructor correctly sets **items**, **front**, **rear**, **size** as attributes
	Instance attributes assigned appropriate data types
	Method **is_empty()** returns boolean correctly
	Method **is_full()** returns boolean correctly
	Method **enqueue(process)** correctly takes in a process object (and not the name of the object)
	Method **enqueue(process)** should ONLY add the process to the queue if not full
	Method **enqueue(process)** displays an appropriate error message if the queue is full
	Method **dequeue()** correctly removes a process from the queue
	Method **dequeue()** correctly returns the removed process
	Method **dequeue()** correctly displays an appropriate error message if the queue is empty
	Method **display()** correctly outputs processes from the **front** to the **rear** of the queue
	Method **display()** correctly displays an appropriate error message if the queue is empty

	Note: implicit requirement for Circular Queue is for enqueue and dequeue to be O(1), and display to be O(n), so any implementation with a worse time complexity will be penalised
	Task 3.3
	FCFSScheduler and **RRScheduler** subclasses inherit from the CircularQueue superclass
	Method **start()** - Polymorphism (both FCFS and RR implement start() method with same interface, different implementation)
	- Process dequeued with the dequeue() method (following encapsulation principle) and not through attribute access
	- Process enqueued with the enqueue() method (where appropriate)
	- Appropriate use of (conditional) loop, including terminating condition (in while statement; while True loop not accepted!)
	- Correct implementation of FCFS scheduling algorithm
	- Correct implementation of RR scheduling algorithm
	- Correct display (both classes are polymorphic and should display same info in the same format; mark only given if this condition is met)
	Task 3.4
	FCFS and **RR Schedulers** correctly instantiated
	Process is instantiated with the correct arguments and argument types
	All the processes are correctly enqueued to the schedulers
	Output from the respective schedulers correctly and clearly displayed
4	**Task 4.1**
	Flask app is run with correct route '/'
	HTML template file uses appropriate tags
	Menu options displayed as per the question
	Output shows menu options, *with correct links to subpages*
	Task 4.2
	Use **DELETE FROM Certificate**... to remove the records
	with **WHERE** clause to filter the erroneous records
	Appropriate use of **HTML** to render output
	Transaction is committed
	Output shows correct number of records removed
	Penalties (Not using parametrised strings, double quotes for string literals etc)
	Task 4.3
	Use **SELECT** to identify firstname, lastname, phone, description and issuedate attributes
[2m]	Use **INNER JOIN** on **Certificate/Member** and **Course** with appropriate **ON** condition
	Use **WHERE** to filter appropriately
	Use **ORDER BY** to sort by lastname ascending **(first)**
	Use **ORDER BY** to sort by firstname **(after lastname)** ascending
	Appropriate use of **HTML** to render table
	Output appropriately displayed
	Task 4.4

i	Route defined for web form with POST method (GET not accepted)
	Appropriate use of HTML elements to render form with submit button - member id, course code, issue date
	- status code appropriately set
	Form data extracted in Python
	Correct use of sqlite connection (instantiation + close)
	SQL: **INSERT INTO "Certificate"** (...) **VALUES** (...)
	Extracted data inserted into SQL query securely (using placeholders/named parameters)
	Appropriate check for insert success/error
	Status appropriately rendered in response (success or error)
ii	Correct result for member ID **145543**
	Correct result for member ID **151179**