# Tasks for IIUM KOE Python Class

| Task No (Marks) | Task Description | Input/Appended Code | | |
|---|---|---|---|---|
| **WEEK 1<br>(WRITE EACH TASK IN SEPARATE PYTHON FILES)** | | | | |
| Task 1a. (2m) | Create a program that asks the user to provide several numbers and output the total and the average of the numbers. | INPUT 1<br>3<br>3<br>4<br>5 | INPUT 2<br>5<br>1<br>2<br>3<br>4<br>5 | INPUT 3<br>2<br>-15<br>5 |
| Task 1b. (2m) | Create a program that prints all numbers from 0 to 50 and tell the user if the number is odd or even. | NO INPUT NOR APPENDED CODE | | |
| Task 1c. (2m) | Create a program that asks the user to provide 2 numbers and output the GCD (Greatest Common Divisor) and LCM (Lowest Common Multiple) of both numbers. | INPUT 1<br>5<br>100 | INPUT 2<br>2000<br>3 | INPUT 3<br>81<br>15 |
| Task 1d. (2m) | Create a program that calculates the length of hypotenuse of a right-angled triangle. The other 2 length are taken as inputs from the user. | INPUT 1<br>3<br>4 | INPUT 2<br>1.2<br>0.5 | INPUT 3<br>5<br>5 |
| **WEEK 2<br>(WRITE ALL TASK IN ONE PYTHON FILE)** | | | | |
| Task 2a. (2m) | Create a linked list that contains addToStart(), displayList() and count() as its internal functions. | APPENDED CODE<br>list = LL()<br>list.addToStart(10)<br>list.addToStart(20)<br>list.addToStart(45)<br>list.addToStart(35)<br>list.addToStart(15)<br>list.displayList()<br>print(list.count())<br>print(list.countOdd())<br>print(list.countEven())<br><br>list.addToStart(30)<br>list.addToStart(40)<br>list.displayList()<br>print(list.count())<br>print(list.countOdd())<br>print(list.countEven())<br><br>list.swap(1,5)<br>list.displayList()<br>list.swap(2,3)<br>list.displayList()<br>list.swap(7,4)<br>list.displayList()<br><br>list.max()<br>list.min()<br>list.range()<br>list.total()<br>list.average() | | |
| Task 2b. (3m) | Implement 2 internal functions – countOdd() and countEven() – that counts the even and odd numbers in the list. | | | |
| Task 2c. (3m) | Implement an internal function – swap() – that swaps 2 elements in the list using their index number. You may print out an error if necessary. | | | |
| Task 2d. (3m) | Implement several internal functions – max(), min(), range(), total(), average() – that calculates the maximum number and the minimum number from the list as well as the range of the numbers respectively. | | | |

| **WEEK 3**<br>**(WRITE ALL TASKS IN ONE PYTHON FILE)** | | |
|---|---|---|
| Task 3a.<br>(2m) | Create a linked list that contains push(), pop(), enqueue(), dequeue(), displayList() and count() as its internal functions. | <u>APPENDED CODE</u><br>`listA = LL()`<br>`listA.push('P')`<br>`listA.push('Y')`<br>`listA.enqueue('T')`<br>`listA.enqueue('H')`<br>`listA.push('O')`<br>`listA.enqueue('N')`<br>`listA.displayList()`<br>`print(listA.count())`<br><br>`listA.pop()`<br>`listA.dequeue()`<br>`listA.pop()`<br>`listA.displayList()`<br>`print(listA.count())` |
| Task 3b.<br>(3m) | Implement an external function that converts the linked list into an array. The original linked list will be cleared, and the array will be returned. The function needs the linked list as its parameter and returns an array. You MUST implement stack or queue functions as well as the count function from 3a into the external functions. | <u>APPENDED CODE</u><br>`listA = LL()`<br>`listA.push('I')`<br>`listA.enqueue('L')`<br>`listA.enqueue('O')`<br>`listA.enqueue('V')`<br>`listA.enqueue('E')`<br>`listA.push('C')`<br>`listA.push('O')`<br>`listA.push('D')`<br>`listA.push('I')`<br>`listA.push('N')`<br>`listA.push('G')`<br>`listA.displayList()`<br><br>`arrayA =`<br>`convertToArray(listA)`<br>`listA.displayList()`<br>`print(arrayA)` |
| Task 3c.<br>(3m) | Implement an external function that reverses a linked list into another empty linked list. The original linked list will be cleared. The function needs 2 linked list as its parameter and have no return value. You MUST implement stack or queue functions from 3a into the external function. | <u>APPENDED CODE</u><br>`listA = LL()`<br>`listB = LL()`<br>`listA.push('I')`<br>`listA.enqueue('A')`<br>`listA.enqueue('M')`<br>`listA.push('A')`<br>`listA.enqueue('M')`<br>`listA.enqueue('U')`<br>`listA.enqueue('S')`<br>`listA.enqueue('L')`<br>`listA.enqueue('I')`<br>`listA.enqueue('M')`<br>`listA.displayList()`<br><br>`reverseList(listA,listB)`<br>`listA.displayList()`<br>`listB.displayList()` |
| **WEEK 4**<br>**(WRITE ALL TASKS IN ONE PYTHON FILE)** | | |

| | | |
|---|---|---|
| Task 4a.<br>(3m) | Create a Hash Table of size 5 that has the following features.<br>- It can hash only string keys. It does so by taking the first letter (all of them are lowercase can change it to a number according to the alphabetical order (a is 1, b is 2 and so on). You don’t have to worry about numerical keys.<br>- It uses linear rehashing whenever collision happens.<br>- It must contain __setitem__ and __getitem__ functions | APPENDED CODE<br>`hash = HT()`<br>`hash[‘name’] = ‘Zikri’`<br>`hash[‘age’] = 20`<br>`hash[‘gender’] = ‘M’`<br><br>`print(hash[‘name’])`<br>`print(hash[‘age’])`<br>`print(hash[‘gender’])`<br>`print(hash.keys)`<br>`print(hash.values)`<br><br>`hash[‘birthdate’] = ‘4/12/2003’`<br>`print(hash.keys)`<br>`print(hash.values)`<br>`print(hash[‘birthdate’])`<br><br>`hash[‘name’] = ‘Hakim’`<br>`print(hash[‘name’])`<br><br>`hash.delete(‘age’)`<br>`print(hash.keys)`<br>`print(hash.values)`<br>`print(hash[‘age’])`<br>`print(hash[‘birthdate’])` |
| Task 4b.<br>(2m) | Implement an internal function delete() that deletes a value inside the hash table using its key as the parameter. | |
| **WEEK 5<br>(WRITE TASK A IN ONE PYTHON FILE WHILE TASK B AND C IN ANOTHER PYTHON FILE)** | | |
| Task 5a.<br>(2m) | Create a binary tree as well as 3 external functions – printPreorder(), printInorder() and printPostorder() – that display the binary tree in their respective order traversals. | APPENDED CODE<br>`root = Node(2)`<br>`root.left = Node(3)`<br>`root.right = Node(7)`<br>`root.left.left = Node(1)`<br>`root.right.left = Node(6)`<br>`root.left.right = Node(4)`<br>`root.left.right.left = Node(5)`<br>`root.right.right = Node(8)`<br><br>`printPreorder(root)`<br>`printInorder(root)`<br>`printPostorder(root)` |
| Task 5b.<br>(2m) | Create a function – insertSort() – that utilizes the binary tree structure to autosort strings being added into the binary tree. Implement also the 3 external functions from 5a. | APPENDED CODE<br>`root = Node('Haziman Sairin')`<br>`insertSort(root,'Zikri Hakim')`<br>`insertSort(root,'Jameel Majdi')`<br>`insertSort(root,'Raniya Waleed')`<br>`insertSort(root,'Syukri Talib')`<br>`insertSort(root,'Saif al-Din')`<br>`insertSort(root,'Nuqman Aliff')`<br>`insertSort(root,'Abd al-Karim Mumtaz')` |
| Task 5c.<br>(4m) | Implement a internal function – deepestBranch() – that finds the deepest branch in the binary tree from 5b. If there are several of them, take only the first one. Display depth of the branch as well as the path taken to get to the branch.<br>(Use ‘Left’ or anything equivalent for left branch and ‘Right’ or anything equivalent for right branch) | |

|  |  |  |
|---|---|---|
|  |  | `insertSort(root,'Kizzy Harriette')`<br>`insertSort(root,'Zharif Aiman')`<br>`insertSort(root,'Sharifa Harun')`<br>`insertSort(root,'Najma Fuad')`<br>`insertSort(root,'Amir Su\'ad')`<br><br>`printPreorder(root)`<br>`printInorder(root)`<br>`printPostorder(root)`<br>`root.deepestBranch()` |