RIVER VALLEY HIGH SCHOOL
YEAR 6 END OF YEAR EXAMINATION

H2 COMPUTING 9597
Paper 1
14 AUGUST 2018

3 HOUR 15 MINUTES

NAME _____

CLASS 6 ( )

INDEX NO. _____

READ THESE INSTRUCTIONS FIRST **DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.**		
Answer **all** questions. All tasks must be done in the computer laboratory. You are not allowed to bring in or take put any pieces of work or materials n paper or electronic media or in any other form. Approved calculator is allowed. The number of marks is given in brackets [ ] at the end of the task. **15 minutes** before the end of the examination, you should start printing your **evidence file**. Submit **all source files** in the **thumb drive** and do not delete your source files in the examination laptop.		**FOR EXAMINERS' USE** **1** **/25** **2** **/35** **3** **/25** **4** **/15** **TOTAL** **/100**

This Question Paper consists of 14 printed pages.

1. Accumulative Season Scores

For the first task, you are required to read the text file *"T1_gamescore.txt"* which contains the accumulative scores of a group of players from an online game over 12 seasons.

The format of a line of the text file is as follow:
`<player1_name>,<class_1>:<ss1>,<ss2>,<ss3>,<ss4>,<ss5>,<ss6>,<ss7>,<ss8>,<ss9>,<ss10>,<ss11>,<ss12>\n`

E.g.
`Rufus,priest:4255,17366,22616,31889,49634,58847,78112,86631,94924,106955,125372,130725`

The above line represents the player `Rufus` plays the class `priest` and his 12 accumulative seasonal scores are `4255, 17366, 22616, 31889, 49634, 58847, 78112, 86631, 94924, 106955, 125372` and `130725` respectively.

Task 1.1 – Read the file

Implement the procedure `readfile`. It reads the file "*T1_gamescore.txt*" and returns a list of lists as below.

```
>>> results = readfile()
>>> results
[['Rufus', 'priest', 4255, 13111, 5250, 9273, 17745, 9213, 19265,
8519, 8293, 12031, 18417, 5353], ['Ione Wolfe', 'warrior', 2827,
17757, 3612, 6818, 11772, 9161, 4393, 10469, 10567, 15424, 7307,
10014],……,['Carola Tegeler', 'rogue', 6407, 14795, 0, 5004,
16084, 14960, 11879, 16545, 10247, 5617, 7345, 0]]
```

Take note that the returned scores in the lists are not accumulative scores but the actual season scores. It can be computed by:

n^{th} season score = n^{th} – $(n – 1)^{th}$ accumulative season score

Evidence 1

Program code of procedure `readfile`. [6]

Task 1.2 – Count Class

Using the data from *"T1_gamescore.txt"*, implement the function `count_class`. It takes in the string `clss` as parameter and returns an integer which contains the number of players who play the class indicated by `clss`.

For example:

```
>>> count_class("warrior")
57
```

This means that the number of players who play the class warrior is 57.

Evidence 2

Program code of function `count_class`. [2]

Evidence 3

Screenshot of the output of the following code. [1]

```
for clss in ["warrior", "mage", "priest"]:
    print("Number of " + clss + ":" + str(count_class(clss)))
```

Task 1.3 – Top Class by Season

Using the data from *"T1_gamescore.txt"*, implement the function `top_class_by_season`. It takes in the integer `season` as parameter and returns a string which contains the class which holds the highest average score of the season. (Code with the assumption that you do not know how many and what the classes are available in this game.)

The average score of the season of a class is calculated by summing up all the scores of the players who play the class divided by the total number of players who play the same class.

```
>>> top_class_by_season (6)
"priest"
```

Evidence 4

Program code of function `top_class_by_season`. [6]

Evidence 5

Screenshot of the output of the following code. [1]

```
for i in range(1, 13):
    print("Top class in season " + str(i) + ":")
    print(top_class_by_season(i))
```

Task 1.4 – Top n Players by Season

Using the data from *"T1_gamescore.txt"*, implement the function `top_n_players_by_season`. It takes in the integers `n` and `season` as parameters and returns a list of strings which contains names of the top `n` players.

```
>>> top_n_players_by_season(3, 2)
['bumpkintiger', 'diamondagile', 'milkshakessulky']
>>> top_n_players_by_season(10, 4)
['Carolann Kintner', 'lewdsterpub', 'fabindustry', 'eggfun',
'fellradial', 'Grazyna Kitzman', 'chapteridentical',
'chubbysourdough', 'palmears', 'leardfluttering']
```

Evidence 6

Program code of function `top_n_players_by_season`. [5]

Evidence 7

Screenshot of the list of names of the top 20 players in season 9. [1]

Task 1.5 – Stagnant Players

Using the data from *"T1_gamescore.txt"*, implement the function `find_stagnant_players` which returns a list of strings which contains names of the players who did not play in at least 1 season.

Evidence 8

Program code of function `find_stagnant_players`. [2]

Evidence 9

Screenshot of the output of program. [1]

2. Health Workshops

To promote healthy lifestyle, a local company decided to sponsor its 500 employees up to 3 health workshops of their choice. The company decides to maintain this piece of information using a binary search tree (BST) with each node contains a string called `employeeID` which is used as the key of the BST, and an array structure called `workshops` to store the name(s) and cost(s) of the health workshop(s) chosen by the employee.

Task 2.1

Implement the BST structure and perform insertion using the 8-character employee ID as key. During the insertion process, the employee's respective health workshop(s) information must also be stored in the node. The information of the employees and their chosen workshops can be found in *"T2_healthworkshops.txt"* and its format is as follow:

```
<employeeID>-[<workshopName1>,<cost1>][<workshopName2>,<cost2>]
```

For example:
`5175590R-[Yoga With Yoyo,60][Decoding The Nutrition Label,55]`
This means that the employee with `employeeID` 5175590R has chosen "Yoga With Yoyo" and "Decoding The Nutrition Label" workshops and the cost of these 2 workshops are $60 and $55 respectively.

The BST structure must be able to handle up to 500 nodes. You need to implement the following functions.

Functions	Description	Marks
`createBST()`	This function creates a BST, reads the file "*T2_workshops.txt*", perform insertion of data and returns a BST structure.	15
`inOrderTraversal()`	This procedure outputs all employee IDs from the BST in-order.	2
`findWorkshopsById (employeeID)`	This function returns a list of names of all the workshops chosen by the employee with employee ID as `employeeID`.	3
`findIdsByWorkshop (workshopName)`	This function returns a list of employee IDs of employees who have chosen the workshop indicated by the string `workshopName`.	2
`findTotalCost()`	This function returns the cost of all the workshops chosen by all the employees in the company as an integer.	2

Evidence 10

Program code for Task 2.1. [24]

Evidence 11

Screenshot the output of the following code. [1]

```
def test1():
    b1 = createBST()
    print(b1.findTotalCost())
    print(b1.findWorkshopsById("1001278B"))
    print(b1.findWorkshopsById("1019563R"))
    print(b1.findWorkshopsById("3161202Y"))
    print(b1.findWorkshopsById("5095845H"))
    print(b1.findWorkshopsById("9965997Y"))
    print(b1.findWorkshopsById("9998622F"))

    print(b1.findIdsByWorkshop('Diabetes 101'))
    print(b1.findIdsByWorkshop('The Truth About Carbs'))
    print(b1.findIdsByWorkshop('Nutrition Nuts And Bolts'))
    print(b1.findIdsByWorkshop('Yoga With Yoyo')
test1()
```

Task 2.2

Write a menu which has the following options.

```
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
```

The validation of the employee ID follows the rules below:

- The last of employee ID is the check code.
- The algorithm to generate the check code is as follows:
    - Obtain the weighted sum of the 7 digits using the weights <2,7,6,5,4,3,2>
    - Find the remainder of the weighted sum when divided by 17
    - Look the check code up in the table below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	B	C	D	E	F	G	H	J	M	N	Q	R	S	T	Y	Z

Below is an example of how the menu works.

```
>>>menu()
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
Choose an action: 1
```

```
File read. BST created.
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
Choose an action: 2
Type a userID: 1001278B
['Stress Management For Managers: Helping You and Your Team
Stress Less']
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
Choose an action: 3
Type a workshop name: Yoga With Yoyo
['1026960S', '1425017Z', '1518324H', '1602033B', '1712684E',
'1855773Y', '1914889R', '1918475F', '2131933A', '2174550G',
'2448074G', '2544510E', '2570896Z', '2882640B', '3307418R',
'4583774N', '4881580J', '5015391T', '5175590R', '6010700R',
'6237846Z', '6285730A', '6382864S', '6397129T', '6695719R',
'6801170S', '6865739T', '7357796T', '7498335T', '7839365D',
'8154005Y', '8958192T', '9023193T', '9038581D', '9540693Y',
'9750174Q', '9964906Y']
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
Choose an action: 4
1001278B
1006415N
...
9978729Q
9998622F
1) Read file to generate BST
2) Find workshop(s) by user ID.
3) Find user ID(s) by workshop.
4) Display users in order.
5) Total cost.
6) Quit.
Choose an action: 6
Quit.
```

Evidence 12

Program code of `menu` with full input validations. [10]

3. Cards Party

A deck of French playing cards is the most common deck of playing cards used today. It includes thirteen **ranks** of each of the four **suits**: clubs (♣), diamonds (♢), hearts (♡) and spades (♠).

Club	A♣	2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣
Diamond	A♢	2♢	3♢	4♢	5♢	6♢	7♢	8♢	9♢	10♢	J♢	Q♢	K♢
Heart	A♡	2♡	3♡	4♡	5♡	6♡	7♡	8♡	9♡	10♡	J♡	Q♡	K♡
Spade	A♠	2♠	3♠	4♠	5♠	6♠	7♠	8♠	9♠	10♠	J♠	Q♠	K♠

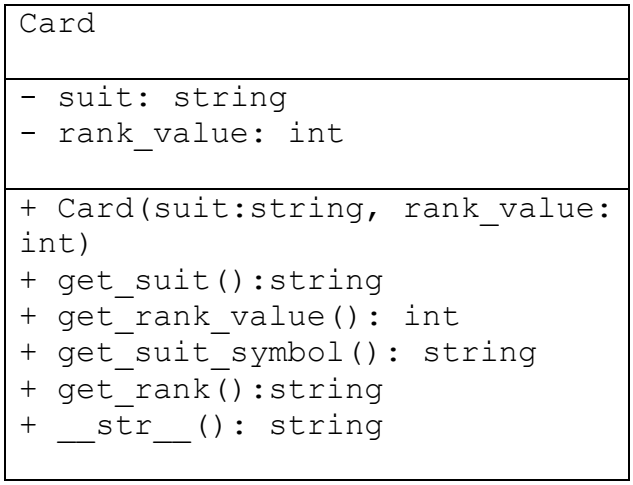
In this task, the `rank` and `suits` will follow the following order when a comparison is needed.

	Ranks from smallest to largest (left to right)												
Rank	A	2	3	4	5	6	7	8	9	10	J	Q	K
Values Represented	1	2	3	4	5	6	7	8	9	10	11	12	13

	Suits from smallest to largest (left to right)			
Suits	Club	Diamond	Heart	Spade
Python Expression	u"\u2663"	u"\u2662"	u"\u2661"	u"\u2660"

Task 3.1 – Card Class

Implement the `Card` class based on the following UML class diagram. The descriptions for some class methods can be found below.



Functions	**Descriptions**
`get_suit_symbol(): string`	Return the Unicode symbol corresponding to the `card`'s `suit`.
`get_rank():string`	Return a string from `"A", "2", "3", … "10", "J", "Q", "K"` corresponding to the `card`'s `rank_value`.

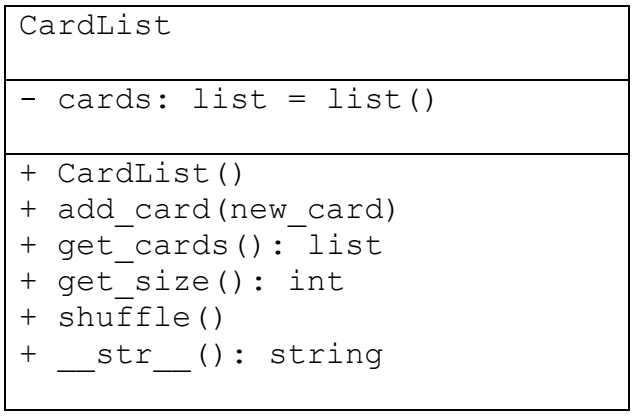
`__str__(): string`	Return a string with length of 3-character, stating the `card`'s `rank` followed by its `suit`. For example: `" K♠", "10♡", " 8♣"`

Evidence 13

Screenshot of the program. [4]

Task 3.2 – CardList Class

Implement the `CardList` class based on the following UML class diagram. The descriptions for some class methods can be found below.



Functions	Descriptions
`add_card(new_card)`	Add a new `card` into the current list of `cards`.
`get_cards(): list`	Return the current list of `cards`.
`get_size():int`	Get the size of the current list of `cards`.
`shuffle()`	Shuffle the current list of `cards`.
`__str__(): string`	Return a `string` consisting of each `card` in the list of `cards`, separated by commas. For example: " K♠, 9♠, 5♠, A♠, 10♡, 6♡, 2♡, J♢, 7♢, 3♢, Q♣, 8♣, 4♣"

Evidence 14

Screenshot of the program. [4]

Task 3.3 – Sort your hand

Implement two additional methods in the `CardList` class based on the descriptions below.
State the best-case and worst-case time complexity of the two methods.

Functions	Descriptions
`sort_by_suit()`:	Sort the current list of `cards` by suit first, then by rank, in **ascending** order. Implement the method by adopting the algorithm of **insertion sort**.
`sort_by_rank()`:	Sort the current list of `cards` by rank first, then by suit, in **ascending** order. Implement the method by adopting the algorithm of **improved bubble sort**.

Evidence 15

Screenshot of the program.
Best-case and worst-case time complexity of the two methods. [8]

Task 3.4 – Test Sorting Algorithms

Implement a function `test_sorts()` and design 2 test cases. Run each test case against both methods and test if they are implemented correctly.
Write down the **justification** of each test case using comments.

Evidence 16

Screenshot of the program and output. [4]

Task 3.5 – Hand Ranking

A hand of five cards can be ranked according to the following categories from low to high.

Name	Example
Normal	A normal hand containing five cards not of sequential rank nor of the same suit, such as K♥ J♥ 8♣ 7♦ 4♠.
Straight	A straight hand containing five cards of sequential rank, not all of the same suit, such as 7♣ 6♠ 5♠ 4♥ 3♥. (AKQJ10 is included.)
Flush	A flush hand containing five cards all of the same suit, not all of sequential rank, such as K♣ 10♣ 7♣ 6♣ 4♣.
Straight Flush	A straight flush is a poker hand containing five cards of sequential rank, all of the same suit, such as Q♥ J♥ 10♥ 9♥ 8♥.

Implement an function `compare_five_cards()`. The function should take in 2 `CardList` objects, `hand1` and `hand2`. If both hands have exactly five cards, compare the hands and return a string indicating if the first hand wins, loses, or there is a draw.

For this task, if both hands fall into the same category, it is not necessary to further compare the value. The comparison result will be deemed as a draw.

Evidence 17

Screenshot of the program. [5]

4. Reversi

In this task you will implement part of the board game called Reversi.

Task 4.1

Implement the function `initiate_board` which takes in an integer `size` and returns a 2-dimension list that contains `size` x `size` `" "`.

For example,

```
>>> initiate_board(2)
[[' ', ' '], [' ', ' ']]

>>> initiate_board(10)
[[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]
```

Evidence 18

Program code of `initiate_board`. [1]

Task 4.2

Implement the function `display_board` which takes in a 2-dimension list `board` and output the playboard following the format below.

For example,

```
>>> b1 = [['X', ' ', 'O'], [' ', 'X', ' '], ['O', 'X', 'O']]

>>> display_board(b1)
--0-1-2-
0|X| |O|
1| |X| |
2|O|X|O|
--------
```

```
>>> b2 = [[" "," "," "," "," "," "," "," "],
          [" "," "," "," "," "," "," "," "],
          [" "," "," "," "," "," "," "," "],
          [" "," "," ","X","O"," "," "," "],
          [" "," "," ","O","X"," "," "," "],
          [" "," "," "," "," "," "," "," "],
          [" "," "," "," "," "," "," "," "],
          [" "," "," "," "," "," "," "," "]]

>>> display_board(b2)
--0-1-2-3-4-5-6-7-
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | |X|O| | | |
4| | | |O|X| | | |
5| | | | | | | | |
6| | | | | | | | |
7| | | | | | | | |
-----------------
```

Evidence 19

Program code of `display_board`. [4]

Task 4.3

Implement the function `reverse` which simulates a valid move in the game Reversi.

It takes in 4 parameters as shown below.

- `board` – a 2-Dimension list that represents the current state of the play board.
- `row` – an integer that represents the row number at which the disc is placed.
- `col` – an integer that represents the column number at which the disc is placed.
- `disc` – a string that takes either the value of `"X"` or `"O"` which represents black and white of the disc.

The function returns `True` if it is a valid move and then updates `board` accordingly, otherwise, returns `False` without updating `board`.

A valid move refers to, for example, placing a “X” disc at a position that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another “X” piece, with one or more contiguous “O” pieces between them, and vice versa.

Examples

`reverse(board, 3, 2, "O")` is a valid move

```
--0-1-2-3-4-5-6-7-        --0-1-2-3-4-5-6-7-
0| | | | | | | | |       0| | | | | | | | |
1| | | | | | | | |       1| | | | | | | | |
2| | |X|O| | | | |       2| | |X|O| | | | |
3| | | |X|O| | | |       3| | |O|O|O| | | |
4| | | |O|X| | | |       4| | | |O|X| | | |
5| | | | | | | | |       5| | | | | | | | |
6| | | | | | | | |       6| | | | | | | | |
7| | | | | | | | |       7| | | | | | | | |
------------------        ------------------
```

`reverse(board, 6, 5, "X")` is a valid move.

```
--0-1-2-3-4-5-6-7-        --0-1-2-3-4-5-6-7-
0|X| | | | | | | |       0|X| | | | | | | |
1|X|X| | | | | | |       1|X|X| | | | | | |
2|X|O|X|O| | | | |       2|X|X|X|O| | | | |
3|X|O|O|X|O| | | |       3|X|O|X|X|O| | | |
4|X| |X|O|O| | | |       4|X| |X|X|O| | | |
5| | | | |O| | | |       5| | | | |X| | | |
6| | | | | | | | |       6| | | | | |X| | |
7| | | | | | | | |       7| | | | | | | | |
------------------        ------------------
```

`reverse(board, 3, 4, "X")` is a valid move.

```
--0-1-2-3-4-5-6-7-        --0-1-2-3-4-5-6-7-
0| | | |X| | | |X|       0| | | |X| | | |X|
1|X| | |O| | |O| |       1|X| | |O| | |X| |
2| |O| |O| |O| | |       2| |O| |O| |X| | |
3| | |O|O| | | | |       3| | |O|O|X| | | |
4|X|O|O|O|O|O|X| |       4|X|O|O|X|O|O|X| |
5| | |O|O|O| | | |       5| | |X|O|O| | | |
6| |O| |O| |O| | |       6| |X| |O| |O| | |
7|X| | |X| | |X| |       7|X| | |X| | |X| |
------------------        ------------------
```

`reverse(board, 4, 3, "X")` is a valid move.

```
--0-1-2-3-4-5-6-7-        --0-1-2-3-4-5-6-7-
0| | | |X| | | |O|       0| | | |X| | | |O|
1|X| | |O| | |O| |       1|X| | |X| | |O| |
2| |O| |O| |O| | |       2| |X| |X| |O| | |
3| | |O|O|O| | | |       3| | |X|X|O| | | |
4|X|O|O| |O|O|X| |       4|X|X|X|X|X|X|X| |
5| | |O|O|O| | | |       5| | |X|X|X| | | |
6| |O| |O| |O| | |       6| |X| |X| |X| | |
7|X| | |X| | |X| |       7|X| | |X| | |X| |
------------------        ------------------
```

Evidence 20

Program code of the iterative version of function `reverse`. [5]

Program code of the recursive version of function `reverse`. [5]

The End