VICTORIA JUNIOR COLLEGE
JC 2 CONTINUAL ASSESSMENT 2
Higher 2

CANDIDATE NAME .....

CT GROUP .....

COMPUTING 9569/02

Paper 2

21 May 2025

2 hours

Additional Materials: Electronic version of `shuffle.py` Python file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.
All tasks must be done in the computer laboratory.

Approved calculators are allowed. The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Save each task as it is completed.

Note that up to **3** marks out of 65 will be awarded for the use of common coding standards for programming style.
The numbers of marks is given in brackets [ ] at the end of each question or part question.
The total number of marks for this paper is 65.

For Examiner's Use	
Total	**/ 65**

This document consists of **7** printed pages and **1** blank page.

[Turn over

Instruction to candidates:

Your program code and output for each of Task 1 to 2 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code and output for Task 1 should be saved as:

`TASK1_<your name>_<class>_<index number>.ipynb`

Make sure that each of your .ipynb files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

`TASK1_<your name>_<your class>_<index number>.ipynb`

A game designer is designing a game for 4 players and needs you to program a prototype for the setup of the game.

Players in the game have 13 randomly generated tiles. This is called a 'hand'. The tiles can be one of 5 suits (Bamboo, Character, Dot, Dragon or Wind). There are 4 copies of each tile. The game also has a 'wall' which the players draw tiles from.

A queue will be used to simulate a player's hand and the wall.

The class `Tile` contains two attributes: Hide or not ?

- `suit` the string suit of the tile
- `value` the integer/string value of the tile

The class `Tile` contains the following methods:

- `get_suit()` that returns the suit of the tile
- `get_value()` that returns the value of the tile

The class `Queue` contains four attributes: Hide or not ?

- `queueList` a 1-dimensional list of `Tile` objects
- `headPointer` the index of the first item in the queue, initialised to 0
- `tailPointer` the index of the last item in the queue, initialised to −1
- `numberItems` the quantity of tiles in the queue

The class `Queue` has the following methods:

- a constructor that initalises the `headPointer`, `tailPointer` and `numberItems` to appropriate values for an empty queue
- `is_empty()` that returns `True` if the queue is empty, or `False` otherwise
- `enqueue()` that takes a `Tile` as a parameter and inserts it into the queue
- `dequeue()` that returns the contents of the removed tile as a tuple `(suit, value)`, or `False` if empty
- `printQueue()` that outputs all current contents in the queue, from the first tile in the queue to the last in a single line, without removing any of the tiles.
  Your output should be in the following format:
  `First (suit_1, value_1) (suit_2, value_2) Last`
  where `First` refers to the start of queue and `Last` refers to the end of queue.

For each of the sub-tasks, 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:
```

Task 1.1

Write program code to declare the class `Tile` and its methods. [2]

Task 1.2

Write program code to declare the class `Queue` and its methods. [8]

Task 1.3

A complete set of tiles in the game consists of:

- 36 tiles each for Bamboo, Character and Dot with each suit containing 4 copies of the values 1 to 9
- 12 Dragon tiles (4 copies of the values Red, Green, White)
- 16 Wind tiles (4 copies of the values East, South, West, North)

The `Wall` class inherits from `Queue` and has the following methods:

- a constructor that inherits the attributes of the parent class and initialises the complete set of tiles to the wall
- `shuffle` that shuffles all the tiles in the wall. Use `shuffle.py` and adapt it for this method. Do **not** change the value of `random.seed()`
- `build_wall()` that adds the complete set of tiles into `queueList` in the following order:
    - 4 copies of values 1 to 9 of Bamboo, Character and then Dot tiles (i.e (Bamboo, 1), (Bamboo, 1), …, (Bamboo, 2), (Bamboo, 2), …)
    - 4 copies of values Red, Green and White of Dragon tiles
    - 4 copies of values East, South, West and North of Wind tiles

The tiles need to be shuffled after being added into the wall.

Write program code to declare the class `Wall` and its methods.

Test your program by outputting the number of tiles in the wall. [8]

Task 1.4

The `PlayerHand` class inherits from `Queue` and has the following methods:

- `add_tile()` that takes a `Tile` as a parameter and inserts it into the hand
- `get_tiles()` that outputs all current contents in the hand, from the first tile in the hand to the last, without removing any of the tiles
- `sort_hand()` that sorts the tiles in the hand by suit in alphabetical order then value in ascending/alphabetical order using optimised bubble sort.

Write program code to declare the class `PlayerHand` and its methods. [6]

Task 1.5

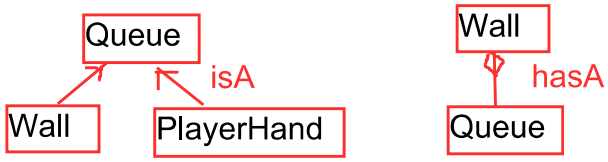
Each player is to draw tiles from the wall until the player's hand has 13 tiles.

Write a program to generate a hand for a player and sort the hand by suit in alphabetical order then value in ascending/alphabetical order. Output the contents of the hand before and after sorting. The contents of a hand must be output on one line. [4]

Save your Jupyter Notebook for Task 1.

This is a bad OOP design!

Violates the The Liskov Substitution Principle (LSP)



2 Name your Jupyter Notebook as:

`TASK2_<your name>_<class>_<index number>.ipynb`

A game designer is creating a colour sorting puzzle game where players need to sort coloured balls into tubes.

The game consists of three tubes, and each tube can hold up to three balls. The balls can be either Red (R) or Blue (B). The objective is to sort the balls so that each non-empty tube contains balls of the same colour.

The rules for moving balls between tubes are:

- A ball can only be moved to another tube if it is the topmost ball in its current tube
- A ball can only be moved to another tube if:
  - The destination tube is empty, or
  - The topmost ball in the destination tube is the same colour

The game uses a stack data structure to represent each tube.

The `Stack` class contains the following attributes: Hide ?

- `capacity` is the maximum number of items a stack can hold (set the default parameter to be 3)
- `items` is a list to store the items where each item is initialised as a single space string, i.e. `" "`
- `top` points to index of the topmost item in a stack

For each of the sub-tasks, 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 2.1
         Program code

         Output:
```

Task 2.1

Write program code to declare the class `Stack` and its constructor. [2]

Task 2.2

Amend your code for **Task 2.1** to declare the following `Stack` methods:

- `is_empty()` to check if the stack is empty
- `is_full()` to check if the stack is full
- `get_capacity()` to return the capacity of the stack
- `get_items()` to return the balls in the stack stack pollution
- `peek()` to return the topmost item without removing it from the stack. [3]

Task 2.3

Amend your code for **Task 2.2** to declare the following `Stack` methods:

- `push()` to add a ball to the top of the stack
- `pop()` to remove and return the topmost ball.

In a new cell, write program code to:

- initialise a new Stack
- call `push()` to insert `B`, `B`, `R` into the stack, where `R` is at the top of stack
- output the contents of the stack in the following format:

```
top
R
B
B
```

- call `pop()`
- output the contents of the stack in the same format. [5]

Task 2.4

Write a function `initialise_game()` that:

- creates three `Stack` objects
- initialises the first tube with `B`, `B` and `R`
- initialises the second tube with `B`, `R` and `R`
- returns a list containing the three `Stack` objects [3]

Task 2.5

Write a function `display_game(tubes)` that displays the current state of all tubes in the following format:

```
[R][R][ ]
[B][R][ ]
[B][B][ ]
 1  2  3
```

```
[
 ['B','B','R'],    1
 ['B','R','R'],    2
 [' ',' ',' ']     3
]
```

[2]

Call the function to display the above output.

Task 2.6

Write a function `is_valid_move(tubes, source, destination)` that:

- takes the list of tubes and two integers, `source` and `destination`
- returns `True` if a ball can be moved from the source tube to the destination tube according to the game rules and by using the methods in `Stack` class.
- returns `False` otherwise. [3]

Task 2.7

Write a function `move(tubes, source, destination)` that:

- moves a ball from the source tube to the destination tube if the move is valid by calling the function declared in **Task 2.6**
- returns `True` if the move was successful
- returns `False` otherwise. [3]

Task 2.8

Write a function `is_game_won(tubes)` that:

- returns `True` if each tube is either empty or full and contains balls that are all the same colour.
- returns `False` otherwise. [2]

In a new cell, write program code to:

- initialise the game
- allow the player to input source and destination tube numbers
- display appropriate messages for invalid moves
- continue until the game is won

Test your program and show the output. [4]

Save your Jupyter Notebook for Task 2.

Task 2.9

Write a Python program and the necessary files to create a web application that displays the initialised state of all three tubes in a table format where:

- each tube is represented as a column in the table
- each cell either shows the ball colour ('R' for red, 'B' for blue) or is empty
- the table header shows the tube number 1, 2 and 3.

The program should return an HTML document that enables the web browser to display the game interface.

Save your Python program as:
`TASK_2_9_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders in a folder named:
`TASK_2_9_<your name>_<centre number>_<index number>` [6]

Run the web application.

Save the webpage output as:
`TASK_2_9_<your name>_<centre number>_<index number>.html` [1]

End of Paper

BLANK PAGE