OBJECT-ORIENTED PROGRAMMING REPORT

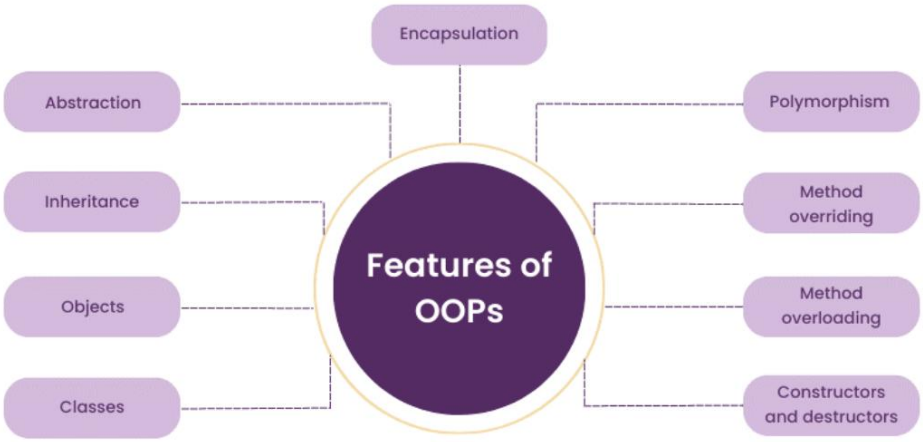
Table of contents

U3_1.1 The key features of object-oriented programming.

Object-oriented programming (OOP) is a programming paradigm that organises code into objects, which are like real-world entities that have attributes (data) and behaviors (functions).

By using these key features of object-oriented programming, developers can create modular, reusable, and organized code that mimics real-world entities and relationships, making it easier to manage and maintain complex software systems

Here are the key features of OOP explained below.



SHORT DESCRIPTION OF KEY FEATURES OF OOP

- **CLASSES** are like blueprints or templates for creating objects. They define the attributes (data) and behaviors (functions) that objects of that class will have.
- **OBJECTS** are instances of classes. They represent specific instances of the class with their own unique data and behavior.
- **ENCAPSULATION** is the idea of bundling data (attributes) and methods (behaviors) together within a class. It helps in hiding the internal workings of an object and only exposing necessary information.
- **INHERITANCE** allows a new class (subclass) to inherit attributes and behaviors from an existing class (superclass). This promotes code reusability and helps in creating a hierarchy of classes.
- **POLYMORPHISM** allows objects of different classes to be treated as objects of a common superclass. This means that different objects can respond to the same message (function call) in different ways.
- **METHOD OVERRIDING:** In method overriding, a subclass can provide a specific implementation for a method that is already defined in its superclass. This allows the subclass to customize or extend the behavior of the inherited method.

- **METHOD OVERLOADING**: Method overloading enables a class to have multiple methods with the same name but different parameters. This allows developers to create methods that perform similar tasks but with different inputs, making the code more flexible and easier to use.
- **CONSTRUCTORS AND DESTRUCTORS**: Constructors are special methods used to initialize objects when they are created. They typically set initial values for object attributes. Destructors, on the other hand, are used to clean up resources and perform necessary actions before an object is destroyed or goes out of scope.

EXAMPLE OF KEY FEATURES (PYTHON)

CLASSES and OBJECTS

Let's consider using built-in, primitive data structures as an option and alternative to **Classes.** Primitive data structures - like numbers, strings, and lists - are designed to represent straightforward pieces of information, such as the cost of an apple, the name of a poem, or your favorite colors, respectively. But if we want to represent something more complex, we have to use **Classes.**

CLASSES & OBJECTS	
description	**example of code**

Primitive data structures
For example, we might want to track employees in an organization. We need to store some basic information about each employee, such as their name, age, position, and the year they started working
	One way to do this is to represent each employee as a list
	Python
	`kirk = ["James Kirk", 34, "Captain", 2265]`
	`spock = ["Spock", 35, "Science Officer", 2254]`
	`mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]`

Issues with Primitive data structures From the previous example, there are several issues with this approach: - make larger code files more difficult to manage. - can introduce errors if employees don't have the same number of elements in their respective lists.	In the **mccoy** list above, the age is missing, so **mccoy[1]** will return **"Chief Medical Officer**" instead of **Dr. McCoy's** age. So, a great way to make this type of code more manageable and more maintainable is to use classes.

Define a class by using the «***class***» keyword followed by a name and a colon. Then we ***use .__init__()*** to declare which attributes each instance of the class should have.

We define a class Employee with attributes for name, age, position, and start year. The __init__ method is used to initialize these attributes when an object of the class is created. We also define a method display_info to print out the information about an employee.

An object is called an instance of a class. Suppose Employee is a class then we can create objects
like employee1, employee2, etc from the class.

So, I create two instances of the Employee class (employee1 and employee2) and display their information using the display_info method.

```
class Employee:
    def __init__(self, name, age, position,
start_year):
        self.name = name
        self.age = age
        self.position = position
        self.start_year = start_year

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Position: {self.position}")
        print(f"Start Year: {self.start_year}")

# Create instances of Employee class
employee1 = Employee("John Doe", 30, "Software
Engineer", 2015)
employee2 = Employee("Jane Smith", 25, "Data
Analyst", 2018)

# Display information about employees
print("Employee 1:")
employee1.display_info()

print("\nEmployee 2:")
employee2.display_info()
```

```
#out put
Employee 1:
Name: John Doe
Age: 30
Position: Software Engineer
Start Year: 2015

Employee 2:
Name: Jane Smith
Age: 25
Position: Data Analyst
Start Year: 2018
```

Using classes in Python helps organize related data and functionality into a single unit, making the code more manageable and maintainable. It also allows for code reusability and abstraction, leading to cleaner and more structured code.

INHERITANCE

The **process of inheriting the properties of the parent class into a child class is called inheritance**. Using inheritance in this way allows us to reuse code from the base class and extend it in the derived class, providing a more structured and organized approach to managing employee data.

The example below will demonstrate how to use inheritance in Python. The case is Create a base class **Person** that stores common information about a person, and a **derived class Employee** that **inherits** from **Person** and includes additional information specific to an employee

INHERITANCE	
description	**example of code**

In this code, we define a base class Person with attributes for name and age. The Employee class is derived from the Person class and adds attributes for position and start year. The __init__ method in the Employee class calls the __init__ method of the base class using super().
 	
Also, override the display_info method in the Employee class to include information specific to an employee while still displaying the common information from the base class
 	
.
	Python
	`class Person:`
	`  def __init__(self, name, age):`
	`  self.name = name`
	`  self.age = age`
	
	`  def display_info(self):`
	`  print(f"Name: {self.name}")`
	`  print(f"Age: {self.age}")`
	
	`class Employee(Person):`
	`  def __init__(self, name, age, position, start_year):`
	`  super().__init__(name, age)`
	`  self.position = position`
	`  self.start_year = start_year`
	
	`  def display_info(self):`
	`  super().display_info()`
	`  print(f"Position: {self.position}")`
	`  print(f"Start Year: {self.start_year}")`
	
	`# Create instances of Employee class`
	`employee1 = Employee("John Doe", 30, "Software Engineer", 2015)`
	`employee2 = Employee("Jane Smith", 25, "Data Analyst", 2018)`
	
	`# Display information about employees`
	`print("Employee 1:")`
	`employee1.display_info()`
	
	`print("\nEmployee 2:")`
	`employee2.display_info()`

	#out put Employee 1: Name: John Doe Age: 30 Position: Software Engineer Start Year: 2015 Employee 2: Name: Jane Smith Age: 25 Position: Data Analyst Start Year: 2018

ABSTRACTION

Abstraction in Python involves hiding the complex implementation details of a class and only exposing the necessary information or functionality to the outside world. This helps in simplifying the usage of the class and focuses on what the class does rather than how it does it. In the context of tracking employees in an organization, we can use abstraction to define a base class with abstract methods that represent the common functionality every employee should have, without providing the specific implementation. Subclasses can then inherit from this base class and provide their own implementations for these abstract methods.

ENCAPSULATION

Encapsulation in Python involves bundling the data (attributes) and methods (functions) that operate on the data within a class, and restricting access to the internal data by using access modifiers such as public, private, and protected. This helps in preventing external code from directly modifying the internal state of an object and promotes data hiding and information hiding.

Let's demonstrate encapsulation in Python with an example of tracking employees in an organization:

ENCAPSULATION

description

In this example, we define a class Employee with attributes _name, _age, _position, and __start_year. The attributes _name, _age, and _position are marked as protected by prefixing them with a single underscore. The attribute __start_year is marked as private by prefixing it with double underscores.

We provide a method display_info to display the employee information, and getter and setter methods get_start_year and set_start_year to access and modify the private attribute __start_year.

When we try to directly access or modify the private attribute __start_year, Python raises an AttributeError. Instead, we use the getter and setter methods to interact with the private attribute, ensuring encapsulation and controlled access to the internal state of the object.

example of code

```
class Employee:
    def __init__(self, name, age, position, start_year):
        self._name = name  # Protected attribute
        self._age = age    # Protected attribute
        self._position = position  # Protected attribute
        self.__start_year = start_year  # Private attribute

    def display_info(self):
        print("Employee Information:")
        print(f"Name: {self._name}")
        print(f"Age: {self._age}")
        print(f"Position: {self._position}")
        print(f"Start Year: {self.__start_year}")

    def get_start_year(self):
        return self.__start_year

    def set_start_year(self, start_year):
        if start_year > 0:
            self.__start_year = start_year
        else:
            print("Invalid start year. Please provide a valid
year.")

# Create an instance of the Employee class
employee1 = Employee("John Doe", 30, "Software Engineer", 2015)

# Display information about the employee
employee1.display_info()

# Try to access and modify private attribute directly (will
result in an error)
# print(employee1.__start_year)
# employee1.__start_year = 2016

# Access and modify private attribute using getter and setter
methods
print("\nUpdating start year...")
employee1.set_start_year(2016)
print(f"New Start Year: {employee1.get_start_year()}")



#output
Employee Information:
Name: John Doe
Age: 30
Position: Software Engineer
Start Year: 2015

Updating start year...
New Start Year: 2016
```

POLYMORPHISM

Polymorphism is another important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios. The built-in function l**en()** calculates the length of an object depending upon its type. If an object is a string (x=«banana»), it returns the count of characters (x=6), and If an object is a list (a), it returns the count of items in a list(a=3).

.

```
# Polymorphism
# Capable of working with data of “many forms”

x = 'banana' # string 6
a = ["apple", "banana", "cherry"] #list of 3
b = {"name" : "John", "age" : 36} #  2

print(len(x))
print(len(a))
print(len(b))
```

In the context of tracking employees in an organization, we can create a superclass called Employee that contains common attributes and methods shared by all employees. Then, we can create subclasses for different types of employees, such as Manager, Engineer, and Intern, which inherit from the Employee superclass.

Here's an example to illustrate polymorphism in Python for tracking employees. By using polymorphism, we can treat instances of Manager, Engineer, and Intern as instances of the Employee superclass. This allows us to call the display_info() method on any employee object, regardless of its specific subclass, and get the relevant information displayed.

POLYMORPHISM	
description	**example of code**

In this example, we define a superclass Employee with attributes name, age, position, and start_year, as well as a method display_info() to print out the information about an employee. We then create subclasses Manager, Engineer, and Intern, each with their own specific implementation of the __init__ method.	Python
	`class Employee:`
	`  def __init__(self, name, age, position, start_year):`
	`  self.name = name`
	`  self.age = age`
	`  self.position = position`
	`  self.start_year = start_year`
	
	`  def display_info(self):`
	`  print(f"{self.name} ({self.position}) - Age: {self.age}, Started in {self.start_year}")`
	
	`class Manager(Employee):`
	`  def __init__(self, name, age, start_year):`
	`  super().__init__(name, age, "Manager", start_year)`
	
	`class Engineer(Employee):`
	`  def __init__(self, name, age, start_year):`
	`  super().__init__(name, age, "Engineer", start_year)`
	
	`class Intern(Employee):`
	`  def __init__(self, name, age, start_year):`
	`  super().__init__(name, age, "Intern", start_year)`
	
	`# Create instances of different types of employees`
	`manager = Manager("Alice", 35, 2010)`
	`engineer = Engineer("Bob", 28, 2015)`
	`intern = Intern("Eve", 22, 2021)`
	
	`# Display information about each employee`
	`manager.display_info()`
	`engineer.display_info()`
	`intern.display_info()`

U3_1.2 Explain the importance of encapsulation, inheritance and polymorphism in object-oriented programming.

Encapsulation

Helps in data hiding, reducing complexity, and providing a clear interface for interacting with the class. It ensures that data is accessed and modified in a controlled manner, enhancing security and maintainability.

- **Security -** protects an object from unauthorized access
- **Simplicity -** keeping classes separated and preventing them from tightly coupling with each other.
- **Aesthetics -** Bundling data and methods within a class makes code more readable and maintainable

Inheritance

Inheritance reduces code duplication by allowing derived classes to reuse the properties and methods defined in the base class. Developers don't have to write the same code multiple times in different classes when a common functionality exists.

This mechanism fosters code reusability, allowing the derived class to leverage the attributes and methods of the base class. It establishes a hierarchical relationship, promoting a structured and organized codebase. Inheritance enables the creation of specialized classes that build upon the functionality of more generalized ones, enhancing modularity and easing maintenance.

Inheritance is the process of extending the existing code functionality for removing the repetitive coding work, as a result, it leads to reduced development time.

Polymorphism

Polymorphism means “one name many forms” that allow developers to provide multiple elements depending on the object type. This will permit developers to redefine the whole work and define how it can be done by updating the parts in which it was previously performed. Polymorphism terms are known as overriding and overloading.

Method Overriding enhances code extensibility, allowing developers to tailor the functionality of inherited methods to suit the specific requirements of individual subclasses. This dynamic behaviour during runtime contributes to the power and versatility of OOPs.

Overloaded methods provide a cleaner, more readable codebase, as developers can invoke the same method name with different argument sets.

This feature streamlines the development process, making it easier to work with diverse data types or handle various scenarios without cluttering the code with multiple method names. Method overloading is a critical element in OOPs, contributing to code versatility and maintainability.

U3_2.1 Demonstrate the use of object-oriented tools and techniques.

During my study of Python, I developed the «Louise Game». I'd like to show below examples of the use of these key features (also there are some of examples in the previous paragraph 1.1 «The key features of object-oriented programming.

Below introduced a few pieces of code from the project «Luisa game».

In the example I have created a PlayerDatabase class with method read_players_from_file to the PlayerDatabase class that reads player information from a file and adds it to the database. The file is assumed to have player information separated by a comma.

```
#-------------------Authorization--------------------------------------
class PlayerDatabase:
  def __init__(self):
      self.players = {}
 # save the player names and passwords, and then compare them with the name given by the player:
  def add_player(self, name, password):
      self.players[name] = password

  def read_players_from_file(self, filename):
      with open(filename, 'r') as file:
          for line in file:
              name, password = line.strip().split(',')
              self.add_player(name, password)

  #Function find the login and password in the file
  def check_player(self, name, password):
    if name in self.players and self.players[name] == password:
      return True
    else:
      return False

#  Useof PlayerDatabase for validation

db = PlayerDatabase()
db.read_players_from_file('NamesandPasswords.txt')
```

Functions in Python. I have used my functions in programming to bundle a set of instructions I want to use repeatedly:

In a card desk I have got FullCardslist = ['Red#1','Red#2','Black#11','Yellow#1']. With function pop(0) FullCardslist.pop(0). I create a variable colling Card, where Card = Color#Namber - it is a string for example Red#1. With method split() I've got ['Red','1'] (Card.split("#", 2)). As resalt I can receive color and number Color = print(Card[0]) , Namber = print(Card[1])

```
#Start Game Function-------------------------------------------------------------
def StartNewGame():

  while len(FullCardslist) > 0:

    print("number of len",len(FullCardslist))
    ColorCard_P1 = ""
    ColorCard_P2 = ""
    # remove card of the top of deck/list (which is 0)
    CardPl1 = FullCardslist.pop(0)
    CardPl2 = FullCardslist.pop(0)
    # Ind = 0 - for picking Color up
    ColorCard_P1 = RecognizeCard(CardPl1,0)
    ColorCard_P2 = RecognizeCard(CardPl2,0)
    # Ind = 1 - for picking Number up
    NumberCard_P1 = int(RecognizeCard(CardPl1,1))
    NumberCard_P2 = int(RecognizeCard(CardPl2,1))
    #print("round-------",round)
    if ColorCard_P1 == ColorCard_P2:
      # Compare card's numbers
      print("Compare card's numbers",ColorCard_P1,ColorCard_P2,NumberCard_P1,NumberCard_P2 )
      if NumberCard_P1 > NumberCard_P2:
        # create a list by adding 2 cards to Player1's list
        PlayerList1.append(CardPl1)
        PlayerList1.append(CardPl2)
      else :
        # create a list by adding 2 cards to Player2's list
        PlayerList2.append(CardPl1)
        PlayerList2.append(CardPl2)

    else :
      # Find out winning COLOR according to the table
```

```
# this function returns the color/number of each card chosen by the player
# Ind = 0 - for picking Color up
# Ind = 1 - for picking Number up
def RecognizeCard(strCard,Ind):

  # split use for string
  lstCard = strCard.split("#", 2) # get back a List
  #print(lstCard)
  Result = lstCard[Ind] # pick up the Color/Number according to Ind
  #print(Result)
  return Result
```

U3_3.1 Design an object-oriented program

LOUISE'S GAME.

Louise is creating a card game for two players.

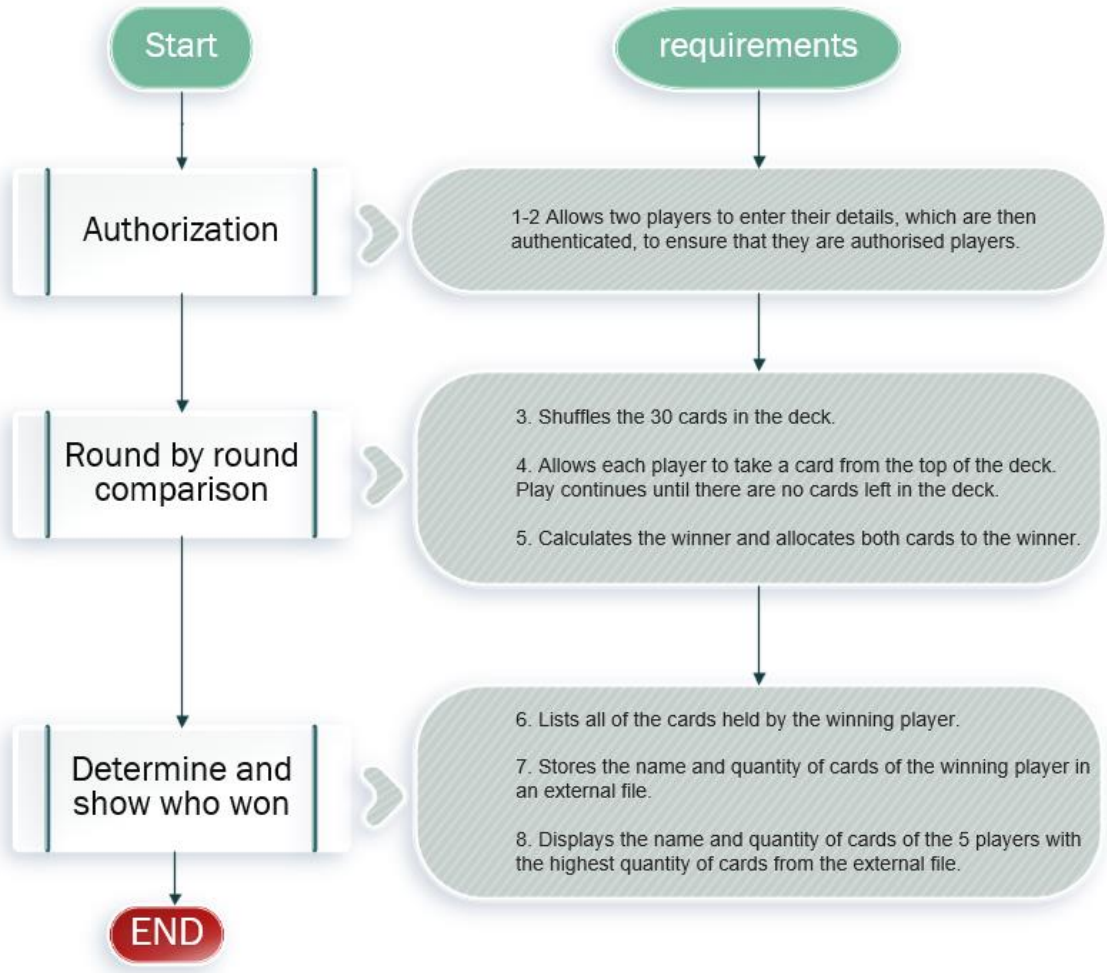
Basic parameters		Additional information
Number of players	**2**	Only authorised players are allowed to play the game.
Number cards	**30**	The 30 cards are shuffled and stored in the deck.
Each card	unique	Each card has one colour (red, black or yellow). Each card has a number (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) for each colour.
Cards colour's	Red Black Yellow	**RED beat BLACK** **BLACK beat YELLOW** **YELLOW beat RED**

How to play

- ✓ Player 1 takes the top card from the deck.
- ✓ Player 2 takes the next card from the deck.
- ✓ If both players have a card of the same colour, the player with the highest number wins.
- ✓ If both players have cards with different colours, the winning colour is shown in the table.
- ✓ The winner of each round keeps both cards.
- ✓ The players keep playing until there are no cards left in the deck.

Additional requirements

- ✓ Lists all of the cards held by the winning player.
- ✓ Stores the name and quantity of cards of the winning player in an external file.
- ✓ Displays the name and quantity of cards of the 5 players with the highest quantity of cards from the external file.



A graphical representation of an algorithm was done using flowchart tools (Lucidchart and MsVisio). A flowchart illustrates the steps of a program graphically.

The following design was created according to the given conditions.

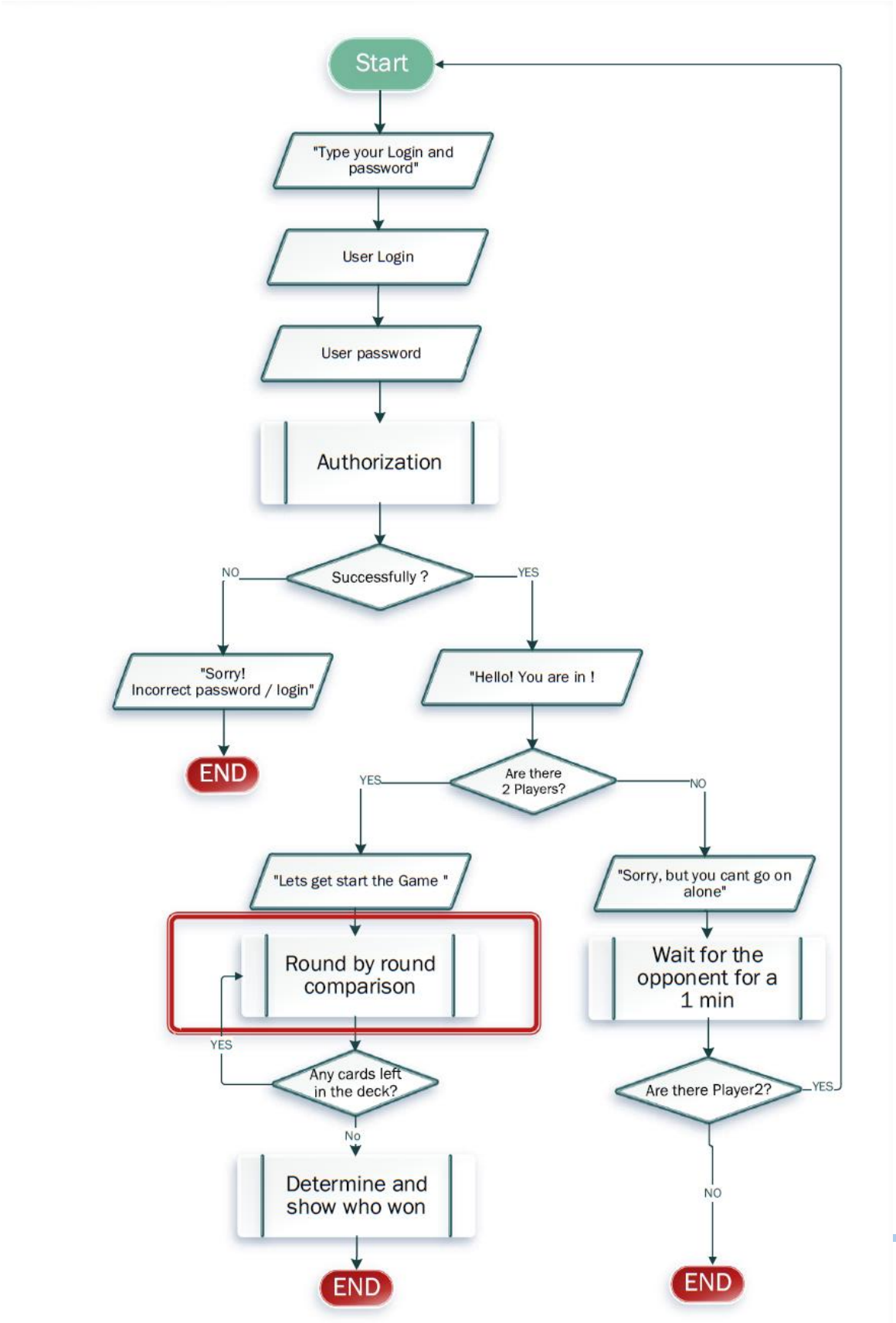
The whole game is divided into three large blocks, in each of which certain steps meet the requirements.

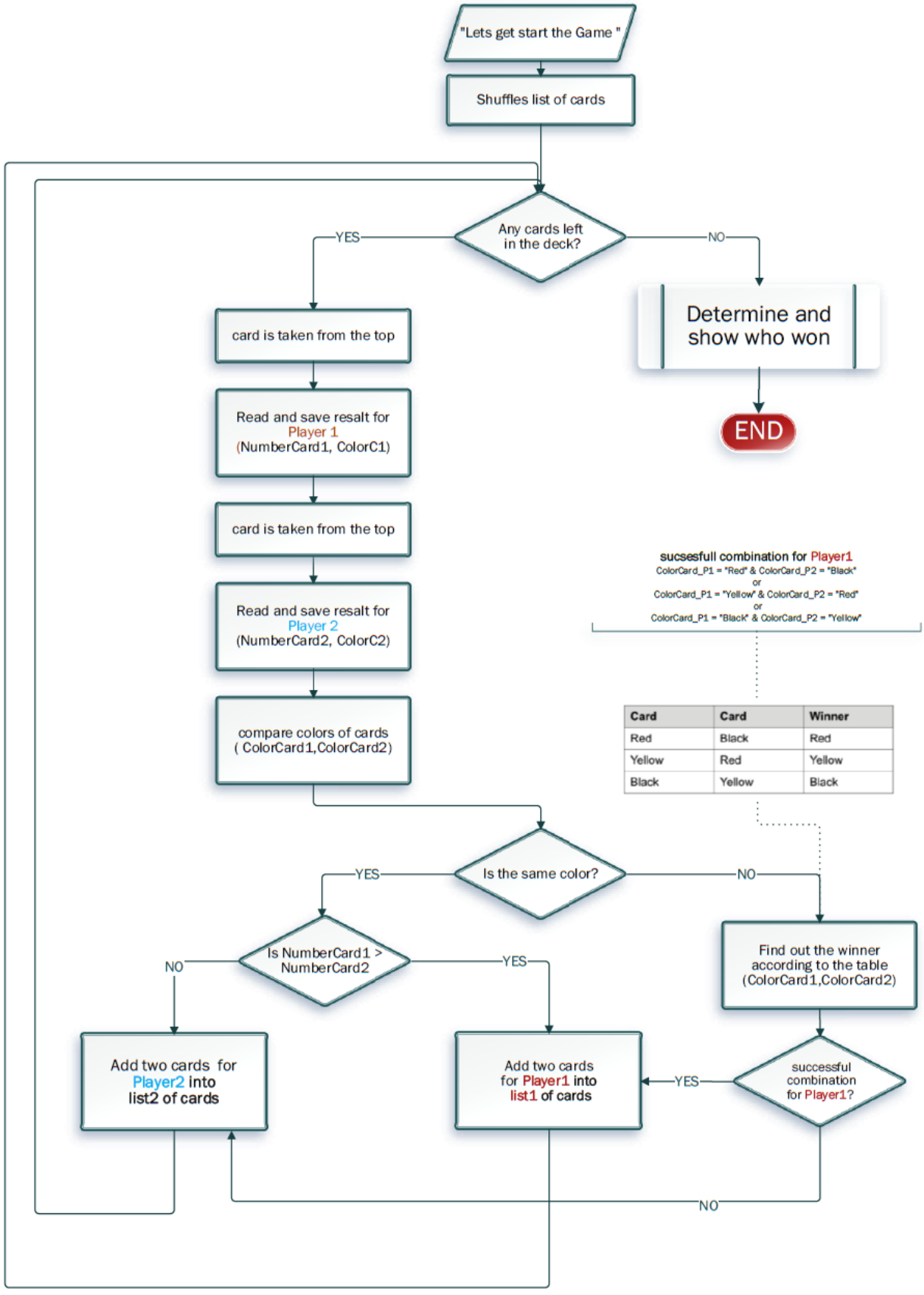
Game's design consists of three pictures/ flowcharts.

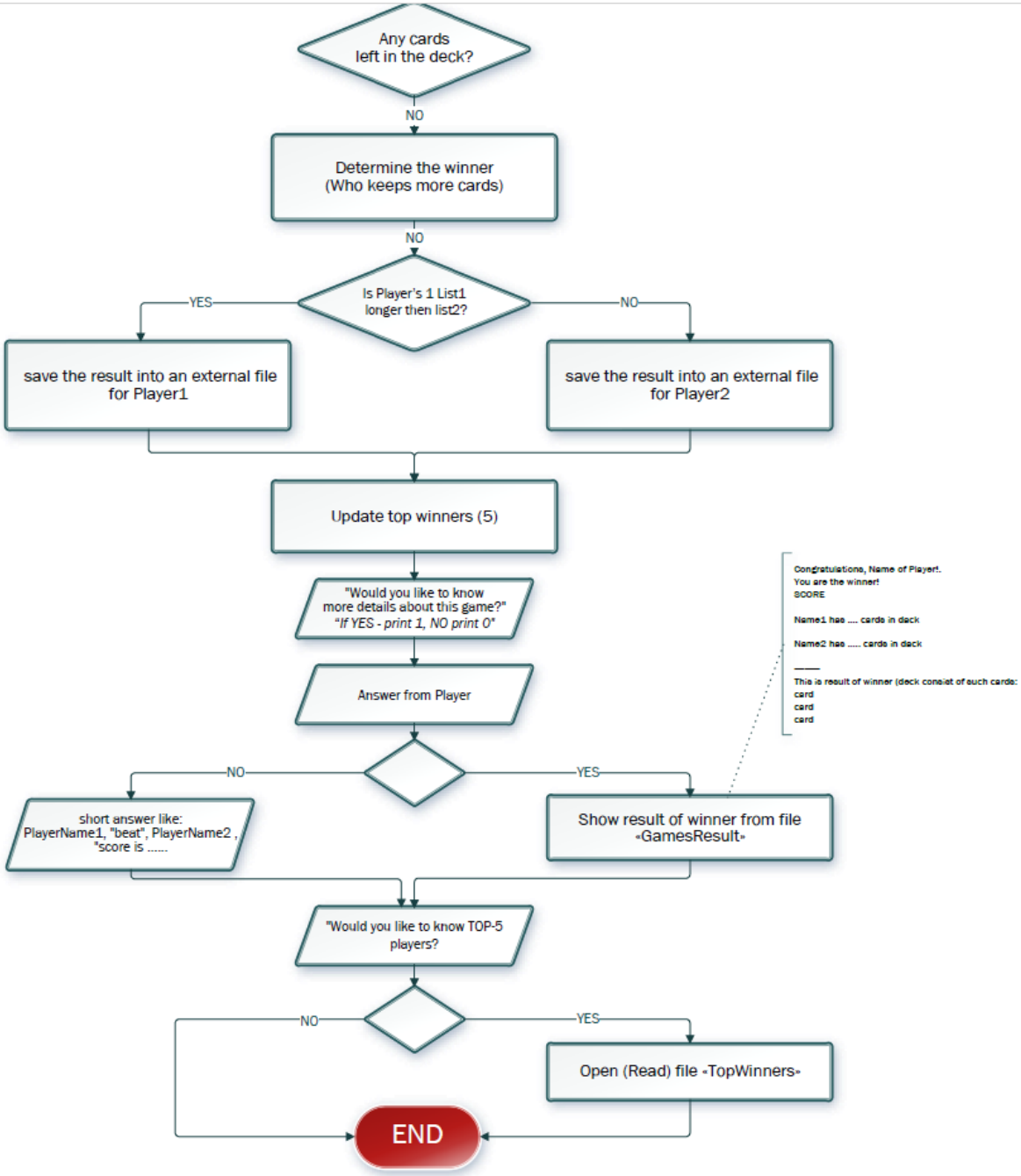
The first picture below shows all the main events of the game.

The next picture shows details for the second block «Round by round comparison», finally third picture – explains the block «Determine and show who won»

The authorization part in the current version is simple, so it is not presented separately in the document. however, for the future, I would like to add the ability to join new players, as well as check whether such a player has already registered, as well as check the reliability (length) of the password


Start
"Type your Login and password"
User Login
User password
Authorization
Successfully ?
NO
YES
"Sorry! Incorrect password / login"
END
"Hello! You are in !
Are there 2 Players?
YES
NO
"Lets get start the Game "
Round by round comparison
YES
Any cards left in the deck?
No
Determine and show who won
END
"Sorry, but you cant go on alone"
Wait for the opponent for a 1 min
Are there Player2?
YES
NO
END


"Lets get start the Game "
Shuffles list of cards
Any cards left in the deck?
YES
NO
Determine and show who won
END
card is taken from the top
Read and save resalt for Player 1 (NumberCard1, ColorC1)
card is taken from the top
Read and save resalt for Player 2 (NumberCard2, ColorC2)
compare colors of cards (ColorCard1,ColorCard2)
sucsesfull combination for Player1
ColorCard_P1 = "Red" & ColorCard_P2 = "Black"
or
ColorCard_P1 = "Yellow" & ColorCard_P2 = "Red"
or
ColorCard_P1 = "Black" & ColorCard_P2 = "Yellow"
Card | Card | Winner
Red | Black | Red
Yellow | Red | Yellow
Black | Yellow | Black
Is the same color?
YES
NO
Is NumberCard1 > NumberCard2
NO
YES
Find out the winner according to the table (ColorCard1,ColorCard2)
Add two cards for Player2 into list2 of cards
Add two cards for Player1 into list1 of cards
successful combination for Player1?
YES
NO


Any cards left in the deck?
NO
Determine the winner (Who keeps more cards)
NO
Is Player's 1 List1 longer then list2?
YES
NO
save the result into an external file for Player1
save the result into an external file for Player2
Update top winners (5)
"Would you like to know more details about this game?" "If YES - print 1, NO print 0"
Answer from Player
NO
YES
short answer like: PlayerName1, "beat", PlayerName2 , "score is .....
Show result of winner from file «GamesResult»
Congratulations, Name of Player!.
You are the winner!
SCORE
Name1 has .... cards in dack
Name2 has ..... cards in dack
This is result of winner (deck consist of such cards:
card
card
card
"Would you like to know TOP-5 players?
NO
YES
Open (Read) file «TopWinners»
END

U3_3.2 Develop an object-oriented program.

Using Python programming language I developed a working prototype of [Louise's game.]

Below you can see parts of the code that were executed using Replit, also there are explanations for how initially designed the algorithm and then later tested the solution (U3_4.1).

```
#--------------------------------Game-----------------------------------
#Start Game Function-------------------------------------------------------
def StartNewGame():
  while len(FullCardslist) > 0:
    print("number of len",len(FullCardslist))
    ColorCard_P1 = ""
    ColorCard_P2 = ""
    # remove card of the top of deck/list (which is 0)
    CardPl1 = FullCardslist.pop(0)
    CardPl2 = FullCardslist.pop(0)
    # Ind = 0 - for picking Color up
    ColorCard_P1 = RecognizeCard(CardPl1,0)
    ColorCard_P2 = RecognizeCard(CardPl2,0)
    # Ind = 1 - for picking Number up
    NumberCard_P1 = int(RecognizeCard(CardPl1,1))
    NumberCard_P2 = int(RecognizeCard(CardPl2,1))
    #print("round-------",round)
    if ColorCard_P1 == ColorCard_P2:
      # Compare card's numbers
      print("Compare card's numbers",ColorCard_P1,ColorCard_P2,NumberCard_P1,NumberCard_P2 )
      if NumberCard_P1 > NumberCard_P2:
        # create a list by adding 2 cards to Player1's list
        PlayerList1.append(CardPl1)
        PlayerList1.append(CardPl2)
      else :
        # create a list by adding 2 cards to Player2's list
        PlayerList2.append(CardPl1)
        PlayerList2.append(CardPl2)
    else :
      # Find out the winning COLOR according to the table
      #successful combination for Player1
      if (ColorCard_P1 == "Red" and ColorCard_P2 == "Black") or (ColorCard_P1 == "Yellow" and
      ColorCard_P2 == "Red") or (ColorCard_P1 == "Black"
      and ColorCard_P2 == "Yellow"):
        print("successful combination for Player1 ", ColorCard_P1, ColorCard_P2)

        PlayerList1.append(CardPl1)
        PlayerList1.append(CardPl2)
        #sucsesfull combination for Player1
        #RED beat BLACK
        #YELLOW beat Red
        #BLACK beat Yellow

      else:
      #sucsesfull combination for Player2
        print("successful combination for Player2 ", ColorCard_P1, ColorCard_P2)
        PlayerList2.append(CardPl1)
        PlayerList2.append(CardPl2)
  print("=========================================Who is winner in game ")
```

Output:

```
type your Name hereLana
type your password here123
Hello and welcome back, Lana!
Sorry, but you can't go on alone
You are the second player. For successful login
type your Name hereSue
type your password here123
Let's get start the Game
Lana  against  Sue
number of len 30
Compare card's numbers Black Black 3 7
number of len 28
successful combination for Player2  Black Red
number of len 26
successful combination for Player1  Yellow Red
number of len 24
successful combination for Player1  Yellow Red
number of len 22
successful combination for Player2  Red Yellow
number of len 20
successful combination for Player2  Black Red
number of len 18
successful combination for Player2  Red Yellow
number of len 16
Compare card's numbers Black Black 8 6
number of len 14
successful combination for Player2  Yellow Black
number of len 12
successful combination for Player1  Yellow Red
number of len 10
successful combination for Player1  Red Black
number of len 8
Compare card's numbers Yellow Yellow 8 2
number of len 6
successful combination for Player2  Red Yellow
number of len 4
Compare card's numbers Black Black 4 2
number of len 2
successful combination for Player2  Red Yellow
==========================================Who is winner in game
['Yellow#5\n', 'Red#7\n', 'Yellow#9\n', 'Red#8\n', 'Black#8\n', 'Black#6\n', 'Yellow#6\n', 'Red#2\n', 'Red#10\n', 'Black#1\
n', 'Yellow#8\n', 'Yellow#2\n', 'Black#4\n', 'Black#2\n'] ['Black#3\n', 'Black#7\n', 'Black#10\n', 'Red#3\n', 'Red#1\n', 'Y
ellow#7\n', 'Black#5\n', 'Red#6\n', 'Red#5\n', 'Yellow#10\n', 'Yellow#1\n', 'Black#9\n', 'Red#4\n', 'Yellow#4\n', 'Red#9\n'
, 'Yellow#3\n']
SaveFile
Would you like to know more details about this game?
If YES - print 1, NO print 01
1
Congratulations, Sue!. You are the winner!
SCORE

Sue has 16 cards in dack

Lana has 14 cards in dack

----------
This is result of winner (deck consist of such cards:)
Black#3

Black#7

Black#10

Red#3

Red#1
```

```
154   print("=========================================Who is winner in game ")
155   print(PlayerList1,PlayerList2)
156   if len(PlayerList1) > len(PlayerList2): # winner is Player1
157     # save the result into an external file for Player1
158       print("SaveFile")
159       SaveExternalFile(PlayerName1,PlayerList1,PlayerName2,PlayerList2)
160       update_top_winners(PlayerName1,PlayerList1)
161
162   else:
163    # save the result into an external file for Player2
164       print("SaveFile")
165       SaveExternalFile(PlayerName2,PlayerList2,PlayerName1,PlayerList1)
166       update_top_winners(PlayerName2,PlayerList2)
167           # Who is winner in game
168
169             #open and read the file:
170
171   print("Would you like to know more details about this game?")
172
173   AnswerDetails = input("If YES - print 1, NO print 0")
174   if int(AnswerDetails) == 1:
175     #full details
176     print(AnswerDetails)
177     GamesResult = open("GamesResult.txt", "r")
178     print(GamesResult.read())
179
180   else :
181      print(AnswerDetails)
182       #short text, or read first 3 lines from code
183      print (PlayerName2, "beat", PlayerName1 , "score is ", len(PlayerList2)," : ", len(PlayerList1))
184
185   print("Would you like to know TOP-5 players?")
186   AnswerTop = input("If yes - print 1, else print 0")
187   if AnswerTop == "1":
188      TopWinners = open("TopWinners.txt", "r")
189      print(TopWinners.read())
190
191
192   print("Game over")
193   answerReStart = input("Would you like to play new game? If yes - print 1, alse print 0")
194   if answerReStart == "1" :
195     StartNewGame()
196   else:
197     print("The END")
198
199 #END Game Function-----------------------------------------------------------------
```

Output

```
Would you like to know more details about this game?
If YES - print 1, NO print 00
0
Sue beat Lana score is  16  :  14
Would you like to know TOP-5 players?
If yes - print 1, else print 00
Game over
Would you like to play new game? If yes - print 1, alse print 00
The END
```

U3_4.1 Test an object oriented program.

Manual testing is a type of software testing where testers manually execute test cases without using any automation tools. Manual testing involves human intervention to identify defects in the software application.

Date of test	Component to be tested	Type of test to be carried out	Prerequisites and dependencies
	Login with Username and Password	White box testing Analyse the code	Password should belong to Username which will be used for entering to the game
	Login with Username and Password	Black box Alpha Testing	If Username is not in the list show the warning “You must be registered first”
	Game just for 2 gamers	White box testing Black box Alpha Testing	If there are only one player, then game should not start Waiting time is 2 min and then finish the game if there only 1 gamer
	Card game	White box testing	Make sure that the color and number belong to the same card

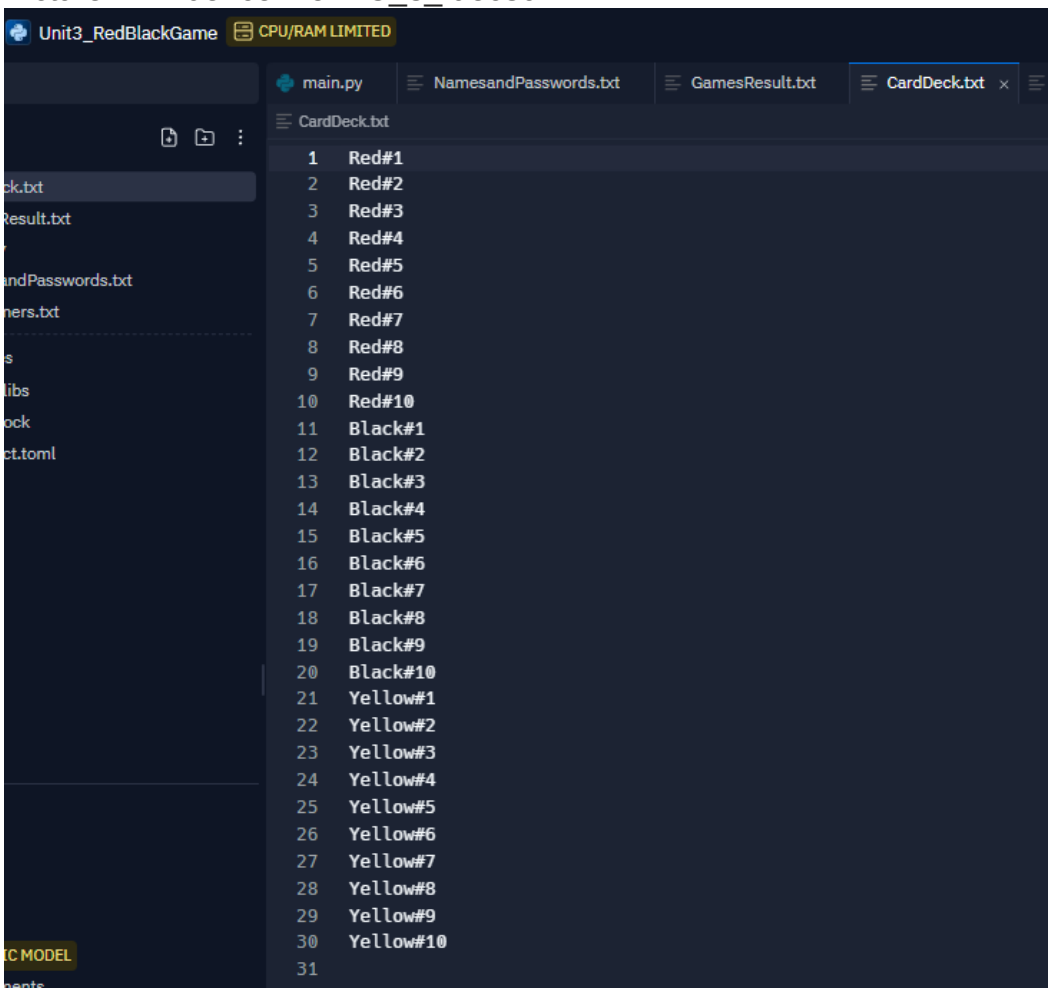
Here are some appropriate test data for the card game which cover various aspects of this game, including authorization, deck initialization, game play mechanics, and data storage/display requirements. These test cases match with part of the DATA REQUIREMENTS from the Requirement specification

ID	TEST CASES (TC)	EVIDENCE OF TESTING
PLAYER AUTHORIZATION:		
TC_1	Verify that an unauthorized player cannot access the game.	Test log in U3_4.2
TC_2	Verify that an authorized player can access the game.	Test log in U3_4.2
DECK INITIALIZATION:		
TC_3	Verify that the deck contains 30 unique cards. Steps 1. Check is there txt file with card as require 2. Include in code part with displaying how is process of shuffle going. 3. Run the game. 4. After proper testing this part of code can be commented	Picture1 Evidence for TC_3_deck30
TC_4	Verify that each card has a unique color and number. **Steps** 1. Open the CardDeck.txt before run the game	Picture2 Evidence for TC_4_UniqueCard Not necessary as CardDeck.txt created

ID	TEST CASES (TC)	EVIDENCE OF TESTING
	2. Make sure that deck consist of red, yellow and black cards with numbers from 1 to 10	according to the rule and should be consist of red, yellow and blac cards.
TC_5	Verify that the deck is shuffled and stored in a random order. **Steps** Option 1 (just wite boxing test) 1. Check just code – make sure that code include import of random and method shuffle. Option 2 (mix of wite and black boxing test) 1. Check is there txt file with card as require 2. Include in code part with displaying how is process of shuffle going. 3. Run the game. 4. After proper testing this part of code can be commented	Picture3 "Evidence for TC_5_ shuffled" Picture4 Evidence for TC_5_AfterShuffle"
GAMEPLAY MECHANICS		
TC_6	Verify the game determines the winner correctly when both players have cards of the same color with different numbers. 1. Include in the code part with displaying how is process going. 2. Check manually with the game rule	Picture5 Evidence for «TC_6_SameColor»
TC_7	Verify the game determines the winning color correctly when players have cards of different colors. 1. Include in the code part with displaying how is process going. 2. Check manually with the game rule	Picture6 Evidence for 'TC_7_DiffColor"
TC_8	Verify that the winning player keeps both cards after each round. 1. Include in the code part with displaying how is process going after each round 2. Check manually the external file "GamesResult" and the explanation from concole	Picture7 "Evidence for TC_8_WinList"
TC_9	Verify that the game ends when there are no cards left in the deck.	The same picture as for TC_8
DATA STORAGE AND DISPLAY		
TC_10	Verify that the system lists all the cards held by the winning player after each round.	The same picture as for TC_8
TC_11	Verify that the system stores the name and quantity of cards of the winning player in an external file.	The same picture as for TC_8
TC_12	Verify that the system displays the name and quantity of cards held by the top 5 players with the highest quantity of cards from the external file.	Picture8 "Evidence for TC_12_TopWin"

ID	TEST CASES (TC)	EVIDENCE OF TESTING

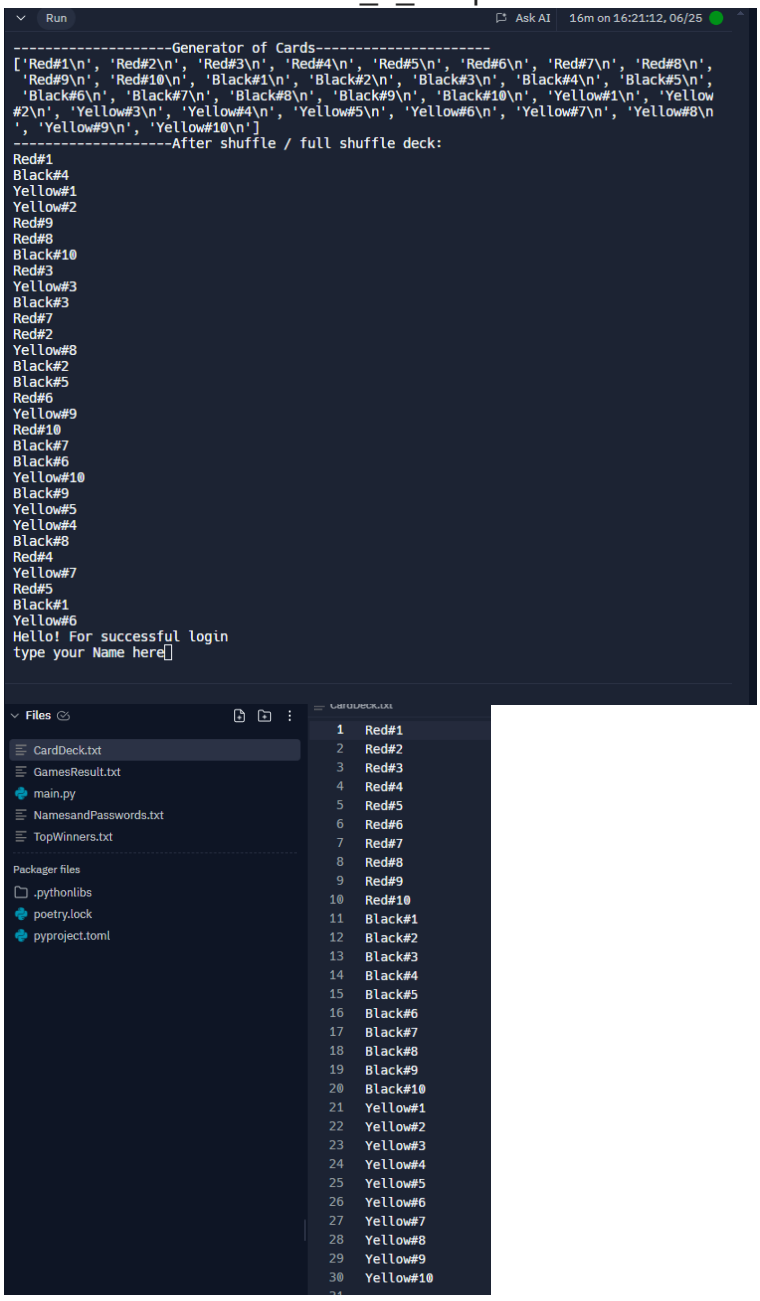
Picture1 “Evidence1 for TC_3_ dec30”



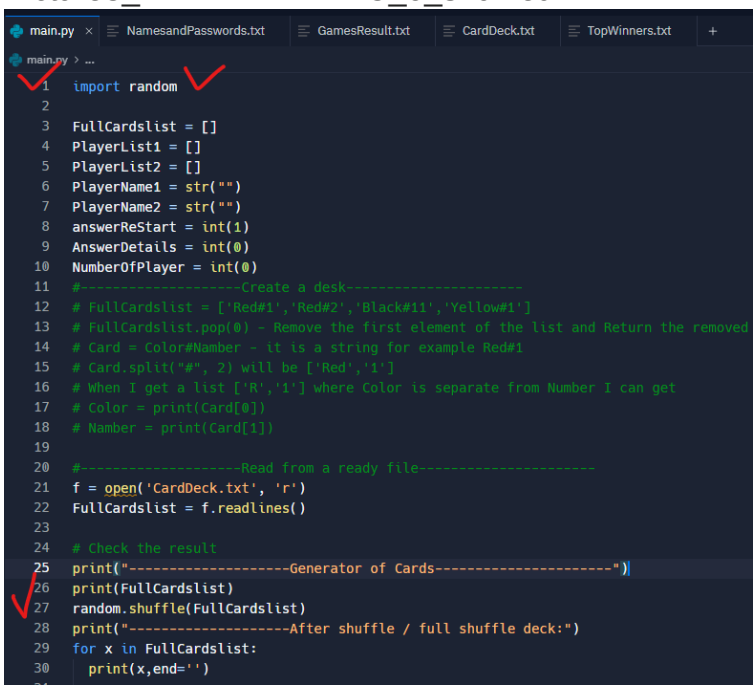

```
#--------------------Read from a ready file----------------------
f = open('CardDeck.txt', 'r')
FullCardslist = f.readlines()

# Check the result
print("--------------------Generator of Cards----------------------")
print(FullCardslist)
random.shuffle(FullCardslist)
print("--------------------After shuffle / full shuffle deck:")
for x in FullCardslist:
  print(x,end='')
```

Picture2 Evidence for TC_4_UniqueCard



Picture3_" Evidence for TC_5_shuffled"



```
import random

FullCardslist = []
PlayerList1 = []
PlayerList2 = []
PlayerName1 = str("")
PlayerName2 = str("")
answerReStart = int(1)
AnswerDetails = int(0)
NumberOfPlayer = int(0)
#-------------------Create a desk---------------------
# FullCardslist = ['Red#1','Red#2','Black#11','Yellow#1']
# FullCardslist.pop(0) - Remove the first element of the list and Return the removed
# Card = Color#Namber - it is a string for example Red#1
# Card.split("#", 2) will be ['Red','1']
# When I get a list ['R','1'] where Color is separate from Number I can get
# Color = print(Card[0])
# Namber = print(Card[1])

#-------------------Read from a ready file---------------------
f = open('CardDeck.txt', 'r')
FullCardslist = f.readlines()

# Check the result
print("--------------------Generator of Cards----------------------")
print(FullCardslist)
random.shuffle(FullCardslist)
print("--------------------After shuffle / full shuffle deck:")
for x in FullCardslist:
  print(x,end='')
```

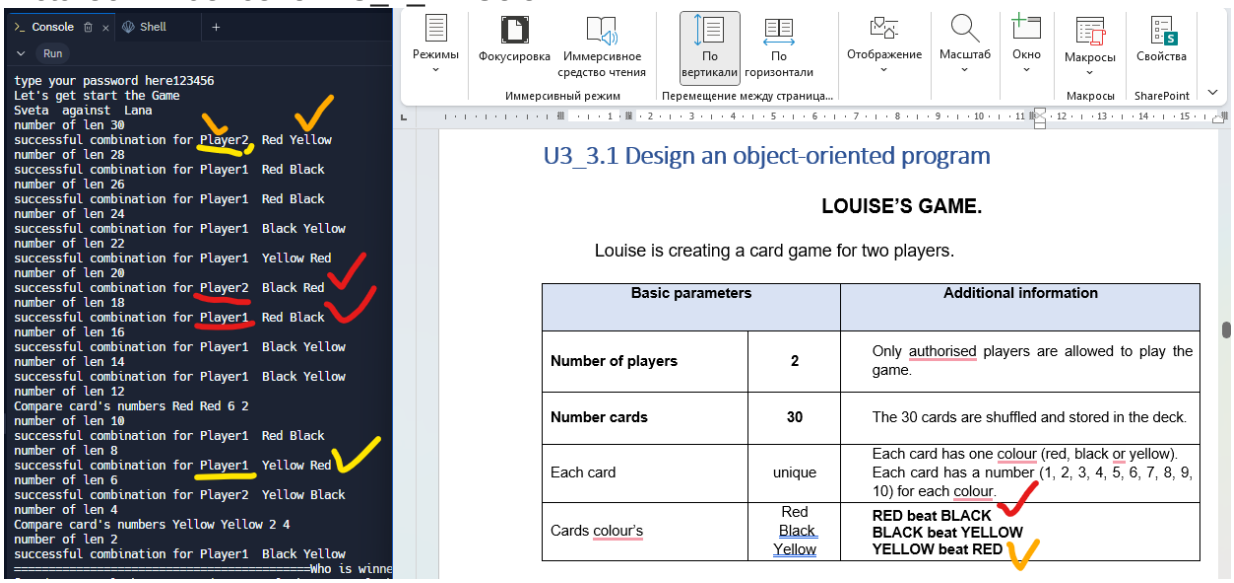
Picture4_"Evidence for TC_5_AfterShuffle"

```
--------------------Generator of Cards----------------------
['Red#1\n', 'Red#2\n', 'Red#3\n', 'Red#4\n', 'Red#5\n', 'Red#6\n', 'Red#7\n', 'Red#8\n', 'Red#9\n', 'Red#1
0\n', 'Black#1\n', 'Black#2\n', 'Black#3\n', 'Black#4\n', 'Black#5\n', 'Black#6\n', 'Black#7\n', 'Black#8\
n', 'Black#9\n', 'Black#10\n', 'Yellow#1\n', 'Yellow#2\n', 'Yellow#3\n', 'Yellow#4\n', 'Yellow#5\n', 'Yell
ow#6\n', 'Yellow#7\n', 'Yellow#8\n', 'Yellow#9\n', 'Yellow#10\n']
--------------------After shuffle / full shuffle deck:
Red#1
Yellow#3
Red#3
Black#2
Red#5
Black#4
Black#8
Yellow#10
Yellow#5
Red#4
Black#6
Red#10
Red#8
Black#10
Black#3
Yellow#7
Black#9
Yellow#1
Red#6
Red#2
Red#7
Black#5
Yellow#8
Red#9
Yellow#9
Black#1
Yellow#2
Yellow#4
Black#7
Yellow#6
Hello! For successful login
```

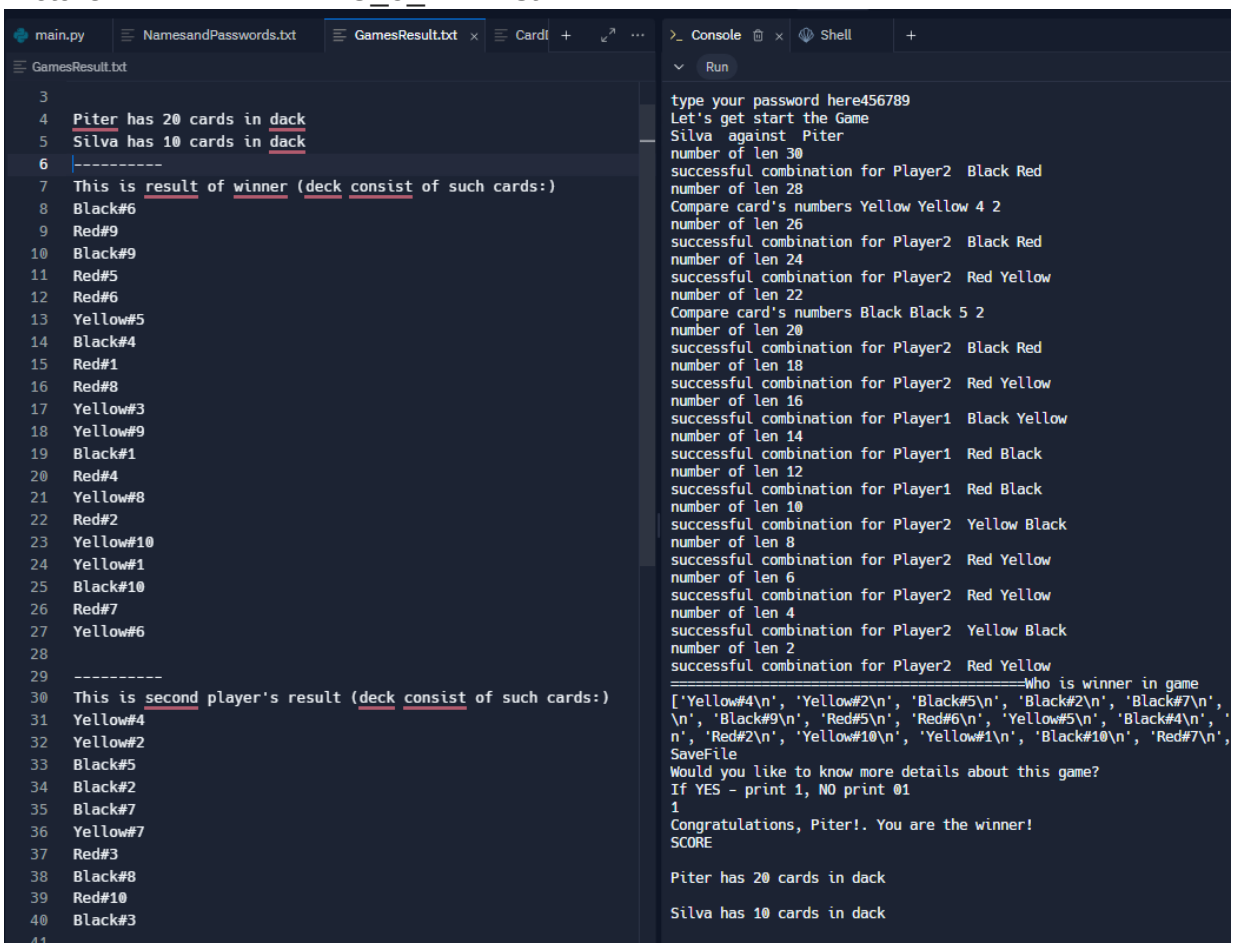
Picture5 “Evidence for TC_6_SameColor”

```
Let's get start the Game
Sveta  against  Lana
number of len 30
successful combination for Player2  Red Yellow
number of len 28
successful combination for Player1  Red Black
number of len 26
successful combination for Player1  Red Black
number of len 24
successful combination for Player1  Black Yellow
number of len 22
successful combination for Player1  Yellow Red
number of len 20
successful combination for Player2  Black Red
number of len 18
successful combination for Player1  Red Black
number of len 16
successful combination for Player1  Black Yellow
number of len 14
successful combination for Player1  Black Yellow
number of len 12
Compare card's numbers Red Red 6 2
number of len 10
successful combination for Player1  Red Black
number of len 8
successful combination for Player1  Yellow Red
number of len 6
successful combination for Player2  Yellow Black
number of len 4
Compare card's numbers Yellow Yellow 2 4
number of len 2
successful combination for Player1  Black Yellow
```

Picture6 “Evidence for TC_7_DiffColor”



Picture7 “Evidence for TC_8_ WinList”



GamesResult.txt

```
Piter has 20 cards in dack
Silva has 10 cards in dack
----------
This is result of winner (deck consist of such cards:)
Black#6
Red#9
Black#9
Red#5
Red#6
Yellow#5
Black#4
Red#1
Red#8
Yellow#3
Yellow#9
Black#1
Red#4
Yellow#8
Red#2
Yellow#10
Yellow#1
Black#10
Red#7
Yellow#6

----------
This is second player's result (deck consist of such cards:)
Yellow#4
Yellow#2
Black#5
Black#2
Black#7
Yellow#7
Red#3
Black#8
Red#10
Black#3
```

Console

```
type your password here456789
Let's get start the Game
Silva  against  Piter
number of len 30
successful combination for Player2  Black Red
number of len 28
Compare card's numbers Yellow Yellow 4 2
number of len 26
successful combination for Player2  Black Red
number of len 24
successful combination for Player2  Red Yellow
number of len 22
Compare card's numbers Black Black 5 2
number of len 20
successful combination for Player2  Black Red
number of len 18
successful combination for Player2  Red Yellow
number of len 16
successful combination for Player1  Black Yellow
number of len 14
successful combination for Player1  Red Black
number of len 12
successful combination for Player1  Red Black
number of len 10
successful combination for Player2  Yellow Black
number of len 8
successful combination for Player2  Red Yellow
number of len 6
successful combination for Player2  Red Yellow
number of len 4
successful combination for Player2  Yellow Black
number of len 2
successful combination for Player2  Red Yellow
==========Who is winner in game
['Yellow#4\n', 'Yellow#2\n', 'Black#5\n', 'Black#2\n', 'Black#7\n',
\n', 'Black#9\n', 'Red#5\n', 'Red#6\n', 'Yellow#5\n', 'Black#4\n', '
n', 'Red#2\n', 'Yellow#10\n', 'Yellow#1\n', 'Black#10\n', 'Red#7\n',
SaveFile
Would you like to know more details about this game?
If YES - print 1, NO print 01
1
Congratulations, Piter!. You are the winner!
SCORE

Piter has 20 cards in dack

Silva has 10 cards in dack
```

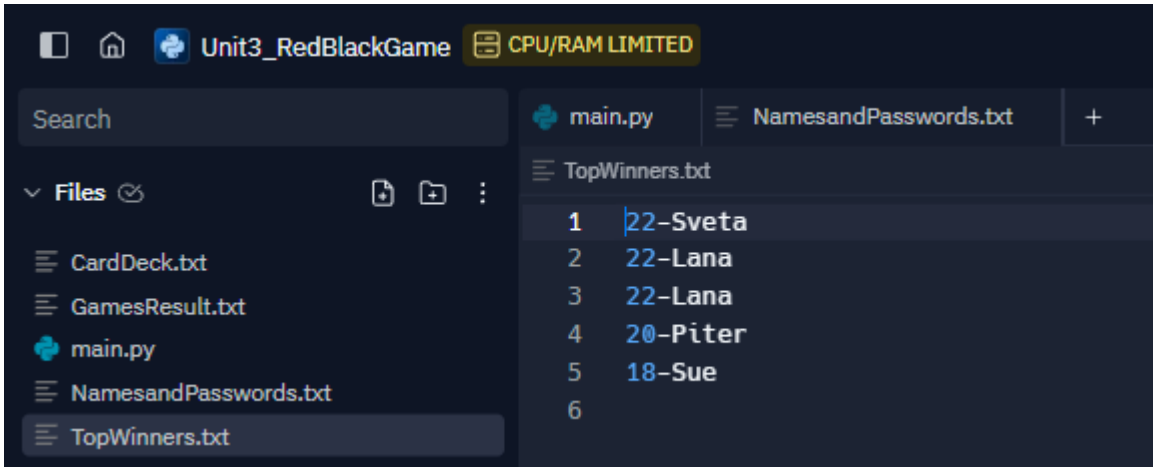
Picture8 “Evidence for TC_12_ TopWin”

```
Congratulations, Sveta!. You are the winner!
SCORE

Sveta has 22 cards in dack

Lana has 8 cards in dack
```

```
Piter beat Sue score is  14  :  16
Would you like to know TOP-5 players?
If yes - print 1, else print 01
22-Sveta
22-Lana
22-Lana
18-Sue
18-Sue
```


Unit3_RedBlackGame
CPU/RAM LIMITED
Search
Files
CardDeck.txt
GamesResult.txt
main.py
NamesandPasswords.txt
TopWinners.txt
main.py
NamesandPasswords.txt
TopWinners.txt
1 22-Sveta
2 22-Lana
3 22-Lana
4 20-Piter
5 18-Sue
6

```
Would you like to know TOP-5 players?
If yes - print 1, else print 01
22-Sveta
22-Lana
22-Lana
20-Piter
18-Sue

Game over
```

U3_4.2 Document appropriate action to correct errors.

Description of test	Test data to be used (if required)	Expected outcome	Pass / Fail	Comments and intended actions
AUTHORIZATION TESTING				
Normal data				
Enter acceptable username for player one	Lana	Entry allowed	Pass	Access was allowed but no conformation message Should be same kind of Hello and welcome back, Lana!
Enter acceptable password for player one	123456	Entry allowed	Pass	Explain why this prototype required entering of second player Like this Sorry, but you can't go on alone
Enter acceptable username for player one	Piter	Entry allowed	Pass	
Enter acceptable password for player one	456789	Entry allowed	Fail	Mistake is Invalid name for piter or password 456789 Clarify what exactly wrong For avoiding misunderstanding
Boundary inputs				
Enter non-acceptable username (one (Username 1 character)	L	Entry not allowed	Pass	Cancellation message needed
Enter non-acceptable username (Username No longer then 12)	Usernametest123	Entry not allowed.	Pass	Explanations about security name and password Username no longer than 12 characters and password longer than 3 characters.
Enter non-acceptable pasword.	Password Qwerty	Entry not allowed	Pass	Explanations about security name and password Password and Qwerty insecurity

Enter empty username. Invalid inputs		Entry not allowed	Pass	Cancellation message needed with explanation what exactly is wrong / Invalid inputs
Enter username which contains spaces	Lana lana	Entry not allowed	Pass	Cancellation message needed with explanation what exactly is wrong
Invalid inputs				
(Unacceptable username not on the list)	&@@@!	Entry not allowed	Pass	Cancellation message needed
Enter username/password which contains spaces.	7777 77777	Entry not allowed	Pass	Cancellation message needed
Erroneous inputs				
Acceptable username not on the list)	Theo Maggie	Entry not allowed	Pass	Cancellation message needed. with offer to become registered
Acceptable username not on the list)	Qw787 Ere7879	Entry not allowed	Pass	Cancellation message needed. with offer to become registered
DATA MANAGEMENT TESTING:				
Verify that the file is updated correctly after each round to reflect the winning player's cards.		Data from the console = data from external file	Pass	Data from console was compared with an external file successfully
LEADERBOARD DISPLAY TESTING				
Confirm that the leaderboard displays the correct information for the top 5 players with the highest quantity of cards.		top 5 players with the highest quantity of cards	Fail	There is the same mistake with recording after the game. It seems that the last winner was not added to the list before sorting. As a result – top5 without the latest winner
Test the sorting mechanism to ensure that players are ranked accurately based on their card quantity.		top 5 players with the highest quantity of cards	Pass	

U3_4.3 Create technical documentation for the support and maintenance of the program.

All types of technical documentation fall into three main categories: product documentation, process documentation, and sales and marketing documents.

Product documents typically cover instructions and tutorials to help end-users accomplish a task. They include guides, illustrations, and reference sheets that cover: Information on the requirements or system specifications users need to run the product efficiently. Installation and usage instructions.

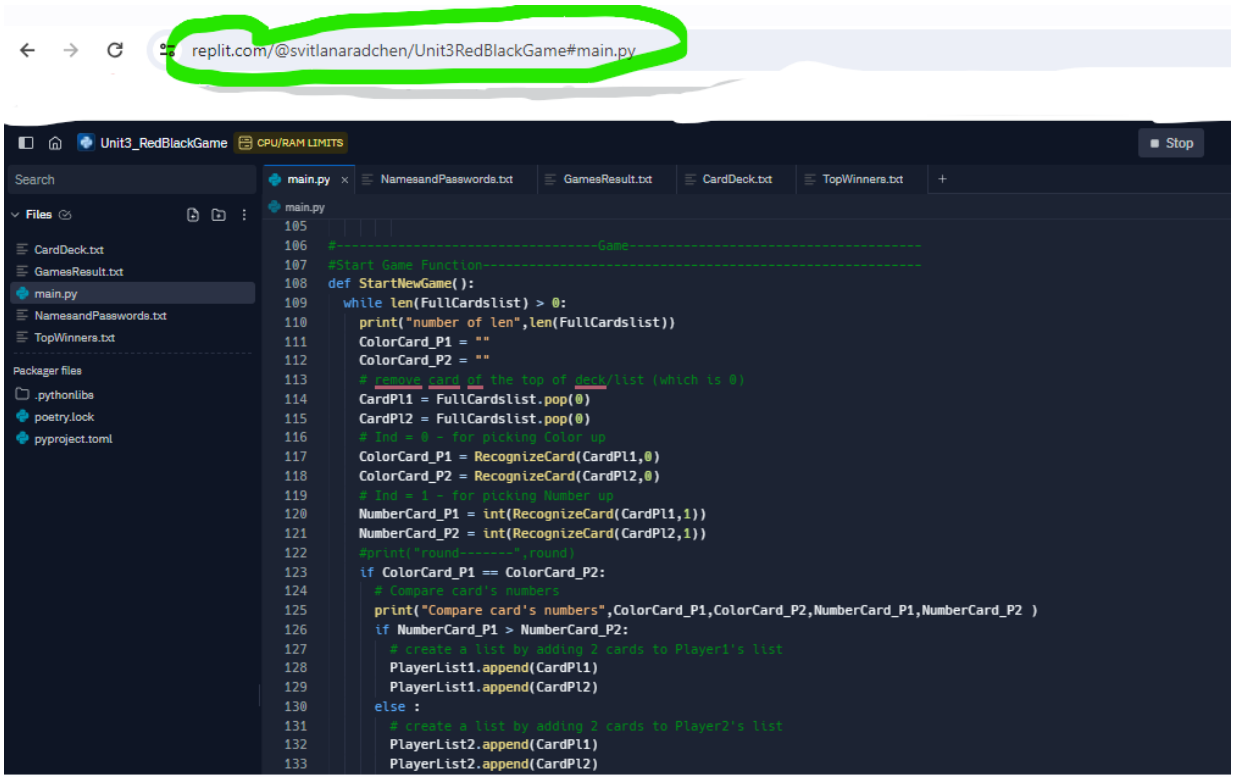
Maintenance guide documentation – tells the user how to maintain the system effectively and can include a definition of the software support environment, roles, and responsibilities of the maintenance personnel.

TECHNICAL DOCUMENTATION FOR LUISE'S CARD GAME

Installation and usage instructions

1. SETTING UP – INSTALLING THE PROGRAM

The program was created by using REPLIT, an online IDLE.



For end users, we recommend using IDLE 3.9 (Python) which is available here: https://www.python.org/downloads/release/python-390/

The user will need 3 texts (e.g. Notepad++) to be saved at the same location:

1. CardDeck.txt

2. NamesandPasswords.txt
3. GamesResult.txt
4. TopWinners.txt

Usage Instructions:

When you open Replit (web or Replit app):

- Step 1: Launch "Louise's Game" by clicking on the icon «Run»
- Step 2: You will be prompted to enter your player credentials to log in.
- Step 3: Once logged in, you can start a new game.
- Step 4: Track the game progress shown on the console
- Step 5: Answer questions 1=yes and 0=no to get information about the progress of the game

Enjoy playing "Louise's Game" with your opponent and have fun!

Additional Information:

- Ensure that you have a stable internet connection to play the game with another player.
- You can adjust game settings, such as sound and graphics, from the options menu.
- If you encounter any issues or need assistance, refer to the game's help section or contact customer support.

Feedback and Support:

- Provide feedback to the game developers regarding any issues, bugs, or suggestions for improvement to help enhance the gaming experience for all players.
- Contact customer support if you encounter technical difficulties or require assistance with the game.

Troubleshooting:

- If you experience any technical difficulties during installation or gameplay, try restarting your computer and relaunching the game.
- Check for updates to ensure you have the latest version of "Louise's Game."
- For further assistance, reach out to our support team via email or online chat.

2. HARDWARE REQUIREMENTS

The program was run and tested using Android phone (Replit app, on Android13) and PC with a Windows10 OS. See the detailed hardware information below.

System Requirements:

- Operating System: Windows 10/8/7
- Processor: Intel Core i3 or equivalent
- RAM: 4GB or higher - Graphics: Integrated or dedicated graphics card
- Storage: 1GB of free space

More specifically information you can see as an example below

Item	Value
OS Name	Microsoft Windows 10 Enterprise
Version	10.0.19045 Build 19045
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	A210648
System Manufacturer	HP
System Model	HP ProDesk 400 G6 Desktop Mini PC
System Type	x64-based PC
System SKU	123V3ET#ABU
Processor	Intel(R) Core(TM) i5-10500T CPU @ 2.30GHz, 2304 Mhz, 6 Core(s), 12 Logical ...
BIOS Version/Date	HP S23 Ver. 02.05.01, 05/01/2021
SMBIOS Version	3.2
Embedded Controller Version	9.148
BIOS Mode	UEFI
BaseBoard Manufacturer	HP
BaseBoard Product	871A
BaseBoard Version	KBC Version 09.94.00
Platform Role	Desktop

3. SOFTWARE REQUIREMENTS:

1. **Player Authentication**:
   Only authorized players should be able to access and play the game. Implement a secure authentication system to verify player identities.
2. **Deck Management**:
   The game's deck should have 30 shuffled cards, each with a unique colour (red, black, or yellow) and number (1-10 for each colour). Develop a mechanism to manage and shuffle the deck.
3. **Gameplay Rules**:
   Player 1 and Player 2 take turns drawing cards from the deck.
   Determine the winner of each round based on the color and number comparison rules specified.
   Keep track of the cards won by the winning player after each round.
4. **Color Comparison Rules:**
   Define the logic for determining the winning color when players have cards of different colors (red beats black, black beats yellow, yellow beats red).
5. **Round Completion**:
   The game continues until all cards in the deck are drawn. Ensure proper handling of the end-game scenario.
6. **Display Winning Player's Cards:**
   List all cards held by the winning player at the end of each round for transparency and record-keeping purposes.

7. **External File Storage**:
   Store the name and quantity of cards held by the winning player in an external file for data persistence and retrieval.
8. **Leaderboard Functionality**:
   Implement a feature to display the names and quantities of cards held by the top 5 players with the highest card counts based on the data stored in the external file.
9. **Error Handling**:
   Include error handling mechanisms to address any unexpected scenarios that may occur during gameplay, such as invalid card draws or file access issues.
10. **User Interface**:
    Design a user-friendly interface that clearly displays game information, player actions, and results to enhance the gaming experience.
11. **Scalability and Performance:**
    Ensure that the game can handle multiple concurrent players efficiently and that the gameplay remains smooth even as the number of players increases.

4. TESTING

Testing of the program was done using the conditions outlined in paragraph 4.1

5. DATA STRUCTURES / ALGORITHMS

Deck of Cards.

- An array or list to represent the deck of cards.
- Each card can be represented as a structure or object containing attributes like color, number, and status (e.g., in deck, played).
- The deck can be shuffled using appropriate algorithms

Player Information:

- A structure or class to store player information, including name, score, and cards held.
- Player data can be organized in a data structure like a hash table or dictionary for efficient retrieval and updating.

Leaderboard

- A data structure to store and maintain **player scores and rankings**.
- This can be implemented using a sorted array, linked list, or priority queue to quickly retrieve top players based on their scores.

ALGORITHMS

The program algorithms were implemented using Flowchart and are demonstrated in paragraph 3.1

Requirement specifications for LUISE'S CARD GAME

Requirement specifications, also known as software requirements specifications (SRS), are a comprehensive document that **outlines the functional and non-functional requirements** of a software application. It serves as a blueprint

for software development by detailing what the software should do, how it should behave, and what constraints or limitations it must adhere to. Requirement specifications are crucial for ensuring that the software meets the needs and expectations of stakeholders, such as clients, users, and developers.

REQUIREMENT SPECIFICATIONS FOR LUISE'S CARD GAME	
INTRODUCTION PURPOSE AND SCOPE	The purpose of this document is to outline the software requirements for developing Luise's Card Game, a two-player card game where players compete to win cards based on color and number combinations. The software will facilitate the gameplay of Luise's Card Game, including managing the deck of 30 unique cards, enforcing game rules, determining winners of each round, and storing player data for future reference.
FUNCTIONAL REQUIREMENTS	**Number of Players** - The game will support only 2 players. - Only authorized players are allowed to play the game. **Deck Management** - The deck will consist of 30 shuffled cards. - Each card in the deck is unique. - Each card has one of three colors: red, black, or yellow. - Each card has a number from 1 to 10 for each color. **Gameplay Rules** - Player 1 takes the top card from the deck. - Player 2 takes the next card from the deck. - If both players have a card of the same color, the player with the highest number wins. - If players have cards of different colors, the winning color hierarchy is as follows: - RED beats BLACK - BLACK beats YELLOW - YELLOW beats RED - The winner of each round keeps both cards. - Players continue playing until there are no cards left in the deck. **Winner Data Management** - Lists all cards held by the winning player after each round. - Stores the name and quantity of cards of the winning player in an external file. **Leaderboard Display** - Displays the name and quantity of cards of the top 5 players with the highest quantity of cards from the external file.

REQUIREMENT SPECIFICATIONS FOR LUISE'S CARD GAME	
NON-FUNCTIONAL REQUIREMENTS	**SECURITY** - Only authorized players can access and play the game. - Player data stored in external files will be encrypted for security purposes. **MAINTAIN-ABILITY** Using coding standards **PERFORMANCE** - The game should be optimized to run smoothly without lag or delays during gameplay - The game should execute efficiently and without noticeable delays, providing a smooth and responsive user experience during gameplay. - Performance Requirement: The game must load within 5 seconds on all supported devices to provide a smooth and responsive user experience. **FLEXI-BILITY** - The game should work as given, without allowing users to customize features, workflows, or settings to suit their preferences.
USER STORIES USE CASES *As a player, I want to*	User Story 1: Player Registration - As a registered player, I want to be able to play the game by providing my username and password. User Story 2: Gameplay - to play cards according to the game rules and see the outcome of each round User Story 3: Winner Declaration - to know who the winner of each round is and see my own progress in the game. User Story 4: Leaderboard Display - to see the leaderboard displaying the top players with the highest quantity of cards. User Story 5: Authentication - to log in securely with my credentials and ensure that unauthorized users cannot access the game. User Story 6: Usability - the game interface to be user-friendly, responsive, and easy to navigate.

REQUIREMENT SPECIFICATIONS FOR LUISE'S CARD GAME	
BUSINESS RULES	1. Player Registration Rule: - Players must register with a unique username and password to access the game. 2. Game Setup Rule: - The game must allow for a minimum of 2 and a maximum of 4 players to participate in each game session. 3. Gameplay Rule: - Players must follow the game rules when playing cards, including following suit and playing valid cards. 4. Winner Declaration Rule: - The winner of each round must be determined based on specific criteria, such as the player with the highest card value or the first player to play all their cards. 5. Data Management Rule: - Game data, including player progress, scores, and game settings, must be securely saved and retrieved for each player. 6. Leaderboard Display Rule: - The leaderboard must display the top players based on the number of cards they have collected throughout the game. 7. Authentication Rule: - Players must log in securely using their credentials, and unauthorized access to the game must be prevented. 8. Usability Rule: - The game interface must be intuitive, responsive, and accessible to players of all skill levels.
DATA REQUIREMENTS	1. **Player Authorization**: - The system must store player credentials, including usernames and passwords, for authorization purposes. - Player status (authorized/unauthorized) must be maintained in the system. 2. **Deck Initialization:** - The system must store a deck of 30 unique cards with attributes such as color and number. - Each card in the deck must have a unique combination of color and number. - The system must have a mechanism to shuffle and store the deck in random order. 3. **Game Play Mechanics:** - The system must track the cards played by each player during the game. - Winning player information, including the player's name and the cards won, must be stored. - Game state data, such as the number of cards remaining in the deck, must be stored. 4. **Data Storage** and Display - The system must maintain a record of all cards held by the winning player after each round. - Player data, including names and quantities of cards won, must be stored in an external file for persistence.

REQUIREMENT SPECIFICATIONS FOR LUISE'S CARD GAME	
	- The system must retrieve and display the top 5 players with the highest quantity of cards from the external file for leaderboard display. 5.**Data Portability**: - The game data should be stored in a format that allows for easy transfer and access across different environments.
INTERFACE REQUIREMENTS	The user interface should be intuitive and easy to navigate for players.
COMPATIBILITY REQUIREMENTS	The game should be compatible with common operating systems and devices
PERFORMANCE REQUIREMENTS:	The game should respond quickly to player actions and provide a seamless gaming experience.
SECURITY REQUIREMENTS	- Only authorized players can access and play the game. - Player data stored in external files will be encrypted for security purposes.
TESTING REQUIREMENTS	**Functional** Testing: - Verify that the game rules are correctly implemented. - Test the card comparison logic to ensure that the correct player wins based on color and number. - Confirm that the deck shuffling and card distribution are working as expected. - Test the winner data management system to ensure it accurately records and stores the winning player's information. **Authorization** Testing: - Verify that only authorized players can access and play the game. - Test the authentication process to ensure that unauthorized users are blocked from playing. **Data Management** Testing: - Test the storage and retrieval of player information in the external file. - Verify that the file is updated correctly after each round to reflect the winning player's cards. **Leaderboard** Display Testing: - Confirm that the leaderboard displays the correct information for the top 5 players with the highest quantity of cards. - Test the sorting mechanism to ensure that players are ranked accurately based on their card quantity. **Integration** Testing: - Verify that all components of the game, including gameplay, data management, and leaderboard display, work together seamlessly. - Test interactions between different modules to ensure they communicate and function properly. **Usability** Testing:

REQUIREMENT SPECIFICATIONS FOR LUISE'S CARD GAME	
	- Gather feedback from users to assess the game's ease of use and overall user experience. - Test the game interface for clarity, responsiveness, and intuitiveness. **Performance** Testing: - Evaluate the game's performance under various conditions, such as different numbers of players or large quantities of cards. - Test for any potential bottlenecks or issues that may affect gameplay.

By documenting and analysing these requirements in detail, requirement specifications provide a **clear roadmap for software development** teams to design, implement, test, and deliver a high-quality software application that meets stakeholder expectations.