

Electrical Engineering Department

**REPORT**

ASSIGNMENT 2

Name: Tashi Tobgay Ghalley

STD no: 02240100

Module: CSF101

Tutor:Mr.Darshan Subedi &Mr.Vincent Ho

Github link:

|  |  |  |
| --- | --- | --- |
| SL.NO | CONTENT | Page |
| 1. | Introduction | 2 |
| 2. | Part A |  |
| 3. | Part B |  |
| 4 | Outputs |  |
| 5. | Conclusion |  |
| 6. | reference |  |

**Introduction**

This Python programming practice is meant to demonstrate expertise in the development of interactive command-line tools from object-oriented programming (OOP) principles. The practice consists of two parts. Part A is running five entertaining mini-games and utilities with mathematical and string-processing algorithms. They include a number guessing game, rock-paper-scissors, a trivia quiz game, a Pokémon card binder manager, and a general scoring system. Part B expands the Pokémon card binder manager as a standalone module that helps the collector sort out their Pokémon cards by Pokedex numbers through the use of in-memory storage logic. The project emphasizes user interaction, input validation, class-based design, and session management in order to replicate an actual program that is educational and entertaining.

**PART A :**

import random

import TashiTobgayGhalley\_02240100\_A2\_PB

class Game:

    """Main game class to display menu and run selected games."""

    def \_\_init\_\_(self):

        self.total\_score = 0  # Overall score tracker

    def display\_menu(self):

        """Displays the main menu options."""

        print("\nMain Menu")

        print("1. Guess Number Game")

        print("2. Rock Paper Scissors")

        print("3. Trivia Pursuit Quiz")

        print("4. Pokemon Card Binder Manager")

        print("5. Show Overall Score")

        print("6. Exit")

    def start(self):

        """Starts the main game loop and handles user selection."""

        while True:

            self.display\_menu()

            choice = input("Select a function (1-6): ")

            if choice == '1':

                self.total\_score += GuessNumberGame().play()

            elif choice == '2':

                self.total\_score += RockPaperScissors().play()

            elif choice == '3':

                self.total\_score += TriviaGame().play()

            elif choice == '4':

                self.total\_score += TashiTobgayGhalley\_02240100\_A2\_PB.PokemonCardBinderManager().menu()

            elif choice == '5':

                print(f"Your Overall Score Across All Games: {self.total\_score}")

            elif choice == '6':

                print("Thanks for playing!")

                break

            else:

                print("Invalid choice. Please enter a number between 1 and 6.")

class GuessNumberGame:

    """Class for the number guessing game."""

    def play(self):

        """Plays the Guess Number Game with input validation and scoring."""

        number = random.randint(1, 100)

        score = 0

        guesses = 0

        print("Guess the Number (1 to 100)")

        while True:

            try:

                guess = int(input("Your guess: "))

                if guess < 1 or guess > 100:

                    print("Please enter a number between 1 and 100.")

                    continue

                guesses += 1

                if guess == number:

                    print("Correct!")

                    break

                elif guess < number:

                    print("Too low.")

                else:

                    print("Too high.")

            except ValueError:

                print("Invalid input. Please enter a valid number.")

        score = max(0, 10 - (guesses - 1))

        print(f"Score: {score}")

        return score

class RockPaperScissors:

    """Class for the Rock Paper Scissors game."""

    def play(self):

        """Plays Rock Paper Scissors game in a loop until user exits."""

        options = ['rock', 'paper', 'scissors']

        score = 0

        print("Rock Paper Scissors")

        print("Type 'exit' to return to main menu.")

        while True:

            user = input("Your choice (rock/paper/scissors): ").lower()

            if user == 'exit':

                break

            if user not in options:

                print("Invalid input. Please type rock, paper, or scissors.")

                continue

            computer\_choice = random.choice(options)

            print(f"Computer chose: {computer\_choice}")

            if user == computer\_choice:

                print("It's a draw!")

            elif (user == 'rock' and computer\_choice == 'scissors') or \

                 (user == 'paper' and computer\_choice == 'rock') or \

                 (user == 'scissors' and computer\_choice == 'paper'):

                print("You win!")

                score += 1

            else:

                print("You lose.")

        print(f"Score: {score}")

        return score

class TriviaGame:

    """Class for Trivia Pursuit Game."""

    def \_\_init\_\_(self):

        self.questions = {

            'Bhutanese Tradition': [

                ("What is the national animal of Bhutan?", ['Cow', 'Yak', 'Takin', 'Sheep'], 'Takin'),

                ("What is the national bird of Bhutan?", ['Raven', 'Peacock', 'Hen', 'Crow'], 'Raven')

            ],

            'Mathematics': [

                ("x = 5y + 1. Find y if x = 11?", ['1', '3', '11', '2'], '2'),

                ("What number comes next: 1, 2, 4, 7, 11, x, 22?", ['6', '12', '16', '18'], '16')

            ]

        }

    def play(self):

        """Runs the trivia quiz game with category selection and question answering."""

        score = 0

        print("Trivia Pursuit Quiz")

        categories = list(self.questions.keys())

        for i, cat in enumerate(categories):

            print(f"{i + 1}. {cat}")

        try:

            selected = int(input("Choose a category (1-{}): ".format(len(categories))))

            if selected < 1 or selected > len(categories):

                print("Invalid category number.")

                return 0

        except ValueError:

            print("Invalid input. Please enter a number.")

            return 0

        category = categories[selected - 1]

        for a, choices, correct in self.questions[category]:

            print(f"\n{a}")

            for i, choice in enumerate(choices):

                print(f"{i + 1}. {choice}")

            try:

                Answer = int(input("Choose the answer (1-4): "))

                if Answer < 1 or Answer > 4:

                    print("Invalid option. Skipping question.")

                    continue

                if choices[Answer - 1] == correct:

                    print("Correct!")

                    score += 1

                else:

                    print("Incorrect.")

            except ValueError:

                print("Invalid input. Please enter a number.")

        print(f"Score: {score}")

        return score

if \_\_name\_\_ == "\_\_main\_\_":

    Game().start()

**PART B:**

class PokemonCardBinderManager:

    """Manages a Pokémon card binder with functions to add, reset, view, and exit."""

    def \_\_init\_\_(self):

        """Initializes an empty binder dictionary."""

        self.binder = {}  # {pokedex\_number: (page, row, column)}

    def add\_card(self):

        """Adds a Pokémon card by Pokedex number with validation and position logic."""

        try:

            pokedex = int(input("Enter Pokedex number (1–1025): "))

            if not (1 <= pokedex <= 1025):

                print(" Invalid Pokedex number. Must be between 1 and 1025.")

                return

        except ValueError:

            print(" Invalid input. Please enter a valid integer.")

            return

        if pokedex in self.binder:

            page, row, col = self.binder[pokedex]

            print(f" Card already exists.\n Page: {page}, Position: ({row},{col})\nStatus: Duplicate")

        else:

            index = len(self.binder)

            page = index // 64 + 1

            position = index % 64

            row = position // 8 + 1

            col = position% 8 + 1

            self.binder[pokedex] = (page, row, col)

            print(f"Card added.\nPage: {page}, Position: ({row},{col})\nStatus: Added")

    def reset\_binder(self):

        """Resets the binder after user confirms action."""

        print("WARNING: This will erase all cards in the binder.")

        choice = input("Type 'CONFIRM' to proceed or 'EXIT' to cancel: ").strip().upper()

        if choice == "CONFIRM":

            self.binder.clear()

            print(" Binder has been reset.")

        elif choice == "EXIT":

            print("Reset cancelled.")

        else:

            print(" Invalid choice. Reset not performed.")

    def display\_binder(self):

        """Displays all stored cards and completion percentage."""

        print("Current Binder Summary")

        if not self.binder:

            print(" Binder is empty.")

        else:

            sorted\_cards = sorted(self.binder.items())

            for pokedex, (page, row, col) in sorted\_cards:

                print(f"Pokedex #{pokedex} ➡ Page: {page}, Position: ({row},{col})")

            total = len(self.binder)

            completion = (total / 1025) \* 100

            print(f"Total cards: {total}")

            print(f"Completion: {completion:.2f}%")

    def exit\_session(self):

        """Displays a summary and exits the session."""

        print(f"Exiting binder. Total cards stored: {len(self.binder)}")

        return len(self.binder)

    def menu(self):

        """Runs the binder manager menu loop until user chooses to exit."""

        while True:

            print("Pokemon Card Binder Manager")

            print("1. Add a Pokemon card by Pokedex number")

            print("2. Reset the Binder")

            print("3. Display current cards and stats")

            print("4. Exit to Main Menu")

            mode = input("Select a mode (1-4): ").strip()

            if mode == '1':

                self.add\_card()

            elif mode == '2':

                self.reset\_binder()

            elif mode == '3':

                self.display\_binder()

            elif mode == '4':

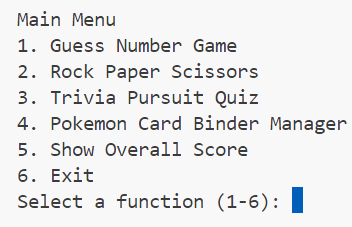
                score = self.exit\_session()

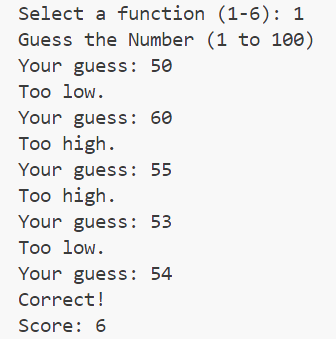
                return score

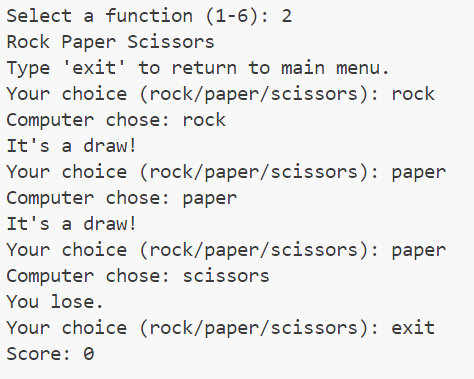
            else:

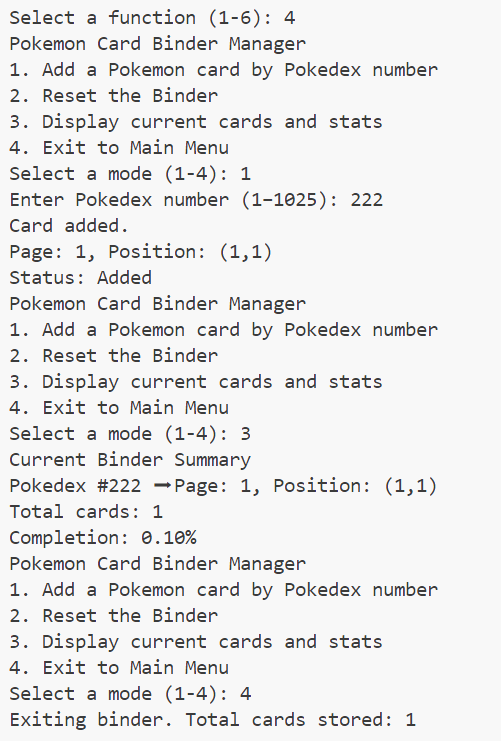
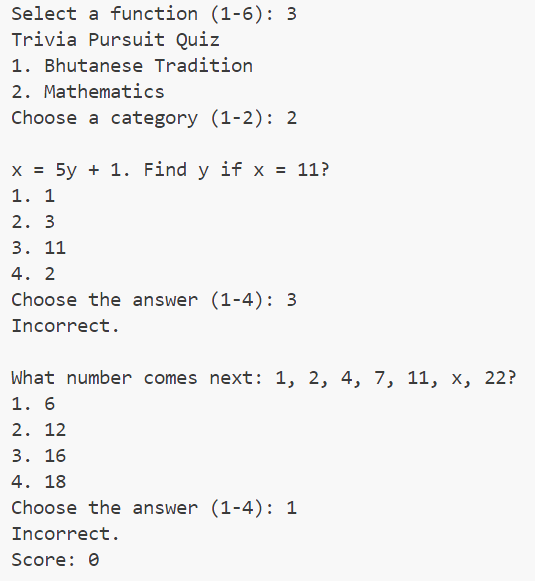
                print("Invalid option. Please enter a number between 1 and 4.")

**OUTPUTS:**

****

****

****

****

**Conclusion**

In conclusion project successfully integrates object-oriented programming concepts and real-world program design to create an interactive menu-driven Python program. Through the development of five standalone modules in Part A and a personalized Pokémon card binder manager in Part B, the project illustrates the effective use of classes, processing user inputs, in-memory data structures, and modular programming concepts. Every part has its role in an integrated system that is both useful and user-friendly. Overall, the project not only strengthens core programming skills but also demonstrates the importance of structuring code for readability, reusability, and real-world applicability.

# References

CheatSheet.com. (n.d.). Python Cheatsheet for Beginners. Retrieved May 9, 2025, from <https://www.cheatsheet.com/python>

W3Schools. (n.d.). Python cheat sheet. W3Schools. from <https://www.w3schools.com/python/python_reference.asp>

Telusko. (2018, October 6). Python Tutorial for Beginners 5 - Python Strings [Video]. YouTube. <https://www.youtube.com/watch?v=8O5kX73OkIY>