A red and blue logo

Description automatically generated

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Tutor-Marked Assignment (TMA01) Jul 2023**

|  |  |
| --- | --- |
| **Name** | **TAN ZEKAI MARCUS** |
| **SUSS PI No** | **H2011558** |
| **Tutorial Group** | **T03** |
| **Course Code** | **ANL252\_Jul2023\_T03: PYTHON\_FOR\_DATA\_ANALYTICS** |
| **Title of TMA** | **ANL252\_TMA01\_Marcustan015\_TanZekaiMarcus** |
| **Submission Date** | 12/09/2023 |

|  |
| --- |
| **1a)** These are the few reasons why Plagiarism in coding happens:   * **Lack of Understanding** – Programming is often viewed as too technical and complicated by people who can’t comprehend the logic easily, which results in demotivation. Solution is to start small and teach basic functions first and develop from there. * **Difficulty and Inexperience** – Programming questions are more open ended and have numerous solutions. However, the downside is that novice students might not know how to tackle the question due to the ambiguity. The solution is to plan sufficient time and practice as much as possible to understand the curriculum. * **Code Availability -** Due to the open-source nature of coding and the surge in generative AI tools. Coding solutions are easily found on the web. (Turnitin 2020). Solution is for professors to conduct code review and make sure the student understands what he/she is coding despite getting the solution online. |

|  |
| --- |
| **1b)** import random  def RPS\_GAME():  player\_choice = input("Choose rock, paper, or scissors: ").lower()  computer\_choice = random.choice(["rock", "paper", "scissors"])  print(f"Computer chose {computer\_choice}")  if player\_choice == computer\_choice:  print("It's a tie!")  elif player\_choice == "rock":  if computer\_choice == "scissors":  print("You win!")  else:  print("Computer wins!")  elif player\_choice == "paper":  if computer\_choice == "rock":  print("You win!")  else:  print("Computer wins!")  elif player\_choice == "scissors":  if computer\_choice == "paper":  print("You win!")  else:  print("Computer wins!")  elif player\_choice not in ['rock','paper','scissors']:  print("Invalid choice. Please choose rock, paper, or scissors.")    RPS\_GAME()  Function Overview: This python code facilitates the classic Rock/Paper/Scissors game between the user and the computer. How it works is that the user will input either ‘Rock’, ‘Paper’ or ‘Scissors’ and the computer will pick from one of the 3 choices above as well. If the input is invalid, an invalid message will be shown. The winner is picked based on what both parties select. For example, if the computer chooses rock and user chooses paper, the user wins. If the same choice is chosen by both parties, it is a tie |

|  |
| --- |
| **1c)** import random  def RPS\_GAME\_v2():  player= input("Choose rock, paper, or scissors: ").lower()  computer = random.choice(["rock", "paper", "scissors"])  print(f"Computer chose {computer}")  outcomes = {  "rock": {"rock": "It's a tie!", "paper": "Computer wins!", "scissors": "You win!"},  "paper": {"rock": "You win!", "paper": "It's a tie!", "scissors": "Computer wins!"},  "scissors": {"rock": "Computer wins!", "paper": "You win!", "scissors": "It's a tie!"}  }  if player in outcomes:  print(outcomes[player][computer])  else:  print("Invalid choice. Please choose rock, paper, or scissors ONLY.")  RPS\_GAME\_v2()    The code above functions the same way as the one shown in 1b. However, there were changes made to the logic behind how the input matches with the computer’s output.   * **Nested Dictionary** – A dictionary of the three potential choices is created call ‘outcome’, inside ‘outcome’ contains another dictionary which shows the potential results of the game based on what the computer has chosen. This provides a more convenient way of looking at each result instead of a bunch of ‘ifelse’ statements. * **Print Statement –** The system will then use an if statement to check if the input the player has entered matches with the ‘Key’ of any of the ‘outcome’ dictionary. If so, it then takes the outcome dictionary and selects the player input as the ‘Key’ and the computer choice as the ‘Value’ to extract out the result to print. This again, makes the code easier to read and saves memory while shortening the code instead of having multiple print statements for every ‘ifelse’ outcomes as shown in 1b. * **Else Condition –** In the code for 1b) the Invalid choice statement is only printed in the ‘elif’ condition if the input by the player IS NOT rocks, papers, or scissors. This was changed 1c) so that if the player’s input is not any of the ‘Keys’ in the outcome dictionary, it will just print out the invalid choice statement. The reason for the change is again, to shorten the logic of the code and to save processing power as the system doesn’t need to handle another ’elif’ condition and just process the ‘else’. |

|  |
| --- |
| **2)** def get\_valid\_float\_input(prompt):  while True:  try:  value = float(input(prompt))  return value  except ValueError:  print("Invalid input! Please enter a valid number.")  products = {  'laptop': None,  'mouse': None,  'webcam': None,  'keyboard': None,  'speaker': None  }  print(f'We have a list of products here: {list(products.keys())}.')  while True:  item = input("Hello! What do you want to buy? ").lower()  if item in products:  price\_of\_item = get\_valid\_float\_input(f"How much is the {item} (in SGD)? ")  products[item] = price\_of\_item  else:  print("Wrong product! Please try again.")  while True:  Continue = input("Would you like to continue? (yes/no) ").lower()  if Continue in ['yes', 'no']:  break  else:  print("Invalid input! Please enter 'yes' or 'no'.")  if Continue == 'no':  print ("Thank you for shopping with us!")  break  print(f"This is our shopping list: {products}")  The code above functions mostly the same as Appendix 1. However, there were alterations done to improve the overall quality of it:   * **Reliability –** Data validation is included for every input. If the user enters a wrong input the code wouldn’t just ‘break’ but prompt the user to enter the correct value, this prevents abrupt termination. Input cleaning is also introduced by converting every input to lowercase to avoid case sensitivity and discrepancy. An exit message is also included once the user exits the loop to notify them that the program has concluded. * **Readability –** The code is made readable by changing the ‘products’ nested list into a dictionary, this allows the user to view the items and their prices better. The ‘Query’ variable is also changed to ‘Continue’ so that readers of the code can get a better description of the variable’s purpose. Structured prompts to let the user know the input is wrong are present to guide them on what to enter which enhances user experience. * **Maintainability –** With numerous while loops and validation checks in the code, one of them was changed into another function instead, this makes the main function less messy and allows for easier future code changes. The dictionary also makes maintenance of products and prices less confusing than a nested list |