**A close-up of a sign

Description automatically generated**

**Programming Tasks (Mark Scheme)**

*It should be noted that solutions to each problem may vary depending on the techniques selected by the candidate. Marks should be awarded for solutions which achieve the correct outcome but through different techniques than those shown here.*

# Task 1 (3 marks)

## Coding

* Prompt the user if they would like to quit. **[1 mark]**
* Exit the application loop correctly and immediately. **[1 mark]**

**Teacher Notes:**

*Candidates should ensure that once the user has chosen to quit the game, the application should not decrement the score or call the UpdateTargets method.*

### Example Solution

Modification to the PlayGame method:

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        UserInput = input("Enter an expression or enter QUIT to end the game: ").upper()

        print()

        if (UserInput == "QUIT"):

            GameOver = True

        else:

            if CheckIfUserInputValid(UserInput):

                UserInputInRPN = ConvertToRPN(UserInput)

                if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                    IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                    if IsTarget:

                        NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                        NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

            Score -= 1

            if Targets[0] != -1:

                GameOver = True

            else:

                Targets = UpdateTargets(Targets, TrainingGame, MaxTarget)

        #END CHANGE

    print("Game over!")

    DisplayScore(Score)

## Testing

* Show the program displaying the identified target and final game score. **[1 mark]**

A screenshot of a computer program

Description automatically generated

# Task 2 (6 marks)

## Coding

* Prompt the user if an expression doesn’t evaluate to a target. [1 mark]
* Prompt the user if an expression includes values which are not valid. [1 mark]
* Prompt the user if an expression is not valid. [1 mark]
* Prompt the user if an expression includes values which are greater than the MaxNumber. [1 mark]
* Test to check if the expression includes a division by zero error. [1 mark]

**Teacher Notes:**

*Error messages could be raised in a number of different places. The example solution extends the selection in the PlayGame method to respond to a false return from the three “checking” methods together with a response in the CheckNumbersUsedAreAllInNumbersAllowed method to a value out of range.*

*Alternative solutions within the “checking” methods which are valid should still gain credit as long as they give an appropriate and correct response for the error raised.*

*Reject solutions to the div zero error which do not use a Regular Expression.*

### Example Solution

Changes to PlayGame:

        if CheckIfUserInputValid(UserInput):

            UserInputInRPN = ConvertToRPN(UserInput)

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

                #CHANGE

                else:

                    print("Your expression doesn't evaluate to one of the targets.")

            else:

                print("Your expression includes values which are not valid.")

        else:

            print("That is not a valid expression, please try again")

        print()

        #END CHANGE

        Score -= 1

Changes to CheckNumbersUsedAreAllInNumbersAllowed:

                Temp.remove(int(Item))

            else:

                return False

A screenshot of a computer

Description automatically generated        #CHANGE

        elif isdigit(Item):

            return False

        #END CHANGE

    return True

Changes to CheckIfUserInputValid:

def CheckIfUserInputValid(UserInput):

    #CHANGE

    if re.search("\/0+", UserInput) is not None:

        return False

    elif re.search("^([0-9]+[\\+\\-\\\*\\/])+[0-9]+$", UserInput) is not None:

        return True

    else:

        return False

    #END CHANGE

## Testing

* Show the program displaying the required error messages. [1 mark] ▶

# Task 3 (2 marks)

## Coding

* Refactor the code to remove the requirement for the Targets list to be iterated through. [1 mark]

**Teacher Notes:**

*The objective of this question is to assess whether the candidate understands the functionality of a queue rather than iterating through a list.*

### Example Solution

Changes to UpdateTargets:

def UpdateTargets(Targets, TrainingGame, MaxTarget):

    #CHANGE

    Targets.pop(0)

    #END CHANGE

    if TrainingGame:

        Targets.append(Targets[-1])

    else:

        Targets.append(GetTarget(MaxTarget))

    return Targets

## A black screen with white text Description automatically generatedTesting

* Show the program correctly displaying the Targets list moved along after a turn. [1 mark] ▶

# Task 4 (8 marks)

## Coding

* Displaying a suitable menu for the modified standard random game. [1 mark]
* Storing the user game type choice in an appropriately named variable. [1 mark]
* Storing the “large” numbers in a suitable data structure. [1 mark]
* Using selection in the FillNumbers method to fill the NumbersAllowed list differently, depending on the user game type choice. [1 mark]
* Correctly populating the NumbersAllowed list with one large number and four standard numbers. [1 mark]
* Correctly populating the NumbersAllowed list for the other different game type choices. [1 mark]
* Maintaining the functionality to repopulate the NumbersAllowed list in a standard and training game. [1 mark]

**Teacher Notes:**

*The difficulty level between turns should be maintained; however, this should be done by repopulating the NumbersAllowed list completely, regardless of whether values within it have been used to identify a target.*

### Example Solution

Changes to Main:

    Choice = input("Enter y to play the training game, anything else to play a random game: ").lower()

    print()

    #CHANGE

    GameChoiceType = 0

    if Choice == "y":

        MaxNumber = 1000

        MaxTarget = 1000

        TrainingGame = True

        Targets = [-1, -1, -1, -1, -1, 23, 9, 140, 82, 121, 34, 45, 68, 75, 34, 23, 119, 43, 23, 119]

    else:

        print("Please select from one of the below game options:")

        print("1: Standard Game")

        print("2: Easy Game - One large number and 4 small numbers")

        print("3: Medium Game - Two large numbers and 3 small numbers")

        print("4: Hard game - Four large numbers and 1 small number")

        GameChoiceType = int(input())

        MaxNumber = 10

        MaxTarget = 50

        Targets = CreateTargets(MaxNumberOfTargets, MaxTarget)

    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber, GameChoiceType)

    PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, GameChoiceType)

    input()

Modification to the PlayGame method to expose the game type choice made by the user:

#CHANGE

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, GameChoiceType):

    Score = 0

    GameOver = False

    while not GameOver:

…

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber, GameChoiceType)   ##CHANGED HERE

        Score -= 1

        if Targets[0] != -1:

Modification to the CheckValidNumber method to recognise a “large” number:

def CheckValidNumber(Item, MaxNumber):

    if re.search("^[0-9]+$", Item) is not None:

        #CHANGE

        Temp = [25, 50, 75, 100]

        ItemAsInteger = int(Item)

        if ItemAsInteger > 0 and ItemAsInteger <= MaxNumber or ItemAsInteger in Temp:

            return True

        #END CHANGE

    return False

Modification to the FillNumbers method to correctly fill the NumbersAllowed list depending on the game type choice made by the user:

#CHANGE

def FillNumbers(NumbersAllowed, TrainingGame, MaxNumber, GameChoiceType):

    LargeNumbers = [25, 50, 75, 100]

    if TrainingGame:

        return [2, 3, 2, 8, 512]

    elif GameChoiceType == 2:

        NumbersAllowed = []

        NumbersAllowed.append(LargeNumbers[random.randint(0,3)])

        for i in range(4):

            NumbersAllowed.append(GetNumber(MaxNumber))

    elif GameChoiceType == 3:

        NumbersAllowed = []

        NumbersAllowed.append(LargeNumbers[random.randint(0,3)])

        NumbersAllowed.append(LargeNumbers[random.randint(0,3)])

        for i in range(3):

            NumbersAllowed.append(GetNumber(MaxNumber))

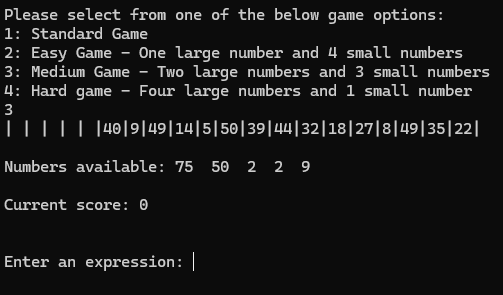
    elif GameChoiceType == 4:

        NumbersAllowed = []

        for i in range(4):

            NumbersAllowed.append(LargeNumbers[random.randint(0,3)])

        NumbersAllowed.append(GetNumber(MaxNumber))

    else:

        while len(NumbersAllowed) < 5:

            NumbersAllowed.append(GetNumber(MaxNumber))

    return NumbersAllowed

    #END CHANGE

## Testing

* Displaying two “large” numbers in the NumbersAllowed list. [1 mark] ▶

# Task 5 (4 marks)

## Coding

* Suitable variable to count the number of operands. [1 mark]
* Iterate through the user expression. [1 mark]
* Correctly increment operand count. [1 mark]

**Teacher Notes:**

*The bonus score should be added prior to the score being decremented between turns to maintain the normal turn-to-turn operation of the program.*

### Example Solution

Modification to the CheckIfUserInputEvaluationIsATarget method:

#CHANGE

def CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score):

    BonusNumberCount = 0

    for s in UserInputInRPN:

        if s.isdigit(): #str.isdigit() doesn't pick up negative numbers, but works for this program

            BonusNumberCount+=1

    UserInputEvaluation = EvaluateRPN(UserInputInRPN)

    UserInputEvaluationIsATarget = False

    if UserInputEvaluation != -1:

        for Count in range(0, len(Targets)):

            if Targets[Count] == UserInputEvaluation:

A screenshot of a computer

Description automatically generated                Score += 2

                Targets[Count] = -1

                UserInputEvaluationIsATarget = True

        Score += BonusNumberCount \* 2

    return UserInputEvaluationIsATarget, Score

#END CHANGE

## Testing

* Show program correctly identifying the target 68 and increasing   
  the user score to 9 points. [1 mark] ▶

# Task 6 (4 marks)

## Coding

* Prompting the user to enter an expression or move the targets list. [1 mark]
* Iterate through the Targets list moving each item to the right (also accept updating just the head and tail of the list). [1 mark]
* Correctly updating the score. [1 mark]

**Teacher Notes:**

*The candidate should use selection to ensure the new method MoveTargetsBack is called instead of a normal turn. This is to ensure that MoveTargetsBack and UpdateTargets are not both called together.*

*The solution shown uses iteration to move the elements of the Targets list one index to the right. This technique is shown because the same technique is used to move the list to the left in the UpdateTargets method. Since the Targets list operates as a queue, the candidate may simply interact with the head and tail of the queue.*

### Example Solution

Modification to the PlayGame method:

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        UserInput = input("Enter an expression or enter MOVE to move the Targets list to the right one step (costs 2 points): ").upper()

        print()

        if UserInput == "MOVE":

            Score = MoveTargetsBack(Targets, Score)

        else:

            if CheckIfUserInputValid(UserInput):

                UserInputInRPN = ConvertToRPN(UserInput)

                if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                    IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                    if IsTarget:

                        NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                        NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

            Score -= 1

            if Targets[0] != -1:

                GameOver = True

            else:

                Targets = UpdateTargets(Targets, TrainingGame, MaxTarget)

        #END CHANGE

Creation of new MoveTargetsBack method:

#CHANGE

def MoveTargetsBack(Targets, Score):

    for Count in range(len(Targets)-1,0,-1):

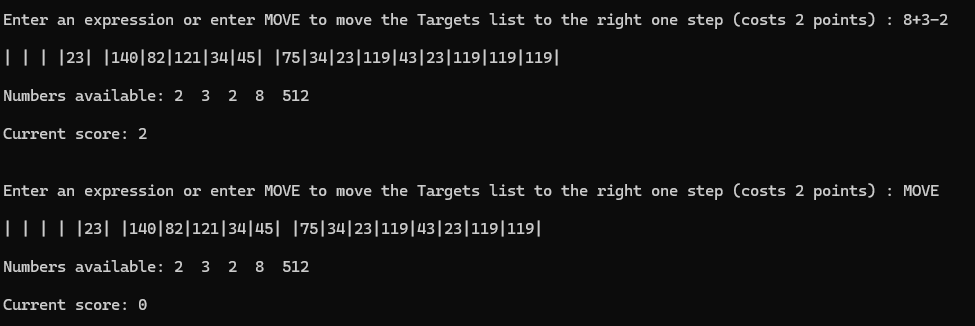
        Targets[Count] = Targets[Count -1]

    Targets[0] = -1

    return Score - 2

#CHANGE

## Testing

* Showing the correctly positioned Targets list following the move back. [1 mark]

# Task 7 (7 marks)

## Coding

* Suitable variable to store a target to be added to the NumbersAllowed list. [1 mark]
* Prompt to advise the user of a successful target identified and whether they would like to use it in the NumbersAllowed. [1 mark]
* Display the component parts of the identified target for the user to select. [1 mark]
* Use of selection to use the correct component part of the identified target. [1 mark]
* Add the selected value into the NumbersAllowed list (increasing the list size to six elements). [1 mark]
* Selection to maintain the functionality of the FillNumbers method if the user chooses to not use a target number. [1 mark]

**Teacher Notes:**

*Candidates should ensure that the FillNumbers method maintains its normal functionality of initially populating the NumbersAllowed list or populating it correctly in the scenario that the user doesn’t want to use any numbers from the identified target. The suggested solution uses an optional variable to identify this, which is only assigned a valid number if the user chooses one to use. An alternative technique could use a Boolean variable as a flag to indicate that the FillNumbers method needs to add the additional value.*

### Example Solution

Changes to the PlayGame method:

        if CheckIfUserInputValid(UserInput):

            UserInputInRPN = ConvertToRPN(UserInput)

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                #CHANGE

                TargetToAddToNumbersAllowed = -1

                TargetToAddToNumbersAllowed, IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score, TargetToAddToNumbersAllowed)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber, TargetToAddToNumbersAllowed)

                #END CHANGE

Changes to the CheckIfUserInputEvaluationIsATarget method:

#CHANGE

def CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score, TargetoAddToNumbersAllowed):

    UserInputEvaluation = EvaluateRPN(UserInputInRPN)

    UserInputEvaluationIsATarget = False

    if UserInputEvaluation != -1:

        for Count in range(0, len(Targets)):

            if Targets[Count] == UserInputEvaluation:

                SuccessfulTarget = Targets[Count]

                Score += 2

                Targets[Count] = -1

                UserInputEvaluationIsATarget = True

                print("Target Successfully hit")

                UserChoice = input(f"Would you like to add either the successful target {SuccessfulTarget} or parts of it to the   
Allowed Numbers List? Y/N").upper()

                if UserChoice == "Y":

                    TargetoAddToNumbersAllowed = SelectValueFromTarget(SuccessfulTarget)

    return TargetoAddToNumbersAllowed, UserInputEvaluationIsATarget, Score

    #END CHANGE

Creation of a new SelectValueFromTarget method:

#CHANGE

def SelectValueFromTarget(SuccessfulTarget):

    FirstDigit = SuccessfulTarget // 10

    SecondDigit = SuccessfulTarget % 10

    print("Please select from the options below which part of the target or the target itself you would like to include in the   
Allowed Numbers:")

    print(f"1: The first digit: {FirstDigit}")

    print(f"2: The second digit: {SecondDigit}")

    print(f"3: The whole target: {SuccessfulTarget}")

    UserChoice = int(input())

    if UserChoice == 1:

        return FirstDigit

    elif UserChoice == 2:

        return SecondDigit

    elif UserChoice == 3:

        return SuccessfulTarget

    else:

        return -1

#END CHANGE

Changes to the FillNumbers method:

#CHANGE

def FillNumbers(NumbersAllowed, TrainingGame, MaxNumber, TargetToAddToNumbersAllowed = -1):

    if TargetToAddToNumbersAllowed == -1:

        if TrainingGame:

            return [2, 3, 2, 8, 512]

        else:

            while len(NumbersAllowed) < 5:

                NumbersAllowed.append(GetNumber(MaxNumber))

            return NumbersAllowed

    else:

        if TrainingGame:

            return [2, 3, 2, 8, 512, TargetToAddToNumbersAllowed]

        else:

            while len(NumbersAllowed) < 5:

                NumbersAllowed.append(GetNumber(MaxNumber))

            NumbersAllowed.append(TargetToAddToNumbersAllowed)

            return NumbersAllowed

#END CHANGE

## Testing

* A screenshot of a computer

  Description automatically generatedSelecting either a correctly identified target or a component part of it and adding it to the NumbersAllowed list.   
  [1 mark] ▶

# Task 8 (8 marks)

## Coding

* 25% chance of activating the challenge mode. [1 mark]
* Prompt advising the user about the challenge. [1 mark]
* Suitable variable to identify a challenge mode in this turn. [1 mark]
* Suitable counter to identify how many numbers are included in the expression. [1 mark]
* Iterate through the expression to count numbers, correctly incrementing when numbers are found. [1 mark]
* Selection to identify if the challenge has been achieved. [1 mark]
* Correct awarding or deduction of points in both scenarios. [1 mark]

**Teacher Notes:**

*Reduction of the score by 1 between each turn should still happen regardless of whether the user has been awarded a bonus or a penalty.*

### Example Solution

Changes to the PlayGame method:

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        if (random.randint(0,3) == 1):

            print("Challenge Mode Activated. In this turn you must use all 5 numbers.")

            print("If you achieve this you will be awarded an extra 10 points.")

            print("If you do not achieve this, your score will be reduced by 5 points.")

            ChallengeMode = True

        UserInput = input("Enter an expression: ")

        print()

        if CheckIfUserInputValid(UserInput):

            UserInputInRPN = ConvertToRPN(UserInput)

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score, ChallengeMode)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

        #END CHANGE

        Score -= 1

Modification of the CheckIfUserInputEvaluationIsATarget method:

#CHANGE

def CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score, ChallengeMode):

    NumbersUsed = 0

    for s in UserInputInRPN:

        if s.isdigit(): #str.isdigit() doesn't pick up negative numbers, but works for this program

            NumbersUsed += 1

    UserInputEvaluation = EvaluateRPN(UserInputInRPN)

    UserInputEvaluationIsATarget = False

    if UserInputEvaluation != -1:

        for Count in range(0, len(Targets)):

            if Targets[Count] == UserInputEvaluation:

                Score += 2

                Targets[Count] = -1

                UserInputEvaluationIsATarget = True

        if UserInputEvaluationIsATarget and ChallengeMode and NumbersUsed == 5:

            print("Challenge Mode achieved. You are awarded an additional 10 points")

            print()

            Score += 10

        elif UserInputEvaluationIsATarget and ChallengeMode and NumbersUsed < 5:

            print("Challenge Mode not achieved. 5 points are deducted from your score.")

            print()

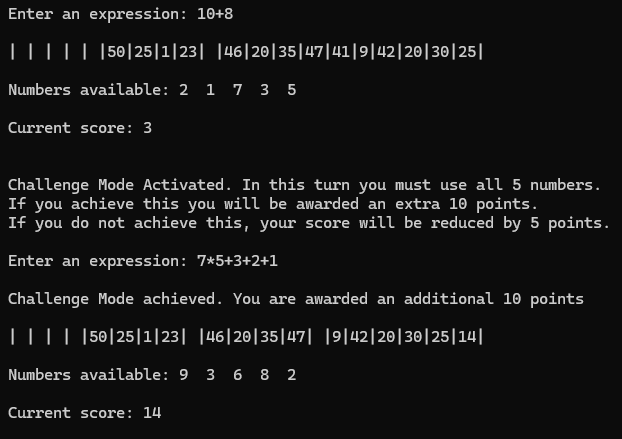
            Score -= 5

    return UserInputEvaluationIsATarget, Score

    #END CHANGE

## Testing

* Awarding points correctly for a successful expression which involves all five values in the NumbersAllowed list. [1 mark]



# Task 9 (8 marks)

## Coding

* Suitable variable to store the position of a target to be frozen. [1 mark]
* Prompt the user to freeze or unfreeze the position of a target dependent on the current state of the game. [1 mark]
* Correct updating of current game state. [1 mark]
* Modification of the UpdateTargets method to compare targets with a frozen target. [1 mark]
* Correct updating of the Targets list without moving a frozen target. [1 mark]
* Comparison of a frozen target when being displayed on the screen. [1 mark]
* Correct display of frozen target. [1 mark]

### Example Solution

Modification to the PlayGame method:

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber):

    #CHANGE

    FrozenPosition = -1

    Score = 0

    GameOver = False

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score, FrozenPosition)

        if FrozenPosition == -1:

            UserChoice = input("Would you like to freeze the position of one of the targets? Y/N ").upper()

            if UserChoice == "Y":

                FrozenPosition = int(input("Please enter in a position to freeze between 1 and 20:")) - 1

                DisplayState(Targets, NumbersAllowed, Score, FrozenPosition)

        else:

            UserChoice = input(f"Would you like to unfreeze the position of target {FrozenPosition + 1}? Y/N").upper()

            if UserChoice == "Y":

                FrozenPosition = -1

                DisplayState(Targets, NumbersAllowed, Score, FrozenPosition)

        UserInput = input("Enter an expression: ")

        print()

        if CheckIfUserInputValid(UserInput):

            UserInputInRPN = ConvertToRPN(UserInput)

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

        Score -= 1

        if Targets[0] != -1:

            GameOver = True

        else:

            Targets = UpdateTargets(Targets, TrainingGame, MaxTarget, FrozenPosition)

    print("Game over!")

    DisplayScore(Score)

    #END CHANGE

Changes to UpdateTargets method:

#CHANGE

def UpdateTargets(Targets, TrainingGame, MaxTarget, FrozenPosition):

    for Count in range (0, len(Targets) - 1):

        if Count == FrozenPosition:

            Targets[Count - 1] = Targets[Count + 1]

        else:

            Targets[Count] = Targets[Count + 1]

    Targets.pop()

    if TrainingGame:

        Targets.append(Targets[-1])

    else:

        Targets.append(GetTarget(MaxTarget))

    return Targets

#END CHANGE

Changes to multiple methods to display the targets on the screen:

#CHANGE

def DisplayState(Targets, NumbersAllowed, Score, FrozenPosition):

    DisplayTargets(Targets, FrozenPosition)

    DisplayNumbersAllowed(NumbersAllowed)

    DisplayScore(Score)

#END CHANGE

#CHANGE

def DisplayTargets(Targets, FrozenPosition):

    Counter = 0

    print("|", end = '')

    for T in Targets:

        if T == -1:

A screenshot of a computer

Description automatically generated            print(" ", end = '')

        else:

            if Counter == FrozenPosition:

                print(f"<{T}>", end = '')

            else:

                print(T, end = '')

        print("|", end = '')

        Counter += 1

    print()

    print()

#END CHANGE

## Testing

* Showing a target being frozen so others move around it. [1 mark] ▶

# Task 10 (11 marks)

## Coding

1. Creation of an UndoState class. [1 mark]
2. Suitable attributes to store the Targets, NumbersAllowed and Score. [1 mark]
3. Constructor which assigns initial values for the Targets, NumbersAllowed and Score. [1 mark]
4. Suitable accessor method to expose the class properties. [1 mark]
5. Suitable data structure to store undo states which allows LIFO access. [1 mark]
6. Advise the user on how many undos are available, if any are. [1 mark]
7. Correctly storing a copy of the Score variable in an undo state. [1 mark]
8. Correctly storing a copy of the Targets and NumbersAllowed lists in an undo state. [1 mark]
9. Correctly restore the Score when undo selected. [1 mark]
10. Correctly restore the Targets and NumbersAllowed lists when undo selected. [1 mark]

**Teacher Notes:**

*This functionality could be completed using a suitable data structure such as a List or Stack of Tuples containing the Targets list, NumbersAllowed list and Score, rather than a list of objects. Marks B–D can be awarded for suitable list management to access elements in a Stack of Tuples, but full marks should only be awarded if objects are used.*

### Example Solution

Modification of the PlayGame method:

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber):

    #CHANGE

    UndoTurns = []

    #END CHANGE

    Score = 0

    GameOver = False

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        if len(UndoTurns) > 0:

            UserChoice = input(f"There are {len(UndoTurns)} Undos left. Would you like to Undo the last turn? Y/N").upper()

            if UserChoice == "Y":

                UndoTurns, NumbersAllowed, Targets, Score = UndoLastTurn(UndoTurns, NumbersAllowed, Targets, Score)

                DisplayState(Targets, NumbersAllowed, Score)

        UndoTurns = AddToUndo(UndoTurns, NumbersAllowed, Targets, Score)

        #END CHANGE

        UserInput = input("Enter an expression: ")

Creation of new Method UndoLastTurn:

#CHANGE

def UndoLastTurn(UndoTurns, NumbersAllowed, Targets, Score):

    Targets = []

    NumbersAllowed = []

    LastMove = UndoTurns.pop()

    Score = LastMove.GetScore()

    Targets = LastMove.GetTargets()

    NumbersAllowed = LastMove.GetNumbersAllowed()

    return UndoTurns,NumbersAllowed,Targets,Score

Creation of new Method AddToUndo:

def AddToUndo(UndoTurns, NumbersAllowed, Targets, Score):

    LastTargetsState = []

    LastNumbersAllowedState = []

    for Item in Targets:

        LastTargetsState.append(Item)

    for Item in NumbersAllowed:

        LastNumbersAllowedState.append(Item)

    UndoTurns.append(UndoState(LastTargetsState,LastNumbersAllowedState,Score))

    return UndoTurns

#END CHANGE

Creation of new class UndoState:

#CHANGE

class UndoState():

    def \_\_init\_\_(self, Targets, NumbersAllowed, Score):

        self.\_\_Targets = Targets

        self.\_\_NumbersAllowed = NumbersAllowed

        self.\_\_Score = Score

    def GetTargets(self):

        return self.\_\_Targets

    def GetNumbersAllowed(self):

        return self.\_\_NumbersAllowed

    def GetScore(self):

        return self.\_\_Score

#END CHANGE

## Testing

* Demonstrating the Targets list and NumbersAllowed list and Score being restored to their state from the previous turn. [1 mark]

A screenshot of a computer program

Description automatically generated

# Task 11 (12 marks)

## Coding

* Prompt the user if they would like random suggestions for solutions. [1 mark]
* Suitable data structure to store solution suggestions without duplication. [1 mark]
* Suitable variable to store the number of suggestions attempted to ensure a cap of 30. [1 mark]
* Condition-controlled iteration to test solutions. [1 mark]
* Suitable variable to store two values from the NumbersAllowed list. [1 mark]
* Random selection of mathematical operator (only need +, -, /, \*). [1 mark]
* Test if the solution found is a target. [1 mark]
* Correct testing of expression after division to ensure the evaluation is an integer. [1 mark]
* Adding correctly identified targets to storage data structure. [1 mark]
* Selection of third operand from NumbersAllowed list and perform mathematical operation with result of first expression evaluation. [1 mark]
* Correctly displaying suggested targets with \* \* around them. [1 mark]

**Teacher Notes:**

*Candidates should not attempt to find every possible solution using the NumbersAllowed, but instead should simply select two or three operands and operators at random to create an expression within the BIDMAS limitations of the pre-release code. The expression should be evaluated and tested against the Targets list. Once five targets are found, the method should stop and display what it has found. If five possible targets are not identified within 30 attempts, the method should stop looking and display what it has found.*

### Example Solution

Modification of PlayGame method:

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber):

    Score = 0

    GameOver = False

    while not GameOver:

        #CHANGE

        UserChoice = input("Would you like up to five random suggestions of solutions involving only 2 or 3 operands? Y/N").upper()

        print()

        if UserChoice == "Y":

            IdentifiedTargets = GetRandomSuggestions(NumbersAllowed, Targets)

            DisplayState(Targets, NumbersAllowed, Score, IdentifiedTargets)

        else:

            DisplayState(Targets, NumbersAllowed, Score)

        #END CHANGE

Creation of new GetRandomSuggestions method:

#CHANGE

def GetRandomSuggestions(NumbersAllowed, Targets):

    Suggestions = set()

    SolutionsTested = 0

    while (len(Suggestions) < 5):

        Temp = []

        for Item in NumbersAllowed:

            Temp.append(Item)

        FirstOperand = Temp[random.randint(0,len(Temp)-1)]

        Temp.remove(FirstOperand)

        SecondOperand = Temp[random.randint(0,len(Temp)-1)]

        Temp.remove(SecondOperand)

        Operator = random.randint(0,3)

        Result = 0

        if Operator == 0:

            Result = FirstOperand + SecondOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 1:

            Result = FirstOperand - SecondOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 2:

            if float(FirstOperand / SecondOperand) - math.floor(FirstOperand / SecondOperand) == 0.0:

                Result = math.floor(FirstOperand / SecondOperand)

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 3:

            Result = FirstOperand \* SecondOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        ThirdOperand = Temp[random.randint(0,len(Temp)-1)]

        Operator = random.randint(0,3)

        if Operator == 0:

            Result = Result + ThirdOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 1:

            Result = Result - ThirdOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 2:

            if float(Result / ThirdOperand) - math.floor(Result / ThirdOperand) == 0.0:

                Result = math.floor(Result / ThirdOperand)

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        elif Operator == 3:

            Result = Result \* ThirdOperand

            if (Result in Targets and Result != -1):

                Suggestions.add(Result)

        SolutionsTested += 1

        if SolutionsTested > 30:

            break

    return Suggestions

#END CHANGE

## Testing

* Displaying correct suggestion targets in the Targets list. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 12 (7 marks)

## Coding

* Suitable variable to store the evaluation of an expression entered by the user. [1 mark]
* Suitable variable to store the number of targets which the evaluation matches. [1 mark]
* Advising the user of the evaluation of their expression and the number of target matches. [1 mark]
* Suitable data structure to store invalid numbers in a user expression. [1 mark]
* Advising the user which numbers in their expression are invalid (if required). [1 mark]

### Example Solution

Modification to the CheckIfUserInputEvaluationIsATarget method:

def CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score):

    #CHANGE

    TargetsFound = 0

    UserInputEvaluation = EvaluateRPN(UserInputInRPN)

    UserInputEvaluationIsATarget = False

    if UserInputEvaluation != -1:

        for Count in range(0, len(Targets)):

            if Targets[Count] == UserInputEvaluation:

                Score += 2

                Targets[Count] = -1

                UserInputEvaluationIsATarget = True

                TargetsFound += 1

    print(f"Your expression evaluated to {UserInputEvaluation} and was found {TargetsFound} times.")

    print()

    return UserInputEvaluationIsATarget, Score

    #END CHANGE

Creation of new CheckNumbersUsedAreAllInNumbersAllowed method:

def CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

    #CHANGE

    AllNumbersValid = True

    InvalidNumbers = []

    Temp = []

    for Item in NumbersAllowed:

        Temp.append(Item)

    for Item in NumbersAllowed:

        Temp.append(Item)

    for Item in UserInputInRPN:

        if CheckValidNumber(Item, MaxNumber):

            if int(Item) in Temp:

                Temp.remove(int(Item))

            else:

                InvalidNumbers.append(int(Item))

                AllNumbersValid = False

    if not AllNumbersValid:

        print("Your expression is not valid because the following numbers are not available to you:")

        for Item in InvalidNumbers:

          print(str(Item) + ",  ", end = '')

        print()

        print()

    return AllNumbersValid

    #END CHANGE

## Testing

* Advising the user that they have entered an invalid number in their expression. [1 mark]

A screenshot of a computer

Description automatically generated

* Advising the user that they have found a valid target and how many times it appears in the Targets list. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 13 (7 marks)

## Coding

* Use of MaxNumberOfTargets variable as upperbound when repopulating the Targets list. **[1 mark]**
* Suitable counting variable to count the number of targets to be removed / calculate the size of slice. **[1 mark]**
* Suitable technique for identifying second target. **[1 mark]**
* Removing slice / range from Targets list. **[1 mark]**
* Targets list repopulated to the correct length. **[1 mark]**
* Targets list repopulated with correct values for both training game and standard random game. **[1 mark]**

**Teacher Notes:**

*Removal of targets from the Targets list could also be achieved through iteration.*

### Example Solution

Modification to the Main method:

    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

    #CHANGE

    PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, MaxNumberOfTargets)

    #END CHANGE

    input()

Modification of the PlayGame method:

#CHANGE

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, MaxNumberOfTargets):

#END CHANGE

    Score = 0

…

        else:

            #CHANGE

            Targets = UpdateTargets(Targets, TrainingGame, MaxTarget, MaxNumberOfTargets)

            #END CHANGE

    print("Game over!")

    DisplayScore(Score)

Modification of the CheckIfUserInputEvaluationIsATarget method:

    if UserInputEvaluation != -1:

        Count = 0

        while Count < len(Targets):

            if Targets[Count] == UserInputEvaluation:

                Score += 2

                #CHANGE

                TargetsToRemove = 0

                for i in range(Count + 1, len(Targets)):

                    TargetsToRemove += 1

                    if Targets[i] == UserInputEvaluation:

                        Score += TargetsToRemove \* 2

                        del Targets[Count : Count + TargetsToRemove]

                        break

                del Targets[Count]  #Remove the first one

                UserInputEvaluationIsATarget = True

                break

            Count += 1

    return UserInputEvaluationIsATarget, Score

    #END CHANGE

Modification of the UpdateTargets method:

#CHANGE

def UpdateTargets(Targets, TrainingGame, MaxTarget, MaxNumberOfTargets):

    for Count in range (0, len(Targets) - 1):

        Targets[Count] = Targets[Count + 1]

    Targets.pop()

    if TrainingGame:

        while (len(Targets) < MaxNumberOfTargets):

            Targets.append(Targets[-1])

    else:

        while (len(Targets) < MaxNumberOfTargets):

            Targets.append(GetTarget(MaxTarget))

    return Targets

#END CHANGE

## Testing

* Show all the values between the two 34 targets (inclusive) being removed, Targets list being backfilled with 119s and the score increasing to 9 points. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 14 (5 marks)

## Coding

* Update the Precedence dictionary to correctly reference the ^ symbol. [1 mark]
* Correct checking for the ^ symbol when evaluating RPN. [1 mark]
* Correct use of mathematical operation in evaluating the RPN. [1 mark]
* Correct inclusion of the escaped ^ symbol in the Regular Expression checking if the input is valid. [1 mark]

### Example Solution

Modification of the ConvertToRPN method:

def ConvertToRPN(UserInput):

    Position = 0

    #CHANGE

    Precedence = {"+": 2, "-": 2, "\*": 4, "/": 4, "^": 6}

    #END CHANGE

Modification of the EvaluateRPN method:

def EvaluateRPN(UserInputInRPN):

    S = []

    while len(UserInputInRPN) > 0:

        #CHANGE

        while UserInputInRPN[0] not in ["+", "-", "\*", "/", "^"]:

            S.append(UserInputInRPN[0])

            UserInputInRPN.pop(0)

        Num2 = float(S[-1])

        S.pop()

        Num1 = float(S[-1])

        S.pop()

        Result = 0.0

        if UserInputInRPN[0] == "+":

            Result = Num1 + Num2

        elif UserInputInRPN[0] == "-":

            Result = Num1 - Num2

        elif UserInputInRPN[0] == "\*":

            Result = Num1 \* Num2

        elif UserInputInRPN[0] == "/":

            Result = Num1 / Num2

        elif UserInputInRPN[0] == "^":

            Result = math.pow(Num1,Num2)

        #END CHANGE

        UserInputInRPN.pop(0)

        S.append(str(Result))

    if float(S[0]) - math.floor(float(S[0])) == 0.0:

        return math.floor(float(S[0]))

    else:

        return -1

Modification of the CheckIfUserInputValid method:

def CheckIfUserInputValid(UserInput):

    #CHANGE

    if re.search("^([0-9]+[\+\-\\*\/\^])+[0-9]+$", UserInput) is not None:

        return True

    else:

        return False

A screenshot of a computer

Description automatically generated    #END CHANGE

## Testing

* Correct identification of the target 119 using indices.   
  [1 mark] ▶

# Task 15 (10 marks)

## Coding

* Suitable adjustment of scope of the Score variable to allow it to be updated in a previous game state load. [1 mark]
* Prompt inviting the user to load a previous game state. [1 mark]
* Correct assignment of the TrainingGame, MaxNumber and MaxTarget variables in all scenarios. [1 mark]
* Condition-controlled loop while invalid code is entered by the user. [1 mark]
* Correct checking that the code only contains valid characters. *This could be done with string handling, ASCII conversion or Regular Expression*. [1 mark]
* Correct checking that the restore code is the right length. [1 mark]
* Correct checking that NumbersAllowed and Score portions of the code do not contain the @ symbol. [1 mark]
* Correct restoration of the Targets list from a previous game state code. [1 mark]
* Correct restoration of the NumbersAllowed list and Score from a previous game state code. [1 mark]

### Example Solution

Modification of the Main method:

    #CHANGE

    Score = 0

    Choice = input("Enter y to play the training game, L to restore a previous game state, anything else to play a random game: ").upper()

    print()

    if Choice == "Y":

        MaxNumber = 1000

        MaxTarget = 1000

        TrainingGame = True

        Targets = [-1, -1, -1, -1, -1, 23, 9, 140, 82, 121, 34, 45, 68, 75, 34, 23, 119, 43, 23, 119]

    elif Choice == "L":

        Targets, NumbersAllowed, Score = RestoreGame(Targets, NumbersAllowed, Score)

        TrainingGame = False

        MaxNumber = 10

        MaxTarget = 50

    else:

        MaxNumber = 10

        MaxTarget = 50

        Targets = CreateTargets(MaxNumberOfTargets, MaxTarget)

    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

    PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, Score)

    input()

    #END CHANGE

Creation of the RestoreGame method:

#CHANGE

def RestoreGame(Targets, NumbersAllowed, Score):

    RestoreCode = ""

    ValidCode = False

    while (not ValidCode):

        RestoreCode = input("Enter in the game restore code. It should be 26 characters.")

        if re.search("^[@A-z[\\]^\_']+$", RestoreCode) is not None:

            if len(RestoreCode) == 26:

                if "@" in RestoreCode[20:]:

                    print("That is not a valid code")

                else:

                    ValidCode = True

            else:

                print("That code is not the right length")

        else:

            print("That code contains invalid characters")

    for i in range(20):

        if ord(RestoreCode[i]) - 64 == 0:

            Targets.append(-1)

        else:

            Targets.append(ord(RestoreCode[i]) - 64)

    for i in range(20,25):

        NumbersAllowed.append(ord(RestoreCode[i]) - 64)

    Score = ord(RestoreCode[25]) - 64

    print()

    return Targets, NumbersAllowed, Score

#END CHANGE

Modification of the PlayGame method:

#CHANGE

def PlayGame(Targets, NumbersAllowed, TrainingGame, MaxTarget, MaxNumber, Score):

    #END CHANGE (removed Score from scope and passing in as a parameter)

    GameOver = False

    while not GameOver:

## Testing

* Displaying game restore code and correctly restored game. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 16 (12 marks)

## Coding

* Use of a Stack data structure to store operators. [1 mark]
* Use of a Queue data structure to store the postfix notation. [1 mark]
* Conversion of the infix expression into individual elements. *This could also be done using iteration.* [1 mark]
* Data structure to suitably handle precedence to correctly handle open parenthesis. [1 mark]
* Correctly pushing operators onto a stack in the right order according to precedence. [1 mark]
* Correctly recognising the start and end brackets in an expression. [1 mark]
* Correctly popping items from the stack to remove all items inside brackets and enqueue them in the right order. [1 mark]
* Correctly removing operators from the stack without underflowing. [1 mark]
* Correctly enqueuing operands into a queue data structure. [1 mark]
* Correctly matching the parenthesis symbols in the CheckValidOperator method. [1 mark]
* Correct modification of the Regular Expression to match an infix expression with brackets. [1 mark]

**Teacher Notes:**

*Using only the Regular Expression meta characters from the AQA specification, it is not possible to count the number of brackets used in an infix expression to ensure that it is mathematically correct. Whilst it could be done with backtracking, that is not a requirement for this question and candidates should not be credited for it. The candidates can assume a mathematically valid input.*

*The suggested solution uses a new method ConvertToRPNWithBrackets rather than modifying the existing ConvertToRPN method because the number of changes which were needed would mean the original method would need significant modifications.*

### Example Solution

Replacement of the ConvertToRPN method with the new ConvertToRPNWithBrackets method wherever it is called:

##PlayGame

if CheckIfUserInputValid(UserInput):

            #CHANGE

            UserInputInRPN = ConvertToRPNWithBrackets(UserInput, MaxNumber)

            #END CHANGE

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

…

##RemoveNumbersUsed

def RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed):

    #CHANGE

    UserInputInRPN = ConvertToRPNWithBrackets(UserInput, MaxNumber)

    #END CHANGE

    for Item in UserInputInRPN:

        if CheckValidNumber(Item, MaxNumber):

            if int(Item) in NumbersAllowed:

                NumbersAllowed.remove(int(Item))

    return NumbersAllowed

Creation of new ConvertToRPNWithBrackets method:

#CHANGE

def ConvertToRPNWithBrackets(UserInput, MaxNumber):

    Operators = []

    RPNQueue = []

    FullInfixExpression = re.split("(\D)", UserInput)

    CleanedInfixExpression = []

    for Item in FullInfixExpression:

        if Item is not None and Item != "":

            CleanedInfixExpression.append(Item)

    Precedence = {"(": 1, "+": 2, "-": 2, "\*": 4, "/": 4}

    for Item in CleanedInfixExpression:

        if CheckValidOperator(Item):

            if Item == "(":

                Operators.append(Item)

            elif Item == ")":

                while len(Operators) > 0:

                    if Operators[-1] != "(":

                        RPNQueue.append(Operators.pop())

                    else:

                        Operators.pop()

                        break

            elif len(Operators) > 0 and Precedence[Operators[-1]] > Precedence[Item]:

                RPNQueue.append(Operators.pop())

                Operators.append(Item)

            else:

                Operators.append(Item)

        elif CheckValidNumber(Item, MaxNumber):

            RPNQueue.append(Item)

    while len(Operators) > 0:

        RPNQueue.append(Operators.pop())

    return RPNQueue

#END CHANGE

Modification of the Regular Expression in the CheckValidOperator method:

#CHANGE

def CheckValidOperator(Item):

    if re.search("[\+\-\\*\/\(\)]", Item) is not None:

        return True

    else:

        return False

#END CHANGE

Modification of the Regular Expression in the CheckIfUserInputValid method:

//CHANGE

static bool CheckIfUserInputValid(string UserInput)

{

return Regex.IsMatch(UserInput, @"^\(\*([0-9]+|[\+\-\\*\/\(\)])+\)\*$");

}

//END CHANGE

## Testing

* Showing a target correctly identified using an infix expression including brackets. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 17 (6 marks)

## Coding

* Prompt to allow the user to choose to enter their expression using infix or postfix notation. [1 mark]
* Suitable variable to store if the game will be operating with postfix or infix notation. [1 mark]
* Splitting the postfix notation using the delimiter *(any different delimiter is acceptable, including a space, as long as it is valid).* [1 mark]
* Correct modification of the CheckIfUserInputValid method to accept any RPN input *(because it will be tested in the EvaluateRPN method)*. [1 mark]
* Ensuring that the program can still match a standard infix expression. [1 mark]

**Teacher Notes:**

*Creating a Regular Expression that matches any valid RPN input is quite challenging due to the complexity and variability of RPN expressions. Regular Expressions are powerful for pattern matching but have limitations when it comes to parsing nested or highly variable structures like RPN. For this reason, the suggested solution simply accepts the RPN input from the user as valid because the EvaluateRPN method will evaluate the result so that it can be compared against targets.*

*Space was not used as the delimiter for this suggested solution because it would require Regular Expression meta characters which are beyond the 7517 specification.*

### Example Solution

Modification of the PlayGame method:

while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        UserChoice = input("Enter 1 if you would like to give your expression in Infix Notation and 2 to give it in RPN: ")

        RPNInput = False

        if UserChoice == "2":

            print()

            print("Enter your RPN using a COMMA between the operators and operands")

            print()

            RPNInput = True

        UserInput = input("Enter an expression: ")

        print()

        if CheckIfUserInputValid(UserInput, RPNInput):

            UserInputInRPN = ConvertToRPN(UserInput, RPNInput)

            if CheckNumbersUsedAreAllInNumbersAllowed(NumbersAllowed, UserInputInRPN, MaxNumber):

                IsTarget, Score = CheckIfUserInputEvaluationIsATarget(Targets, UserInputInRPN, Score)

                if IsTarget:

                    NumbersAllowed = RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed, RPNInput)

                    NumbersAllowed = FillNumbers(NumbersAllowed, TrainingGame, MaxNumber)

        #END CHANGE

        Score -= 1

Modification of the RemoveNumbersUsed method:

#CHANGE

def RemoveNumbersUsed(UserInput, MaxNumber, NumbersAllowed, RPNInput):

    UserInputInRPN = ConvertToRPN(UserInput, RPNInput)

    #END CHANGE

Modification of the ConvertToRPN method:

#CHANGE

def ConvertToRPN(UserInput, RPNInput):

    if RPNInput:

        Temp = UserInput.split(',')

        return Temp

    #END CHANGE

    Position = 0

Modification to the CheckIfUserInputValid method:

#CHANGE

def CheckIfUserInputValid(UserInput, RPNInput):

    if RPNInput:

            return True

    elif re.search("^([0-9]+[\\+\\-\\\*\\/])+[0-9]+$", UserInput) is not None:

        return True

    else:

        return False

#END CHANGE

## Testing

* Showing all three 23 targets removed from the Targets list and the score of 5. [1 mark]

A screenshot of a computer

Description automatically generated

# Task 18 (5 marks)

## Coding

* Suitable data structure for creation / update of Targets list which does not allow duplication of values. [1 mark]
* Condition-controlled loop to initially populate the Targets list until it contains five unique values. [1 mark]
* Condition-controlled loop to repopulate the Targets list (after a target has been found) until it contains 20 unique values. [1 mark]
* Condition-controlled loop to repopulate the NumbersAllowed list (after a target has been found) until it contains five unique values. [1 mark]

### Example Solution

Modification of the UpdateTargets method:

def UpdateTargets(Targets, TrainingGame, MaxTarget):

    for Count in range (0, len(Targets) - 1):

        Targets[Count] = Targets[Count + 1]

    Targets.pop()

    if TrainingGame:

        Targets.append(Targets[-1])

    else:

        #CHANGE

        while len(Targets) < 20:

            NewTarget = GetTarget(MaxTarget)

            if NewTarget not in Targets:

                Targets.append(NewTarget)

        #END CHANGE

    return Targets

Modification of CreateTargets method:

def CreateTargets(SizeOfTargets, MaxTarget):

    Targets = []

    for Count in range(1, 6):

        Targets.append(-1)

    #CHANGE

    UniqueNumbers = set()

    while len(UniqueNumbers) < SizeOfTargets - 5:

        UniqueNumbers.add(GetTarget(MaxTarget))

    Targets.extend(UniqueNumbers)

    #END CHANGE

    return Targets

Modification of the FillNumbers method:

def FillNumbers(NumbersAllowed, TrainingGame, MaxNumber):

    if TrainingGame:

        return [2, 3, 2, 8, 512]

    else:

        #CHANGE

        UniqueNumbers = set()

        while len(UniqueNumbers) < 5:

            UniqueNumbers.add(GetNumber(MaxNumber))

        return list(UniqueNumbers)

        #END CHANGE

## Testing

* Showing unique values in the Targets list and NumbersAllowed list before and after a turn. [1 mark]

**A screenshot of a computer

Description automatically generated**

# Task 19 (14 marks)

## Coding

* Prompt to ask the user if they would like helper suggestions. **[1 mark]**
* Selection to branch program appropriately depending on their choice to display helper suggestions. **[1 mark]**
* Suitable data structure to store text expressions and associated evaluations. **[1 mark]**
* Count-controlled loop to iterate through data structure storing text expressions and associated evaluations. [1 mark]
* Iterating through the NumbersAllowed list to test permutations. **[1 mark]**
* Rotating the NumbersAllowed list (or similar functionality) to test different permutations of numbers. [1 mark]
* Appropriately displaying the combination of text expressions and associated evaluations on the screen for the user. [1 mark]
* Use of recursion to try combinations. **[1 mark]**
* Only storing suggestion solutions for targets which have not already been identified. **[1 mark]**
* Correctly calculating expressions which use division to ensure they evaluate to an integer. **[1 mark]**
* Testing expressions to ensure they correctly follow BIDMAS if needed (required for expressions built up through recursion). **[1 mark]**
* Generate expressions which can use the four mathematical operators: + - / \* **[1 mark]**
* Storage of expression with associated evaluation. **[1 mark]**

**Teacher Notes:**

*This functionality could be completed using iteration. Marks should be awarded for techniques, but full marks should only be awarded if recursion is used.*

*Because the expression is built up step by step, it must be tested at each stage because the impact of BIDMAS may change the evaluation as the expression builds.*

### Example Solution

Modification of the PlayGame method:

    while not GameOver:

        DisplayState(Targets, NumbersAllowed, Score)

        #CHANGE

        UserChoice = input("Would you like helper suggestions: Y/N ").upper()

        if UserChoice == "Y":

            Temp = []

            PossibleSolutions = {}

            for Item in NumbersAllowed:

                Temp.append(Item)

            for i in range(5):

                TestSolutions = GenerateEvaluations(Temp, Targets)

                for key, value in TestSolutions.items():

                    if key not in PossibleSolutions:

                        PossibleSolutions[key] = value

                Temp.append(Temp[0])

                del Temp[0]

            print()

            for key, value in PossibleSolutions.items():

                print(f"{key} can be calculated using the expression: {value}")

            print()

        #END CHANGE

        UserInput = input("Enter an expression: ")

        print()

Creation of new GenerateEvaluations method (and associated helper method):

#CHANGE

def GenerateEvaluations(NumbersAllowed, Targets):

    PossibleExpressions = {}

    GenerateEvaluationsHelper(NumbersAllowed, Targets, 0, NumbersAllowed[0], PossibleExpressions, str(NumbersAllowed[0]))

    return PossibleExpressions

def GenerateEvaluationsHelper(NumbersAllowed, Targets, Index, CurrentResult, PossibleExpressions, CurrentExpression):

    if Index == len(NumbersAllowed)-1:

        #Because the recursion calculates expressions step by step rather than as one whole expression

        #the new code needs to test the end result using RPN evaluator to ensure BIDMAS is correctly followed

        if EvaluateRPN(ConvertToRPN(CurrentExpression)) in Targets and EvaluateRPN(ConvertToRPN(CurrentExpression)) != -1:

            if EvaluateRPN(ConvertToRPN(CurrentExpression)) not in PossibleExpressions:

                PossibleExpressions[EvaluateRPN(ConvertToRPN(CurrentExpression))] = CurrentExpression

        return

    NextNumber = NumbersAllowed[Index + 1]

    GenerateEvaluationsHelper(NumbersAllowed, Targets, Index + 1, CurrentResult \* NextNumber, PossibleExpressions, f"{Current  
Expression}\*{NextNumber}")

    if NextNumber != 0:

        if float(CurrentResult / NextNumber) - math.floor(float(CurrentResult / NextNumber)) == 0.0:

            GenerateEvaluationsHelper(NumbersAllowed, Targets, Index + 1, math.floor(CurrentResult / NextNumber), Possible  
Expressions, f"{CurrentExpression}/{NextNumber}")

    GenerateEvaluationsHelper(NumbersAllowed, Targets, Index + 1, CurrentResult + NextNumber, PossibleExpressions, f"{Current  
Expression}+{NextNumber}")

    GenerateEvaluationsHelper(NumbersAllowed, Targets, Index + 1, CurrentResult - NextNumber, PossibleExpressions, f"{Current  
Expression}-{NextNumber}")

#END CHANGE

## Testing

* Show the program displaying the suggested valid expressions for targets. **[1 mark]**

A screenshot of a computer

Description automatically generated