|  |
| --- |
| import random  MAX\_LINES = 7  MAX\_BET = 1000  MIN\_BET = 5  ROWS = 5  COLS = 5 |
| symbol\_count={      "A": 10,      "B": 12,      "C": 16,      "D": 20,      "E": 24,      "\*": 5,      "@": 5  } |
| symbol\_value={      "A": 5,      "B": 4.5,      "C": 4,      "D": 3.5,      "E": 3,      "\*": 2,  "@": 2  } |

Slot machine with Python.

In this document, the process of creating a ***primitive slot machine*** with Python will be documented.

**Step №1: Setting up the necessary values for the program to run correctly.**

In this step, as shown in the photo, the following parameters are set in place:

* ***MAX\_LINES*** – indicating the maximum playable lines, the slot machine offers the customer (*5 rows + 2 diagonal rows*).
* ***MAX\_BET*** – setting a limit on the maximum bet a customer can place for each line (the total bet will be calculated by multiplying the placed bet for each line by the number of lines the user chose to play.
* ***MIN\_BET*** – setting a minimum bet per line that the customer can place.
* ***ROWS*** – number of active rows in the slot machine.
* ***COLS*** – number of active collumns in the slot machine.
* ***SYMBOL\_COUNT*** – A dictionary of all the symbols the slot machine can generated and the number of symbols each collumn has.
* ***SYMBOL\_VALUE*** – A dictionary of the values for each symbol the slot machine can generate. Used for calculating the winnings if the customer has won the game.

*(All of the variables can be modified later in the future for improvemets or expansion.)*

After defining the variables above, we can use the “***import random***” module in order for the program to randomly select items from the dictionary “**symbol\_count**” and print them on the screen.

**Step №2: Defining the deposit amount.**

The amount that the user of the program will be plaing with will be defined in the the “***Def deposit ():***” function.

def deposit():

    while True:

        amount = input("What would you like to deposit? $")

        if amount.isdigit():

            amount = int(amount)

            if amount > 0:

                break

            else:

                print("The deposit must be greater than 0$")

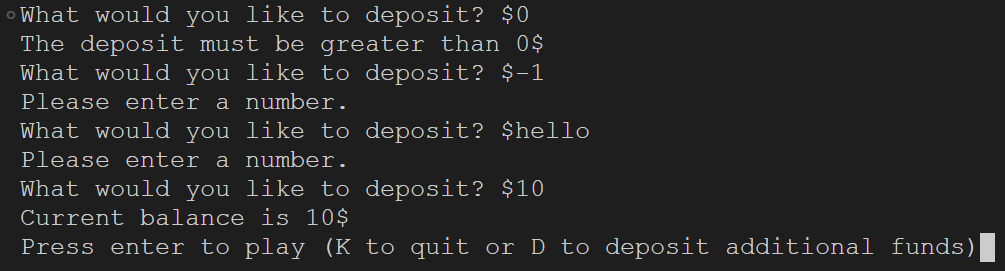
        else:

            print("Please enter a number.")

    return amount

With this function we utilize the “***while True:***” loop giving the user a possability to continuously provide an input to the console until all conditions are met.

Using the “***.isdigit***” we can check whether or not the customer provided a number for the deposit (1,10,100, etc) and that check is required for the system as the next action is to convert the amount the user provided to an integer using the “***amount = int(amount)***” command.

If all of the conditions are met the program will successfully break out of the loop and continue forward. If not the nested “***If/else***” statements will be in effect until the user meets the requirements:

**Step №3: Defining the number of lines and the bet amount.**

With the functions “***Def get\_number\_of\_lines ():”*** and “***def get\_bet ():***” we can gather information from the user regarding the number of lines they would like to play on and the amount for each line they would like to bet. Once the information is collected it will be printed out in the console as confirmation for the player to see.

def get\_number\_of\_lines():

    while True:

        Lines = input("Enter the number of lines to bet on: (1-"+str(MAX\_LINES) + ")? ")

        if Lines.isdigit():

            Lines = int(Lines)

            if 1 <= Lines <= MAX\_LINES:

                break

            else:

                print("Enter a valid number of lines")

        else:

            print("Please enter a number.")

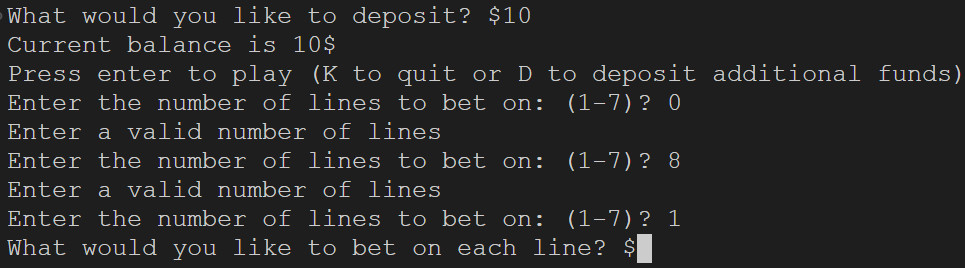
    return Lines

Once again we are implementing a loop for the customer to break out only if certain conditions are men in order for the slot machine to work properly and not produce errors when compiling.

That being said the when a customer cofnrims the deposit amount, the will be asked for the number of lines they would like to play on. This is done with a “***concatenation***” in order to print the range of lines available to the user for reference:

Lines = input("Enter the number of lines to bet on: (1-"+str(MAX\_LINES) + ")? ")

Resulting in the following being visible for the ***user*** if all conditions set in the function are met:



After successfully confirming the number of lines the user would like to bet on, they will receive a promt regarding the amount they would like to play on each line described in the “***Def get\_bet():”*** function:

def get\_bet():

    while True:

        amount = input("What would you like to bet on each line? $")

        if amount.isdigit():

            amount = int(amount

            if MIN\_BET <= amount <= MAX\_BET:

                break

            else:

                print(f"Amount must be between {MIN\_BET}$ and {MAX\_BET}$")

        else:

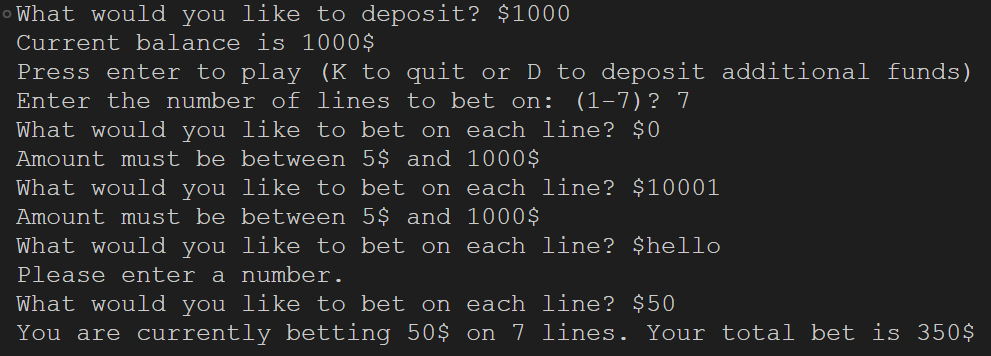
            print("Please enter a number.")

    return amount

In it, the we initate a loop that the user can break out of by inputting a number which falls between the “ ***MIN\_BET*** ” and “ ***MAX\_BET*** ” amounts confirmed in the beginnign. Instead of using concatenation here, we are using a “***formated string***” of an “***f-string***” to print out the range in which the bet amount must be:

print(f"Amount must be between {MIN\_BET}$ and {MAX\_BET}$")

When the program is running ***correctly*** the user will see the following:



**Step №4: Defining the gameplay of the machine.**

When the initial information regarding the deposita mount/lines played/bet is collected from the user, the program will move forward with defining the spin of the slot machine with the “***Def spin():***” function:

def spin(balance):

    lines = get\_number\_of\_lines()

    bet = get\_bet()

    total\_bet = bet \* lines

    if total\_bet > balance:

        print(f"You do not have enough to bet that amount. Your current balance is {balance}$")

        return 0

    print(f"You are currently betting {bet}$ on {lines} lines. Your total bet is {total\_bet}$")

    slots = get\_slot\_machine\_spin(ROWS, COLS, symbol\_count)

    print\_slot\_machine(slots)

    winnings, winning\_lines = check\_winnings(slots, lines, bet, symbol\_value)

    print(f"You won {winnings}$.")

    print(f"You won on line:", \*winning\_lines)

    return winnings - total\_bet

With the fragment above the program runs the slot machine and provides the user with a confirmation on whether they win or nor, and if they won – which is the winning line/lines. Additional balance check is included in order for the game to see if the user is trying to bet an amount bigger that their current balance. Including “**\***” before the variable “***winning\_lines***” instructs the program to pass every single line from the list “***winning\_lines***”. It is also know as the “*unpack/splat operator*”

**Step №5: Defining the actual spin of the slot machine.**

def get\_slot\_machine\_spin(rows, cols, symbols):

    all\_symbols = []

    for symbol, symbol\_count in symbols.items():

        for \_ in range(symbol\_count):

            all\_symbols.append(symbol)

    columns = []

    for \_ in range(cols):

        column = []

        current\_symbols = all\_symbols[:]

        for \_ in range(rows):

            value = random.choice(current\_symbols)

            current\_symbols.remove(value)

            column.append(value)

        columns.append(column)

    return columns

In the fragment above, we can see the actual spin of the slot machine being generated. By passing the parameters “***ROWS***” “***COLS***” “***SYMBOLS***” through the function “***Def get\_slot\_machine\_spin***():” we can program the machine to use the items to create a reel in order for the user to see if they won/lost.

We are using “***.items***” as it provides us with the opportunity to get the key alongside its value rather that writing a value for each key manually.

“***\_***” is an anonymous variable in Python that is used when the iteration value is not needed. This way of coding reduces the possibility of two variables having the same name, hence producing an error or incorrect results. Additionally“***.append***” is used to add the item to the end of the empty list.

When we generate the columns in the slot machine, we are using a **copy** of the list named “***all\_symbols***” as to not affect the original list making the program prone to errors. A copy of a list is created by adding “**[:]**”

Since the module “***random***” was imported in the start of the program we can now use the “***random.choice***” command in order to randomly select values from the list to be printed out on the “screen’ of the slot machine. With the “.***remove***” command once an item is chosen it cannot be chosen again.

Once the two loops are completed, we then append the colums to the screen.

**Step №6: Printing the screen of the slot machine.**

def print\_slot\_machine(columns):

    print('\n')

    for row in range(len(columns[0])):

        for i, column in enumerate(columns):

            if i != len(columns)-1:

                print(column[row], end="|")

            else:

                print(column[row], end="")

        print()

    print('\n')

In order for us to print the infomration the user will be seeing in the console, also know as transposing a matrix we are again using a nested for loop.

The “***enumerate***” command does not let the program print a “**|**” at the end of the last column by checking if the length of the printed matrix is the maximum available ***[I != len(columns)-1]***. Additionally by addint “(***end=“ ”)***” we instruct the program to print the screen on the correct line.

|  |  |
| --- | --- |
| Results before adding the empty end string. | Results after adding it. |
|  |  |

**Step №6: Checking the winning combinations and calculations.**

With the function “***Def check\_winnings (colums, lines, bet, values):***” we can instruct the program to check the active lines, for matching items and then determine what the user won/lost depending on the values of the symbols, whether or not the scater symbol is present, and the active bet for each line.

def check\_winnings(columns, lines, bet, values):

    winnings = 0

    winning\_lines = []

    for line in range(lines):

        if len(columns) > 0 and len(columns[0]) > line:

            symbol = columns[0][line]

        else:

            break

        for column in columns:

            symbol\_to\_check = column[line]

            if symbol != symbol\_to\_check:

                break

        else:

            if symbol in ["\*", "@"]:

              winnings += values[symbol] \* bet \* 2

            else:

              winnings += values[symbol] \* bet

            winning\_lines.append(line + 1)

    diagonal1 = [columns[i][i] for i in range(len(columns))]

    diagonal2 = [columns[i][len(columns)-1-i] for i in range(len(columns))]

    diagonals = [diagonal1, diagonal2]

    for diagonal in diagonals:

        symbol = diagonal[0]

        if all(symbol == d for d in diagonal):

            if symbol in ["\*", "@"]:

                winnings += values[symbol] \* bet \* 2

            else:

                winnings += values[symbol] \* bet

            winning\_lines.append("Diagonal")

    return winnings, winning\_lines

**Step №7: Executing the main() function for the complete game.**

In order for the slot machine to work, and results to be printed accurately in the console we call the function “***main ():***”

def main():

    balance = deposit()

    while True:

        print(f"Current balance is {balance}$")

        answer = input(

            "Press enter to play (K to quit or D to deposit additional funds)")

        if answer.lower() == "k":

            break

        elif answer.lower() == "d":

            additional\_deposit = deposit()

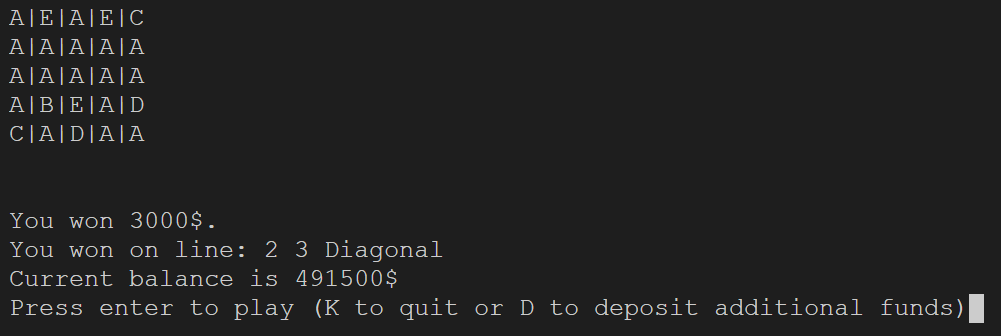
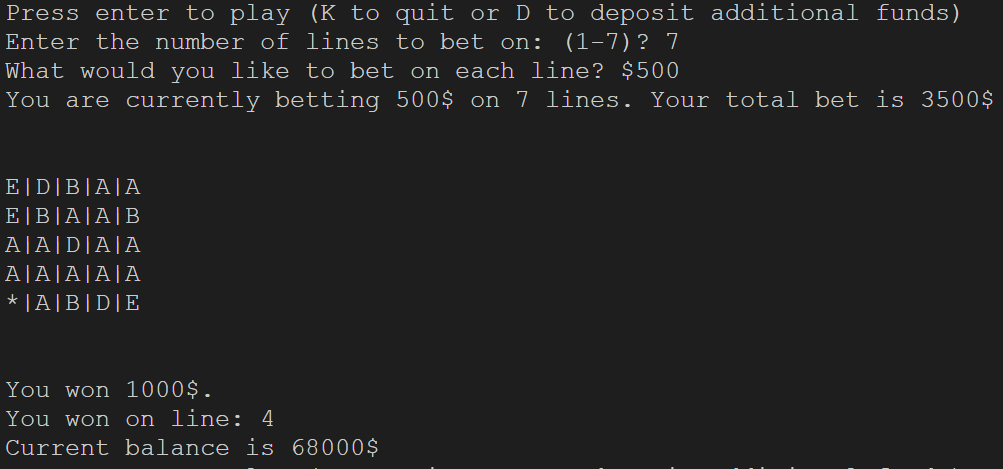
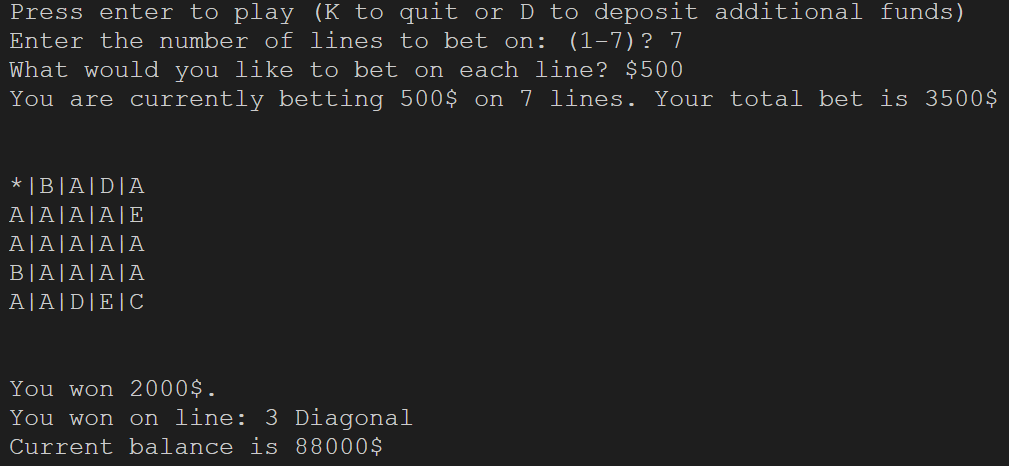
            balance += additional\_deposit

        else:

            balance += spin(balance)

    print(f"You left with {balance}")

In this function we implement a while loop that will be infinite, unless the user deliberately chooses to quit the game.

Examples of the current final version of the program showing the abovementioned code woriking in different ways: