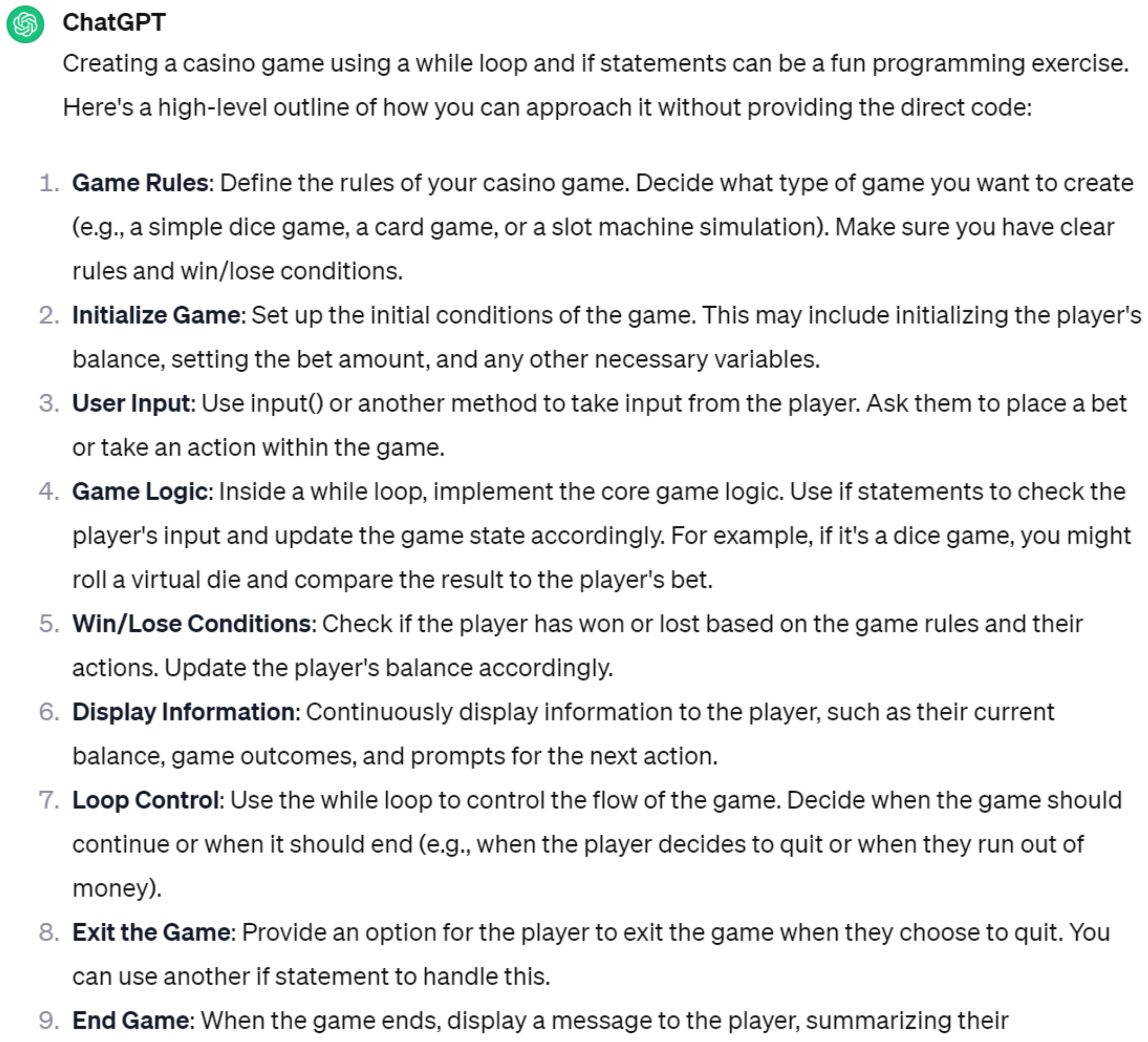
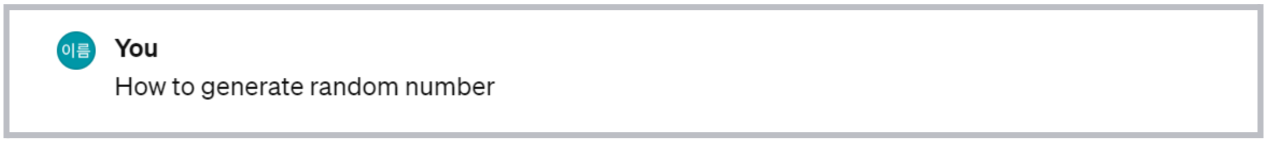


ChatGPT

Creating a casino game using a while loop and if statements can be a fun programming exercise. Here's a high-level outline of how you can approach it without providing the direct code:

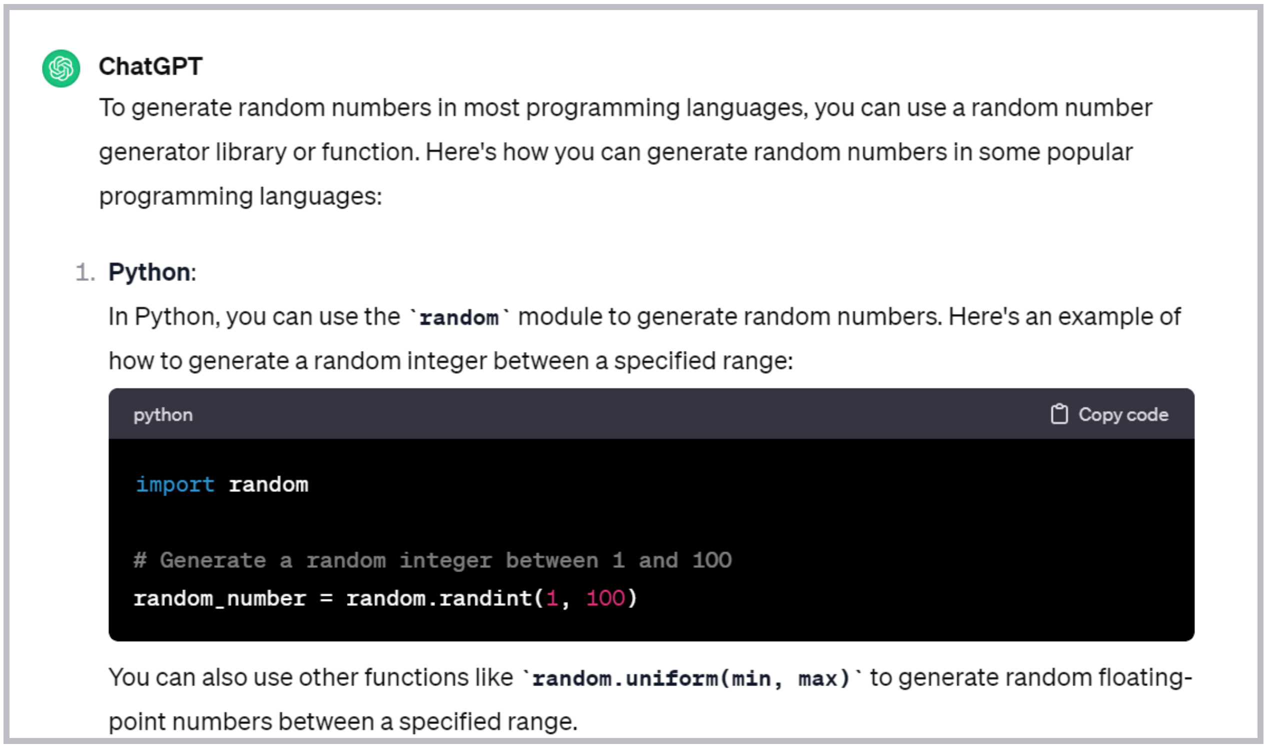
1. **Game Rules**: Define the rules of your casino game. Decide what type of game you want to create (e.g., a simple dice game, a card game, or a slot machine simulation). Make sure you have clear rules and win/lose conditions.
2. **Initialize Game**: Set up the initial conditions of the game. This may include initializing the player's balance, setting the bet amount, and any other necessary variables.
3. **User Input**: Use input() or another method to take input from the player. Ask them to place a bet or take an action within the game.
4. **Game Logic**: Inside a while loop, implement the core game logic. Use if statements to check the player's input and update the game state accordingly. For example, if it's a dice game, you might roll a virtual die and compare the result to the player's bet.
5. **Win/Lose Conditions**: Check if the player has won or lost based on the game rules and their actions. Update the player's balance accordingly.
6. **Display Information**: Continuously display information to the player, such as their current balance, game outcomes, and prompts for the next action.
7. **Loop Control**: Use the while loop to control the flow of the game. Decide when the game should continue or when it should end (e.g., when the player decides to quit or when they run out of money).
8. **Exit the Game**: Provide an option for the player to exit the game when they choose to quit. You can use another if statement to handle this.
9. **End Game**: When the game ends, display a message to the player, summarizing their

This passage reveals that when creating casino games, it's crucial to ensure fair randomness and use conditions to determine wins and losses. Additionally, the repetitiveness of the game is of utmost importance. Based on this understanding, the next steps will involve using conditional statements to decide wins and losses and starting to create a game that is repetitive in nature.



You

How to generate random number



ChatGPT

To generate random numbers in most programming languages, you can use a random number generator library or function. Here's how you can generate random numbers in some popular programming languages:

1. **Python:**

   In Python, you can use the `random` module to generate random numbers. Here's an example of how to generate a random integer between a specified range:

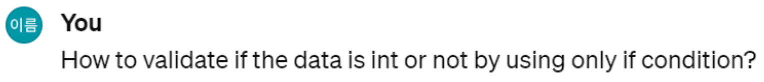
```python
import random

# Generate a random integer between 1 and 100
random_number = random.randint(1, 100)
```

   You can also use other functions like `random.uniform(min, max)` to generate random floating-point numbers between a specified range.

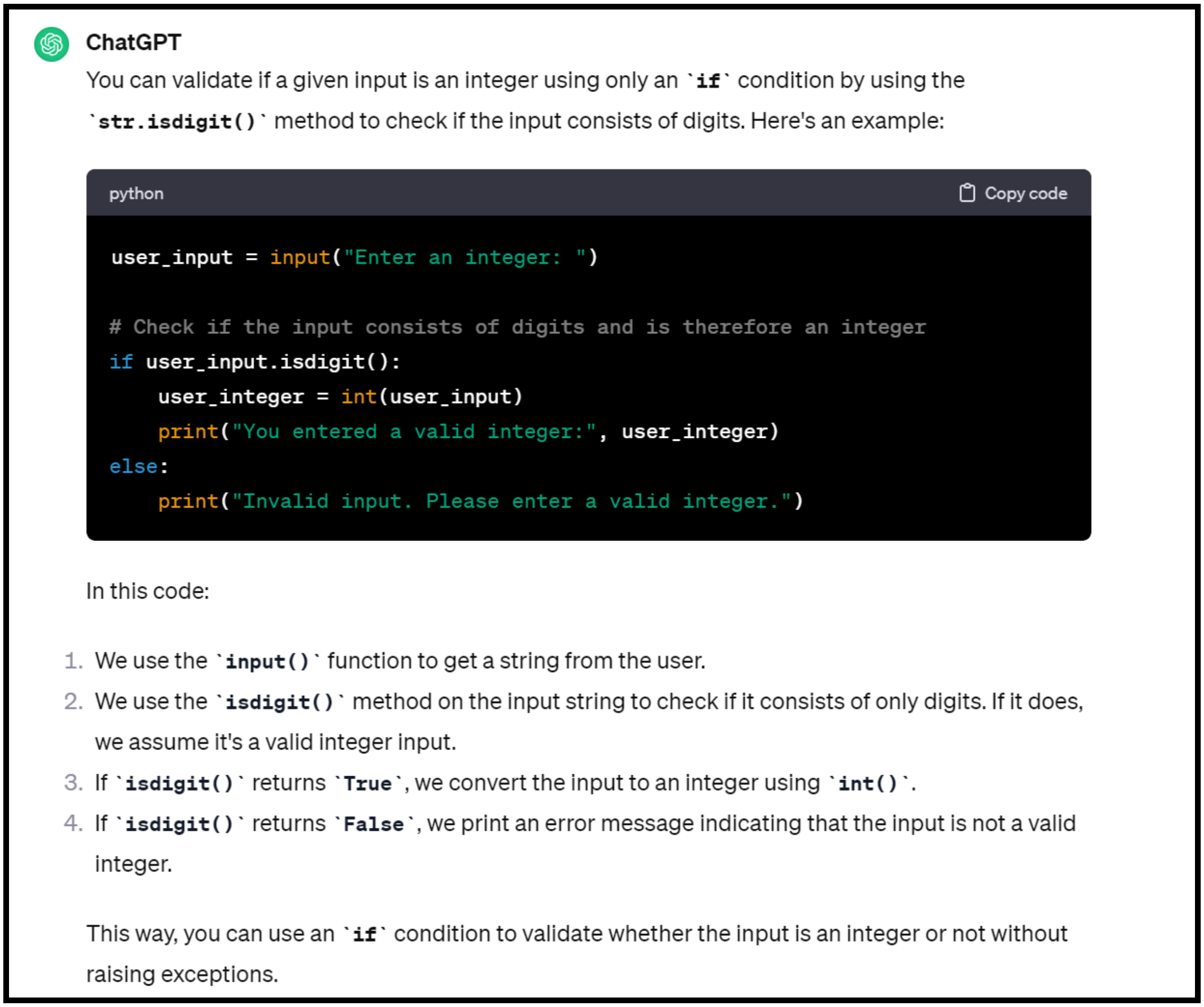
The first thing is the implementation of randomness.

I can use the 'random' module to generate random numbers within a specific range. Since the output of 'randint' is an integer, I can easily determine if it's odd or even



You

How to validate if the data is int or not by using only if condition?



ChatGPT

You can validate if a given input is an integer using only an `if` condition by using the `str.isdigit()` method to check if the input consists of digits. Here's an example:

```python
user_input = input("Enter an integer: ")

# Check if the input consists of digits and is therefore an integer
if user_input.isdigit():
    user_integer = int(user_input)
    print("You entered a valid integer:", user_integer)
else:
    print("Invalid input. Please enter a valid integer.")
```

In this code:

1. We use the `input()` function to get a string from the user.
2. We use the `isdigit()` method on the input string to check if it consists of only digits. If it does, we assume it's a valid integer input.
3. If `isdigit()` returns `True`, we convert the input to an integer using `int()`.
4. If `isdigit()` returns `False`, we print an error message indicating that the input is not a valid integer.

This way, you can use an `if` condition to validate whether the input is an integer or not without raising exceptions.

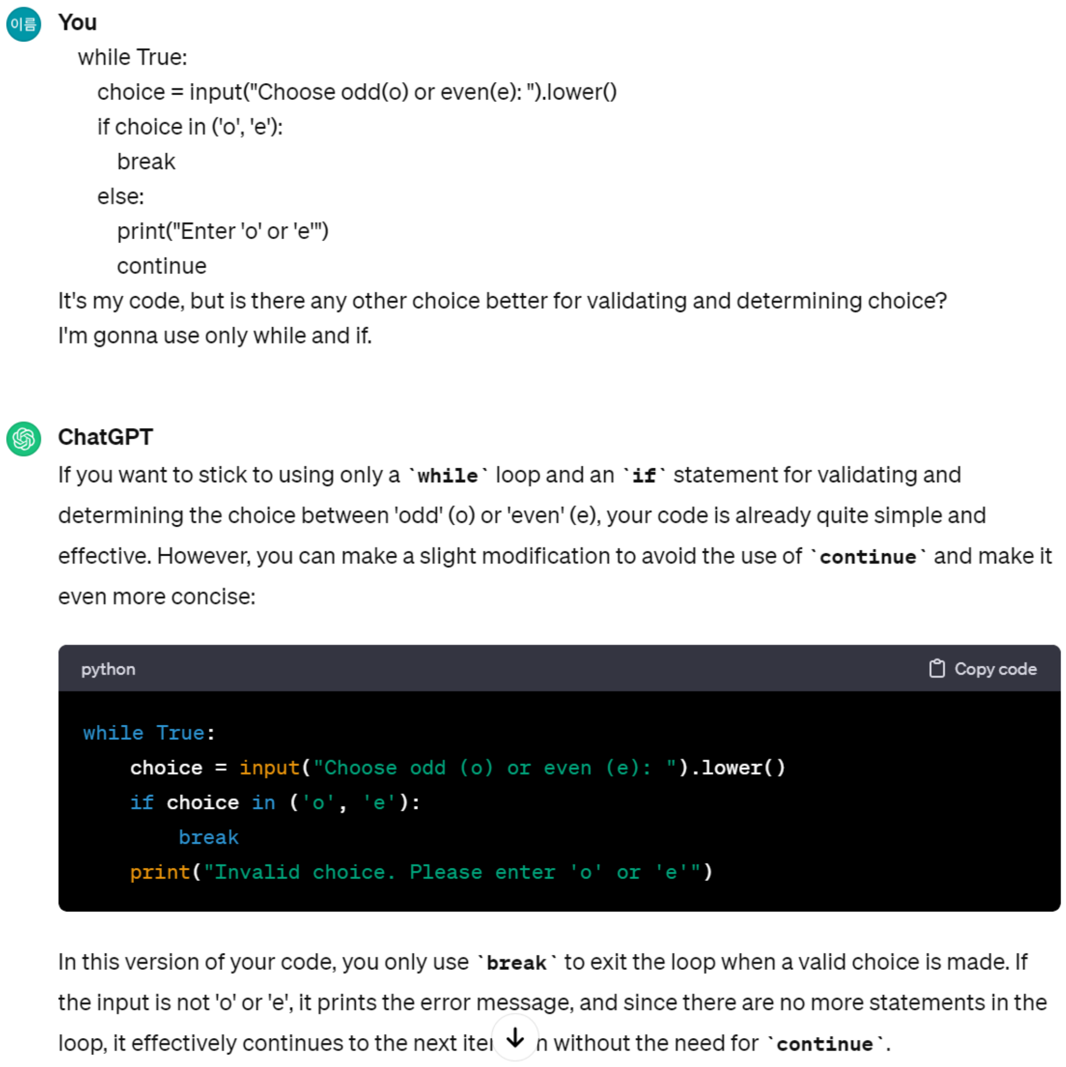
In the next step, when you get an input from a player, you should validate the input data. But I'm going to validate my input data not using try and except but using only if condition.

The first thing I have to do is determine if the input value is integer or not. And then, I should check if the integer exists in a proper range.

Here’s my validation code for an input.

```
while True:
    user_input = input("Choose your bet size. Minimum size is 100$: ")
    if user_input.isdigit():
        bet_size = int(user_input)

        if bet_size >= min_bet_size and bet_size <= player_balance:
            break
        else:
            print(f"Wrong Input. Please enter a bet between {min_bet_size} and {player_balance}.")
        continue
    else:
        print("Please enter an integer")
        continue
```



You

```
while True:
  choice = input("Choose odd(o) or even(e): ").lower()
  if choice in ('o', 'e'):
    break
  else:
    print("Enter 'o' or 'e'")
    continue
```

It's my code, but is there any other choice better for validating and determining choice?
I'm gonna use only while and if.

ChatGPT

If you want to stick to using only a `while` loop and an `if` statement for validating and determining the choice between 'odd' (o) or 'even' (e), your code is already quite simple and effective. However, you can make a slight modification to avoid the use of `continue` and make it even more concise:

```python
while True:
    choice = input("Choose odd (o) or even (e): ").lower()
    if choice in ('o', 'e'):
        break
    print("Invalid choice. Please enter 'o' or 'e'")
```

In this version of your code, you only use `break` to exit the loop when a valid choice is made. If the input is not 'o' or 'e', it prints the error message, and since there are no more statements in the loop, it effectively continues to the next ite n without the need for `continue`.

I inquired about a more efficient method for validating and determining choice in the last script I wrote and made modifications according to the answers provided.

Script

```python
import random


player_balance = 10000
dealer_balance = 10000


min_bet_size = 100


while player_balance > 0 and dealer_balance > 0:

   while True:
       try:
           bet_size = int(input("Choose your bet size. Minimum size is 100$"))
           if bet_size < min_bet_size or bet_size > player_balance:
               print(f"Wrong Input. Please enter a bet between {min_bet_size} and {player_balance}.")
               continue
           break
       except ValueError:
           print("Invalid input. Please enter a valid number")
```

```
while True:
    choice = input("Choose odd (o) or even (e): ").lower()
    if choice in ('o', 'e'):
        break
    print("Invalid choice. Please enter 'o' or 'e'")



    random_number = random.randint(1, 100)

    check_even = random_number % 2 == 0

    if (check_even and choice == 'e') or (not check_even and choice == 'o'):
        player_balance += bet_size
        dealer_balance -= bet_size * 2
        print("You win", bet_size, '$')
    else:
        player_balance -= bet_size
        dealer_balance += bet_size
        print("You lose", bet_size, '$')

if player_balance <= 0:
    print("You lose!")
```

```
else:

   print("You win!", player_balance)
```

Conclusion

In this time,I attempted a challenge to create a casino program using only a while loop and if statements. The most remarkable thing of this challenge was the validation of input values using if conditionals.

Typically, the validation of input values is done using try and except blocks, and if the input is invalid, it is handled within the except block. However, in this challenge, I used if statements to determine the validity of input values and handled exceptional cases in the else block instead of using try and except.

Additionally, I used a while True loop to handle exceptions and re-run the script when necessary. In this scenario, if the expected conditions were met, the program would proceed to the next step using a break statement. However, if conditions were not met, for example, if the input was invalid, the program would not proceed and would instead revalidate using continue.

Through this series of challenges, I got a chance to understand more deeply while loops and if statements.