Binary Search Game

A Python-based game where the player seeks to determine the position of an element in a list, similar in fashion to the Binary Search algorithm.

Before we do anything, we have to include the `random` library that Python comes packaged with by default.

```
import random
```

Naturally, we want to start by defining a function that implements the algorithm:

```
def binary_search(array, n):
    low = array[0]
    high = len(array) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if array[mid] < n: low = mid + 1
        elif array[mid] > n: high = mid - 1
        else: return mid
    return -1
```

It works by defining a clear start, middle, and end, and works similarly to the way an individual might look up words in a dictionary. It checks the middle of the list to get an idea of it's whereabouts, and deduces wether it should go back a bit or if it should advance forward in the list. It repeats this process until it finds the desired number.

The goal of the game is to teach the player to think like a Binary Search algorithm.
To make it into an actual game, we create a win/lose state dictated by an arbitrary amount of choices.

Afterwards, we build the sorted list:

```
game_list = [ ]
game_list_length = random.randrange(8, 32, 4)
for i in range(game_list_length):
    n = random.randint(1, 256)
    while n in game_list:
        n = random.randint(1, 64)
    game_list.append(n)
game_list.sort()
```

The length of the list is can be anywhere between 8-32, and must be divisible by 4.
The `while loop` ensures that numbers already added to the list don't get repeated.

Now that we have a sorted list of randomly generated numbers, we can decide which one the player should search for. Remember, they must find the position of the element, and that is determined by calling the `binary_search()` function.

```
find_number = random.choice(game_list)
find_number_position = binary_search(game_list, find_number)
```

At this point, we've initialized all the data needed to start playing, and we can reveal the game's instructions to the end user:

```
print("A sorted list of " + str(game_list_length) + " elements has been generated.")
print("Use a \"Binary Search\" approach to find the position of number " + str(find_number) + ".")
print("You have " + str(choices) + " choices in total. Running out of choices means you lose the
game.")
print("\nGood hunting, Stalker.")
```

Now, we only have the gameplay loop left:

```
def end_game(state: False):
    if state: print("You win!")
    else: print("You lose, as expected.")
    exit()

while choices > 0:
    print()
    choices -= 1
    user_picked_index = int(input("Pick a position: "))
    print()
    print("Number at position " + str(user_picked_index) + " is " +
str(game_list[user_picked_index]) + ".")
    if user_picked_index is find_number_position:
        end_game(True)
    if choices > 1: print("You have " + str(choices) + " choices left.")
    elif choices == 1: print(" - C a r e f u l   n o w . . .")
    else: end_game(False)
```

Once the player runs out of choices, the game ends and mocks the player.
However, if the player manages to find the correct position, they are congratulated.
After which, the game closes.