PyArcade: Guess & Clash

A Python Console Project

Project Overview

Level 1: Logic

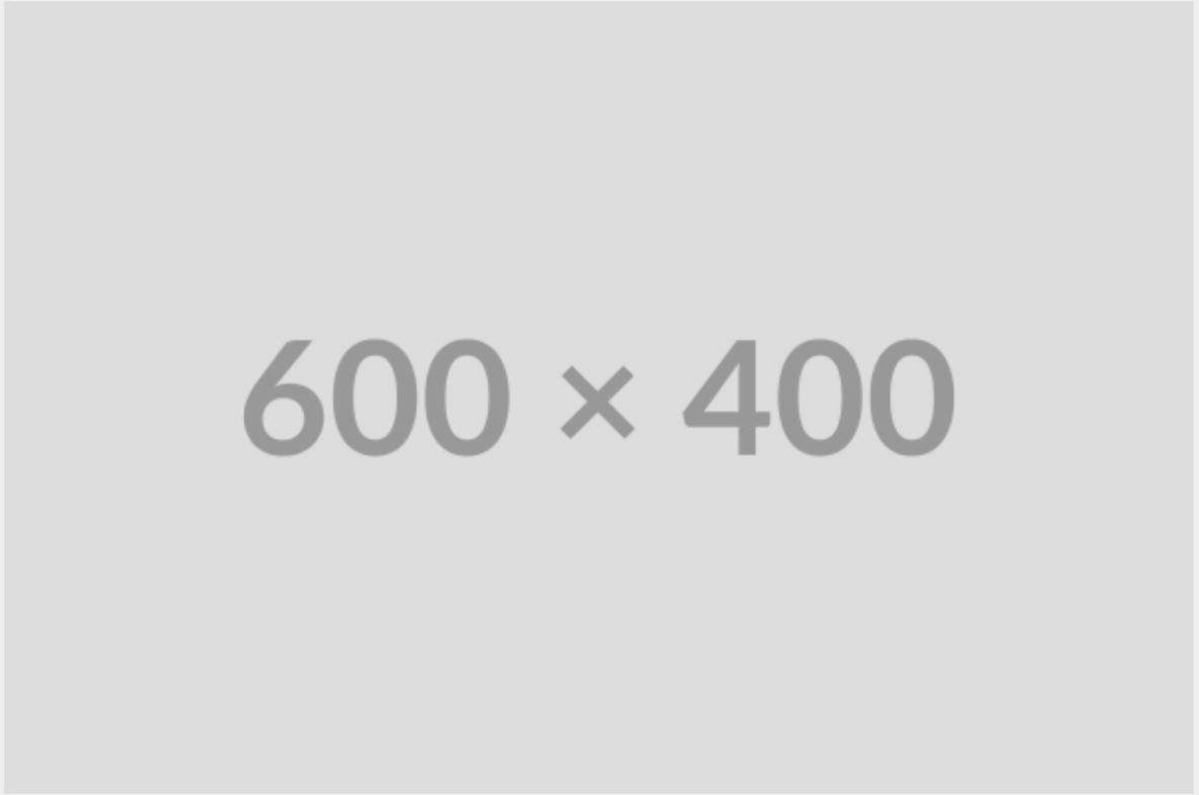
A classic "Guess the Number" challenge. The user must find a hidden integer between 10 and 100 using logical deduction and real-time feedback.

Level 2: Chance

An unlocked "Rock, Paper, Scissors" duel against the computer. This level introduces randomness and continuous gameplay loops.

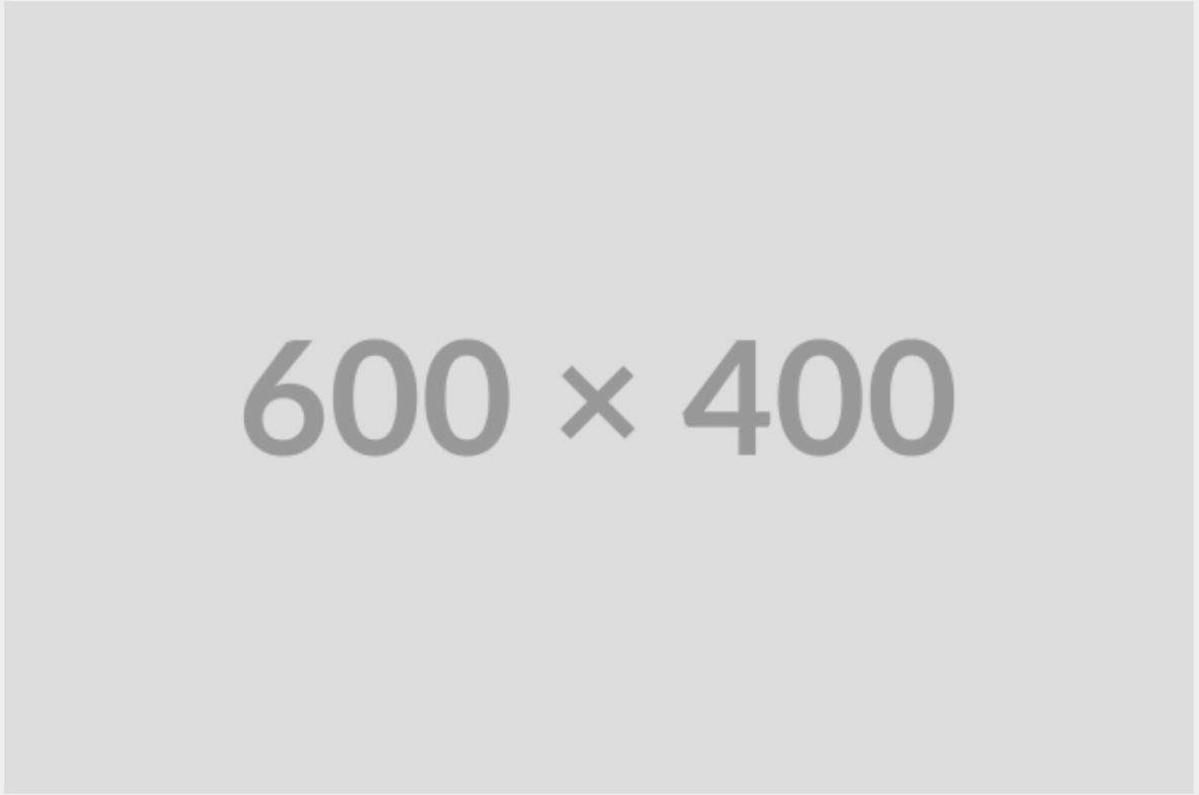
Level 1: The Number Challenge

- **Objective:** Identify a randomly generated number (10-100).
- **Feedback System:** The program guides the user with "Too High" or "Too Low" alerts after every guess.
- **Input Validation:** Prevents crashes by checking if inputs are valid digits before processing.



Level 2: The Duel

- **Classic Rules:** Rock crushes Scissors, Scissors cuts Paper, Paper covers Rock.
- **The Opponent:** The computer uses the `random` module to make unpredictable choices.
- **Game Loop:** Unlike Level 1, this stage allows for endless replayability until the user decides to quit.



Key Functions Used

I/O Operations

`input()` captures user decisions, while `print()` delivers instructions and game results.

Random Module

`random.randint()` generates the secret number, and `random.choice()` selects the computer's weapon.

Data Handling

`int()` converts strings for math comparison, and `lower()` normalizes text for case-insensitive matching.

Control Flow Logic

Conditional Logic

The "Brain" of the game. `if/elif/else` structures determine winners, validate ranges, and check win conditions.

Game Loops

`while` loops keep the game running. They persist until a specific condition (like a correct guess or "quit" command) is met.

Algorithm: Guessing Game

Step 1: Initialize

Import random module. Set range (10-100). Generate secret number. Reset attempt counter.

Step 2: Loop

Accept user input. Validate if it is a digit. If invalid, prompt again without crashing.

Step 3: Compare

Compare Guess vs. Secret. specific feedback (High/Low). If Match: End Loop & Unlock Level 2.

Algorithm: Rock Paper Scissors

Step 1: Selection

Computer picks a random option from the list. User inputs their choice (normalized to lowercase).

Step 2: Judgment

Check for Tie. If not tie, apply game rules (e.g., Rock > Scissors) to declare a Winner or Loser.

Step 3: Recursion

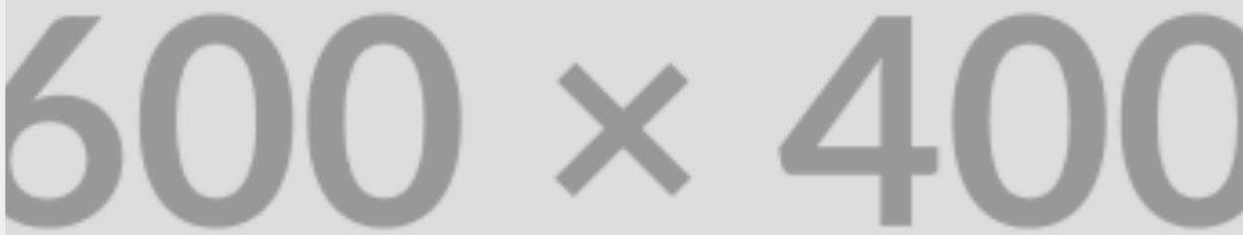
Ask user to play again. If 'yes', restart loop. If 'no', break loop and terminate program.

Code Structure

Modular & Clean

The code is structured sequentially. Level 1 acts as a gatekeeper; Level 2 code is nested within the success condition of the first game.

This ensures a linear progression system, rewarding the player for their logic skills before offering the chance-based game.



Source Code: Level 1

```
import random
lowest_num = 10
highest_num = 100
number = random.randint(lowest_num, highest_num)
guesses = 0
is_running = True

print("CAN YOU GUESS THE NUMBER?")
while is_running:
    guess = input("enter your guess:")
    if guess.isdigit():
        guess = int(guess)
        guesses += 1
        if guess < lowest_num or guess > highest_num:
            print("Out of range!")
        elif guess < number:
            print("Too low!")
        elif guess > number:
            print("Too high!")
        else:
            print(f"CORRECT! Number was {number}")
            is_running = False # Win condition met
```

Source Code: Level 2

```
# ... Inside the win condition of Level 1 ...
choice = input("Play Level 2? (yes/no):").lower()
if choice == "yes":
    options = ("rock", "paper", "scissors")
    running = True
    while running:
        computer = random.choice(options)
        player = input("rock, paper, scissors?").lower()
        if player == computer:
            print("It's a tie")
        elif (player == "rock" and computer == "scissors") ...:
             print("YOU WON!!!")
        else:
             print("YOU LOST!!!")

        if input("Play again? (y/n)") == "n":
            running = False
```

Output Scenarios

Success Path

User guesses 62 (Correct) -> "Total guesses: 3" -> "Go to Level 2?" -> User enters "Yes" -> RPS Game Starts.

Error Handling

User enters "Hello" -> System prints "Invalid Input" -> User enters 200 -> System prints "Out of Range".

Game Over!

This project demonstrates how basic Python logic can build engaging, multi-stage interactive applications.