# Platformer Coin Collector Challenge

## Challenge Preamble

This challenge is designed to help you practice basic Python concepts such as variables, loops, conditional statements, and file handling. You will develop a program that simulates a 2D platformer-style level where the player moves through a grid, collecting coins (C), avoiding walls (#), and following movement instructions from a text file.

Your task is to read a grid file (`level.txt`) and a movement file (`moves.txt`), interpret each move, and update the player's position accordingly. The program will count how many coins are collected and display the total at the end of the run.

## Part 1 – Level Parsing and Movement: Theory

In this program, we will read a map file representing a 2D level. Each character in the grid represents a tile:
- `'P'` is the player's starting position
- `'C'` is a coin to collect
- `'#'` is a wall that blocks movement
- `'.'` is empty space the player can walk on

We will also read a movement file containing single-letter commands per line (`U, D, L, R`). Each move updates the player's position in the grid unless blocked by a wall. If the player steps on a coin, it is collected and replaced by `'.'`.

Key Concepts:
• Reading files using the open() function with a context manager.
• Converting strings into mutable lists of characters for grid manipulation.
• Using loops and match-case statements to handle directional logic.
• Tracking positions with `(x, y)` coordinates.
• Counting coins as they are collected.

## Part 2 – Code Walk-through

### Function: `find_start()`

This function scans the grid to find the coordinates of the player's starting position marked by 'P'. It uses a nested loop: one loop for rows (y) and another for tiles within each row (x). When 'P' is found, the function returns its (x, y) coordinates.

### Function: `parse_level()`

The parse_level function converts the text-based level file into a mutable 2D list of characters. Each line from the file is stripped of newline characters and split into a list of characters using list(). Empty lines are ignored to ensure the grid stays consistent.

### Function: `main()`

The main() function controls the overall flow of the program. It opens both files (**`level.txt`** and **`moves.txt`**), parses them, locates the starting position, and then simulates each movement command.

Step-by-step explanation:

1. Read the level file using **`open()`** and **`readlines()`**, then call **`parse_level()`** to convert it into a grid.
2. Read the moves file in the same way, storing all movement commands.
3. Use **`find_start()`** to locate the player's starting coordinates and store them in pos.
4. Replace **`'P'`** in the grid with **`'.'`** since we only need the coordinates from now on.
5. For each move in moves.txt, strip whitespace and interpret it using match-case:
   - **`'R'`** → move right (+1 x)
   - **`'L'`** → move left (-1 x)
   - **`'U'`** → move up (-1 y)
   - **`'D'`** → move down (+1 y)
6. After determining the new position, check what tile exists there:
   - **`'.'`** means it's empty, move there.
   - **`'C'`** means collect a coin, increment coins, and replace the tile with **`'.'`**.
   - **`'#'`** means a wall—stay in place.

7. Continue processing all moves, and at the end, print the total coins collected. If you succeed you would collect **29 coins**.

## Part 3 – Bonus Challenge: TRON Trail Mode

In this bonus variation, the player leaves behind a solid wall (`'#'`) after every step. This simulates a TRON-like light trail: once the player moves over a tile, it becomes impassable for the rest of the game. This rule drastically changes the number of reachable coins depending on the order of moves.

To implement this mode, simply modify the main loop to add:

```
grid[pos[1]][pos[0]] = '#'
```

just before updating the player's position.

Challenge Question: Run your program once in normal mode and once in TRON mode. How many coins can you collect in each? What difference do you observe?

If you succeed you would collect only **7 coins**.