**Assignment 2: Genetic Algorithm for Automated Test Case Generation Course:** Applied Artifical Intelligence, Spring 2025

**Due Date:** 15th March 2025

# Objective

Design and implement a genetic algorithm (GA) to automatically generate test cases for a **Date Validation Function**. The goal is to maximize test coverage by generating inputs that validate all possible valid and invalid date scenarios, including edge cases and boundary values.

# Problem Statement

A date validation function accepts a string in the format DD/MM/YYYY and returns:

1. **Valid**: If the date is a real calendar date (e.g., 15/05/2023).
2. **Invalid**: If the date is impossible (e.g., 31/04/2023, 29/02/2021). Your task is to evolve test cases (date strings) that cover:
   * All valid date categories (e.g., leap years, month/day boundaries).
   * All invalid categories (e.g., month > 12, day > 31, incorrect leap years).
   * Boundary values (e.g., 31/12/9999, 01/01/0000).

# Requirements

1. **Genetic Algorithm Components**
   * **Chromosome Representation**: Each test case is a date string represented as a tuple (day, month, year).
   * **Population Initialization**: Randomly generate dates with day ∈ [1-31], month ∈

[1-12], year ∈ [0000-9999].

# Fitness Function:

* + - Reward test cases that cover new equivalence classes (e.g., leap year, 30- day months).
    - Penalize redundant test cases (e.g., multiple dates in the same category).
    - Formula:
  + **Selection**: Use **rank-based selection** to prioritize high-coverage test cases.
  + **Crossover**: Swap day/month/year segments between parents (e.g., child inherits day from parent 1 and month/year from parent 2).
  + **Mutation**: Perturb a value (e.g., change day ±3, month ±1, year ±100) with 15% probability.
  + **Termination**: Stop after 100 generations or when 95% coverage is achieved.

# Constraints

* + Generate at least **10 valid** and **10 invalid** test cases.
  + Include **5 boundary cases** (e.g., 31/12/9999, 29/02/2020).
  + Dates must be strings in DD/MM/YYYY format.

# Output

Print the best-evolved test cases, their categories, and coverage percentage.

# Example Problem Instance Target Categories:

* Valid: Leap year, 30-day month, 31-day month.
* Invalid: Day > 31, Month > 12, Non-leap February 29.
* Boundaries: Min/Max year, day/month transitions.

**Sample Solution**:



**Deliverables**

1. **Source Code**
   * Python implementation with functions for GA operations and fitness calculation.

# Report

* + Explanation of fitness function design and chromosome representation.
  + Analysis of parameter tuning (e.g., mutation rate impact).
  + Coverage results for valid, invalid, and boundary categories.
  + Comparison of GA efficiency vs. random testing.
  + Include a **line graph** in your report to visualize how the GA’s coverage improves over generations.

# Test Cases

* + Submit a CSV/JSON file with the best-evolved test cases and their categories.

|  |  |  |
| --- | --- | --- |
| **Grading Rubric** | **Criteria** | **Weight** |
|  | Coverage (Valid/Invalid/Boundary) | 30% |
|  | Fitness Function Design | 25% |
|  | GA Implementation Correctness | 20% |
|  | Report Ǫuality C Graph Completeness | 15% |
|  | Code Readability C Comments | 10% |

# Bonus (10% Extra Credit)

Integrate a **local search** algorithm to refine test cases post-GA execution. Compare coverage results with the baseline GA.

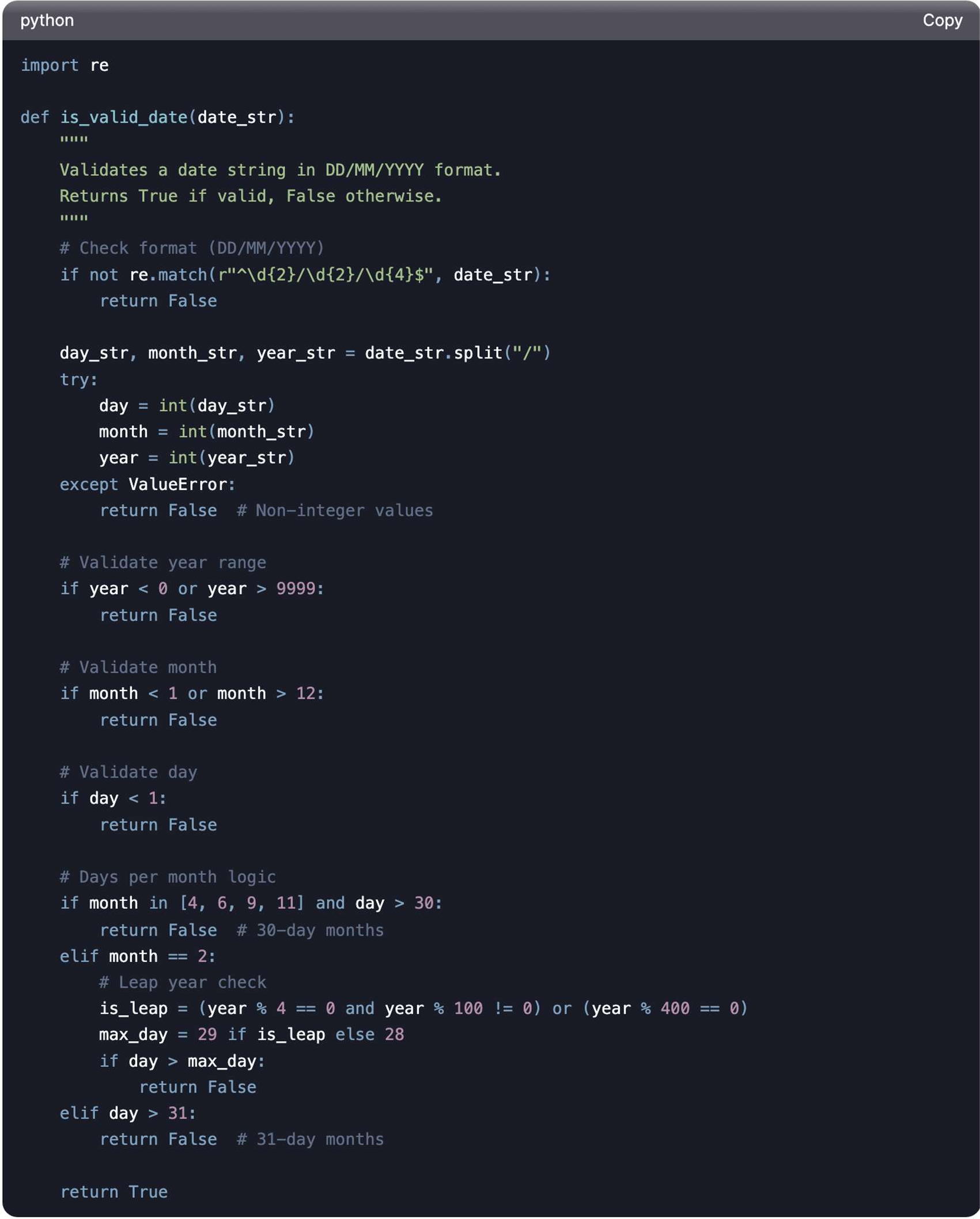
# Submission

Compress code, report, and test cases into ROLLNO\_NAME.zip.

**Note**: You may use libraries like random or numpy, but pre-built test case generation tools and code generations from AI assistants are prohibited.

# Example Output:

Here’s a **date validation function** to test the GA-generated test cases. This function checks if a date string in DD/MM/YYYY format is valid or invalid, adhering to the rules specified (including leap years and boundaries):



# Example Usage

****

**Additional Sample Problem Example Instances**:

Here are **three additional sample problem instances** with varying complexity and constraints to test the GA’s adaptability:

# Sample Problem Instance 1: Basic Date Validation Description:

Generate test cases for a date validator with basic checks (no leap year logic).

# Target Categories:

* **Valid**: Dates with day ≤ 30 for April, June, September, November.

# Invalid:

* + day > 31 for any month.
  + month > 12.
  + day > 30 for April, June, September, November.

# Constraints:

* At least 5 valid and 5 invalid test cases.
* Include the boundary case 31/01/2023 (valid 31-day month).

# Sample Solution:

**Coverage**: 80%

# Sample Problem Instance 2: Advanced Leap Year G Boundaries Description:

Test a date validator with strict leap year rules (Gregorian calendar).

# Target Categories:

* **Valid**:
  + Leap years: 29/02/2020 (divisible by 400 or 4 but not 100).
  + Non-leap years: 28/02/1900 (divisible by 100 but not 400).

# Invalid:

* + 29/02/2021 (non-leap year).
  + 29/02/1900 (divisible by 100 but not 400).

# Boundaries:

* + Minimum year: 01/01/0000.
  + Maximum year: 31/12/9999.

# Constraints:

* Include at least 3 valid leap year and 3 invalid leap year test cases.
* Penalize solutions missing 01/01/0000 or 31/12/9999.

# Sample Solution:

**Coverage**: 90%

# Sample Problem Instance 3: Complex Month-Day Combinations Description:

Test all possible invalid day-month combinations, including edge cases like February 30th.

# Target Categories:

* **Invalid**:

o 30/02/2023 (February 30th).

o 31/04/2023 (April 31st).

o 31/06/2023 (June 31st).

o 31/09/2023 (September 31st).

o 31/11/2023 (November 31st).

# Constraints:

* Generate test cases for **all 12 months** with invalid days.
* Prioritize diversity: Penalize duplicate invalid categories.

# Sample Solution:

**Coverage**: 95%

# Sample Problem Instance 4: Format Variations Description:

Test a date validator that accepts multiple formats: DD/MM/YYYY, MM/DD/YYYY, YYYY/MM/DD.

# Target Categories:

* **Valid**:

o 15/05/2023 (DD/MM).

o 05/15/2023 (MM/DD).

o 2023/05/15 (YYYY/MM/DD).

# Invalid:

* + Ambiguous dates (e.g., 05/06/2023 in both DD/MM and MM/DD).

# Constraints:

* Generate test cases for all three formats.
* Ensure no ambiguity in valid test cases.

# Sample Solution:

**Coverage**: 85%

# Notes for Students

1. Use these instances to test your GA’s ability to:
   * Prioritize diverse test cases.
   * Handle edge cases and boundary values.
   * Adapt to varying fitness function penalties.
2. Experiment with mutation rates (e.g., higher rates for boundary years) and crossover strategies (e.g., swapping month/day segments).
3. For ambiguous formats (Instance 4), ensure your GA explicitly labels the format used in each test case.

# Example Output for Instance 2:

You can use these examples to refine your GA’s fitness function and mutation logic!

**Good luck!** ,•˙Q