Khan_MuhammadSharjeel_21885084_ISErepo

- **Student Name:** Muhammad Sharjeel Khan
- **Student ID:** 21885084
- **Date:** 26-June-2025

Introduction

This project implements a numerology analysis tool that processes birthday input and performs the following tasks:

Scenario A

- Calculate the Life Path Number by adding digits (with recursive reduction), preserving master numbers (11, 22, 33)
- Determine the Lucky Colour corresponding to the Life Path Number
- Compare Life Path Numbers when two birthdays are provided

Scenario B

- Identify the generation (Silent Generation, Baby Boomers, Generation X, Millennials, Generation Z, or Generation Alpha) based on the birth year

Only birthdays between 1925 and 2025 are valid. The design adheres to modularity principles and is thoroughly tested using both black-box and white-box approaches. All work is tracked using version control (Git).

Detailed Description

Module Descriptions

1. birthday_checker

- **Name:** birthday_checker
- **Task:** Validate a birthday string, split it into day, month, and year, and convert these values
- **Input:** Birthday string (formats such as "13 November 1987", "13 Nov 1987", or "13 11 1987")
- **Output:** A tuple (day, month, year) if valid; otherwise, a ValueError is raised
- **Assumptions:** Only dates between 1925 and 2025 are accepted

2. life_path_calculator

- **Name:** life_path_calculator
- **Task:** Calculate the Life Path Number by summing the digits of the day, month, and year. The sum is recursively reduced until a single digit is obtained (except when a master number is encountered)
- **Input:** Three integers (day, month, year)
- **Output:** An integer representing the Life Path Number

3. color_finder

- **Name:** color_finder
- **Task:** Map a given Life Path Number to its corresponding Lucky Colour
- **Input:** Life Path Number (an integer)
- **Output:** A string representing the Lucky Colour (e.g., 5 → "Sky Blue")

4. generation_finder

- **Name:** generation_finder
- **Task:** Determine the generation to which a person belongs based on their birth year
- **Input:** Year (integer)
- **Output:** A string indicating the generation (e.g., "Silent Generation", "Baby Boomers")

5. main

- **Name:** main
- **Task:**
    - Act as the entry point for the program
    - Obtain birthday input from the user via keyboard
    - Call the modules for validating the birthday, calculating the Life Path Number, determining the Lucky Colour, and checking the generation
    - Display the analysis results
- **Input:** Birthday string from keyboard
- **Output:** Printed results on the console

Modularity

1. Design Principles and Decisions

- **Single Responsibility:** Each module is responsible for one distinct functionality
- **High Cohesion and Low Coupling:** Modules are designed to work independently and interact through well-defined interfaces. For example, birthday_checker returns a date tuple that is then consumed by life_path_calculator
- **Information Hiding:** Implementation details (e.g., the recursive digit reduction) are encapsulated within the module
- **Reusability and Extensibility:** The modules are designed to be reusable and can be easily extended to add further numerology features

2. Running the Production Code

Environment: Linux command-line with Python 3

Steps:

1. Open the terminal in the project root
    a. Execute the following command:

    python code/main.py

Sample Output:

Enter your birthday (e.g., 09 July 2005 or 13 Nov 1987): 13 November 1987

Your birthday : 13 November 1987
Your life path number : 3
Your lucky color : Yellow
Your generation : Generation X

Do you want to compare with another birthday? (yes/y, otherwise press enter): y
Enter second birthday (e.g., 09 July 2005 or 13 Nov 1987): 15 August 1990

Second birthday : 15 August 1990
Life path number : 6
Lucky color : Indigo
Generation : Millennials

```
==================================================
                   COMPARISON
==================================================
        Person 1         |         Person 2
--------------------------------------------------
Birthday: 13 November 1987 | Birthday: 15 August 1990
Life Path Number: 3      | Life Path Number: 6
Lucky Color: Yellow      | Lucky Color: Indigo
Generation: Generation X | Generation: Millennials
--------------------------------------------------
      Different Life Path Numbers: 3 vs 6
==================================================
```

3. Review Checklist and Refactoring Decisions

Review Checklist:

- **Single Responsibility:** Each module handles one specific task
- **Interface Consistency:** All modules use uniform input and output types
- **Error Handling:** Each module raises clear exceptions for invalid input
- **Naming Conventions:** All modules and variables have descriptive names
- **Elimination of Code Duplication:** Common logic is abstracted into helper functions

Refactoring Decisions:

- *birthday_checker* was simplified by using clear month mappings and leap year logic
- *life_path_calculator* had its helper functions organized for better readability
- *main* was structured for clear user interaction flow and comparison display

Test Design

Test cases have been designed using both black-box (functional) and white-box (structural) approaches. Detailed tables for each module follow.

Black-Box Test Cases

1. Module: birthday_checker

Test ID	Description	Input	Expected Output	Test Approach
BC-EP-1	Valid birthday with full month name	"13 November 1987"	(13, 11, 1987)	Equivalence Partitioning
BC-EP-2	Valid birthday with month abbreviation	"13 Nov 1987"	(13, 11, 1987)	Equivalence Partitioning
BC-EP-3	Valid birthday with numeric month	"13 11 1987"	(13, 11, 1987)	Equivalence Partitioning
BC-EP-4	Invalid format (missing components)	"13 1987"	Raises ValueError	Equivalence Partitioning
BC-EP-5	Invalid day (non-numeric)	"AA November 1987"	Raises ValueError	Equivalence Partitioning
BC-EP-6	Invalid month (non-existent)	"13 Novem 1987"	Raises ValueError	Equivalence Partitioning
BC-EP-7	Invalid year (non-numeric)	"13 November XXXX"	Raises ValueError	Equivalence Partitioning
BC-EP-8	Day out of range for month	"31 February 2000"	Raises ValueError	Equivalence Partitioning
BC-EP-9	Year out of valid range	"13 November 2026"	Raises ValueError	Equivalence Partitioning
BC-BVA-1	Boundary: Earliest valid year	"01 January 1925"	(1, 1, 1925)	Boundary Value Analysis

BC-BVA-2	Boundary: Latest valid year	"31 December 2025"	(31, 12, 2025)	Boundary Value Analysis
BC-BVA-3	Just below earliest valid year	"31 December 1924"	Raises ValueError	Boundary Value Analysis
BC-BVA-4	Just above latest valid year	"01 January 2026"	Raises ValueError	Boundary Value Analysis

2. Module: life_path_calculator

Test ID	**Description**	**Input (day, month, year)**	**Expected Output**	**Test Approach**
LPC-EP-1	Basic calculation with single-digit result	(1, 1, 2000)	4	Equivalence Partitioning
LPC-EP-2	Calculation requiring digit reduction	(29, 8, 1994)	6	Equivalence Partitioning
LPC-EP-3	Calculation resulting in a master number	(29, 2, 1980)	22	Equivalence Partitioning
LPC-EP-4	Input includes a master number	(11, 3, 1986)	2	Equivalence Partitioning

3. Module: color_finder

Test ID	**Description**	**Input (Life Path Number)**	**Expected Output**	**Test Approach**
CF-EP-1	Mapping for a regular number	5	"Sky Blue"	Equivalence Partitioning
CF-EP-2	Mapping for master number 11	11	"Silver"	Equivalence Partitioning
CF-EP-3	Mapping for master number 22	22	"White"	Equivalence Partitioning

CF-EP-4	Mapping for master number 33	33	"Crimson"	Equivalence Partitioning

4. Module: generation_finder

Test ID	Description	Input (Year)	Expected Output	Test Approach
GF-EP-1	Silent Generation	1940	"Silent Generation"	Equivalence Partitioning
GF-EP-2	Baby Boomers	1960	"Baby Boomers"	Equivalence Partitioning
GF-EP-3	Generation X	1970	"Generation X"	Equivalence Partitioning
GF-EP-4	Millennials	1990	"Millennials"	Equivalence Partitioning
GF-EP-5	Generation Z	2000	"Generation Z"	Equivalence Partitioning
GF-EP-6	Generation Alpha	2015	"Generation Alpha"	Equivalence Partitioning
GF-EP-7	Year outside valid range	1900	"Unknown"	Equivalence Partitioning
GF-BVA-1	Boundary: Start of Silent Generation	1901	"Silent Generation"	Boundary Value Analysis
GF-BVA-2	Boundary: End of Silent Generation	1945	"Silent Generation"	Boundary Value Analysis
GF-BVA-3	Boundary: Start of Baby Boomers	1946	"Baby Boomers"	Boundary Value Analysis
GF-BVA-4	Boundary: End of Baby Boomers	1964	"Baby Boomers"	Boundary Value Analysis

GF-BV A-5	Boundary: Start of Generation X	1965	"Generation X"	Boundary Value Analysis
GF-BV A-6	Boundary: End of Generation X	1979	"Generation X"	Boundary Value Analysis
GF-BV A-7	Boundary: Start of Millennials	1980	"Millennials"	Boundary Value Analysis
GF-BV A-8	Boundary: End of Millennials	1994	"Millennials"	Boundary Value Analysis
GF-BV A-9	Boundary: Start of Generation Z	1995	"Generation Z"	Boundary Value Analysis
GF-BV A-10	Boundary: End of Generation Z	2009	"Generation Z"	Boundary Value Analysis
GF-BV A-11	Boundary: Start of Generation Alpha	2010	"Generation Alpha"	Boundary Value Analysis
GF-BV A-12	Boundary: End of Generation Alpha	2024	"Generation Alpha"	Boundary Value Analysis

5. Module: main (Simulated I/O)

Test ID	**Description**	**Input (Simulated)**	**Expected Outcome**	**Test Approach**
MAIN-EP-1	Valid input scenario	"13 November 1987"	Console output with birthday, LPN, Lucky Colour, and Generation	Equivalence Partitioning (Simulated I/O)
MAIN-EP-2	Invalid input scenario	"13 XX 1987"	Raises ValueError	Equivalence Partitioning (Simulated I/O)

White-Box Test Cases

1. Module: birthday_checker (White-Box)

Test ID	Internal Component Tested	Input	Expected Internal Behavior	Expected Outcome
WB-BC-1	String splitting and length check	"13 November 1987"	Splits into ["13", "November", "1987"]	Returns (13, 11, 1987)
WB-BC-2	Month normalization (abbreviation mapping)	"13 Nov 1987"	Converts "Nov" to month number 11 using dictionary	Returns (13, 11, 1987)
WB-BC-3	Numeric month conversion	"13 11 1987"	Converts numeric "11" to corresponding month (11)	Returns (13, 11, 1987)
WB-BC-4	Leap year detection and adjustment	"29 February 2000"	Detects leap year; accepts 29 days in February	Returns (29, 2, 2000)
WB-BC-5	Insufficient components (error path)	"13 1987"	Fails length check and raises error	Raises ValueError

2. Module: life_path_calculator (White-Box)

Test ID	Internal Component Tested	Input (day, month, year)	Expected Internal Behavior	Expected Outcome
WB-LPC-1	Digit addition helper function	29	Sums digits: 2 + 9 = 11	Intermediate sum: 11
WB-LPC-2	Recursive reduction function	42	Reduces 4 + 2 = 6	Returns 6
WB-LPC-3	Master number preservation	(29, 2, 1980)	Recognizes master number condition and preserves 22	Returns 22

WB-LPC-4	Overall calculation for non-master input	(1, 1, 2000)	Adds reduced digits: 1 + 1 + 2 = 4	Returns 4

3. Module: color_finder (White-Box)

Test ID	Internal Logic Tested	Input (Life Path Number)	Expected Behavior (internal lookup)	Expected Outcome
WB-CF-1	Lookup for regular number	5	Retrieves value for key 5 from color mapping	"Sky Blue"
WB-CF-2	Lookup for master number	11	Retrieves value for key 11 from mapping	"Silver"

4. Module: generation_finder (White-Box)

Test ID	Internal Logic Tested	Input (Year)	Expected Internal Range Check Behavior	Expected Outcome
WB-GF-1	Check for Silent Generation range	1940	Verifies $1901 \leq 1940 \leq 1945$	"Silent Generation"
WB-GF-2	Check for Baby Boomers range	1960	Verifies $1946 \leq 1960 \leq 1964$	"Baby Boomers"
WB-GF-3	Check for Generation X range	1970	Verifies $1965 \leq 1970 \leq 1979$	"Generation X"
WB-GF-4	Handling a year outside defined ranges	1900	Fails all range checks	"Unknown"

5. Module: main (White-Box)

Test ID	Internal Sequence Tested	Input (Simulated)	Expected Internal Flow	Expected Outcome

WB-MAIN-1	Data flow: input → birthday_checker → processing	"13 November 1987"	Validates date, computes LPN, determines color, checks generation	Correct output printed
WB-MAIN-2	Exception path when validation fails	"13 XX 1987"	Fails during validation; exception raised	Raises ValueError

Test Implementation and Execution

All tests are implemented using Python's unittest framework. The project structure is:

```
/
├── code/
│   ├── source/
│   │   ├── birthday_checker.py
│   │   ├── life_path_calculator.py
│   │   ├── color_finder.py
│   │   └── generation_finder.py
│   ├── test/
│   │   ├── test_birthday_checker.py
│   │   ├── test_life_path_calculator.py
│   │   ├── test_color_finder.py
│   │   ├── test_generation_finder.py
│   │   └── test_main.py
│   └── main.py
└── README.md
```

To Run Tests:

1. Open a Linux terminal and navigate to the project root

Execute:
python -m unittest discover code/test

2.

Example Output:

```
.....................................................
----------------------------------------------------------------------
Ran 55 tests in 0.002s

OK
```

All 55 tests pass, confirming that both the functional outcomes and internal paths are correctly implemented.

Traceability Matrix

Module	Test Design Method	Test Cases Implemented	Comments
birthday_checker	EP, BVA, WB	13+ cases covering valid formats, invalid inputs, and boundaries	Comprehensive date validation tested
life_path_calculator	EP, WB	4 EP cases and 4 WB cases	Correct recursive reduction and master preservation validated
color_finder	EP, WB	4 EP cases; 2 WB cases	Accurate mapping for regular and master numbers confirmed
generation_finder	EP, BVA, WB	7 EP/BVA cases and 4 WB cases	Generation ranges and boundary conditions fully covered
main	Simulated I/O (EP, WB)	2 simulated I/O tests	Overall input/output flow and exception handling verified

Branch Plan

- main - Main development branch for stable code
- code - For implementing codes
- test - For implementing tests
- doc - For preparing documentation

Discussion and Reflection

Achievements:

- Developed a numerology analysis tool using strong modularity principles
- Designed and implemented comprehensive black-box and white-box test cases covering a wide range of input formats, boundary conditions, and internal logic paths
- Systematically refactored code based on a review checklist to improve clarity, maintainability, and error handling
- Effectively used Git for version control, documenting the iterative development process

Challenges:

- Handling multiple date formats and ensuring correct leap year validations
- Creating white-box tests that cover recursive functions without overexposing internal logic
- Balancing extensive test coverage with maintaining a clean, modular codebase

Limitations and Future Work:

- The current implementation provides basic numerology functions; future enhancements could include extended analysis and a graphical user interface
- Future iterations may integrate continuous integration and automated regression testing
- Enhanced error logging and dynamic configuration (e.g., for generation ranges) are potential improvements

Conclusion: This project demonstrates a solid application of modularity, thorough testing (both black-box and white-box), and robust version control practices. The iterative development process and detailed documentation ensure that the solution is both reliable and maintainable.