# Murad Nabil_22683179_ISEReport

- **Assessment:** Introduction to Software Engineering– ISAD1000/5004 2024 Trimester 1
- **Assignment:** Final Assessment - (Assignment Specification V1)
- **Student Name:** Nabil Murad
- **Student ID:** 22683179
- **Date:** 31-March-2025

# Introduction

This project implements a numerology analysis tool. It processes a 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.
- (When two birthdays are provided, compare their Life Path Numbers.)

### Scenario B

- Identify the generation (e.g., 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).

# Scenario

A software company is developing tools for numerology analysis. The key functionalities are:

## Scenario A

- **For a Single Birthday:**
    - Calculate the Life Path Number.
    - Determine the Lucky Colour.
    - Indicate if the Life Path Number is a master number.
- **For Two Birthdays:**
    - Compare if the Life Path Numbers are identical.

## Scenario B

- **Generation Identification:**
    - Determine which generation a person belongs to, based on the birth year (using predefined generation ranges as provided on Blackboard).

# Detailed Description

## Module Descriptions

### 1. `validate_birthday`

- **Name:** validate_birthday
- **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_number`

- **Name:** life_path_number
- **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. `lucky_color`

- **Name:** lucky_color
- **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_checker`

- **Name:** generation_checker
- **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, `validate_birthday` returns a date tuple that is then consumed by `life_path_number`.
- **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:**

A. Open the terminal in the project root.

   Execute the following command:

```
python main.py
```

B. Sample Output:

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


Your birthday         : 13 November 1987

Your life path number : 3

Your lucky color      : Sky Blue

Your generation       : Generation X
```

### 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:**

- *validate_birthday* was simplified by using the `datetime` module and creating helper functions for month normalization and leap year adjustment.
- *life_path_number* had its nested helper functions moved to top-level to improve testability.
- *main* was refactored for dependency injection to simplify testing of user input and output flows.

# 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: validate_birthday**

| Test ID | Description | Input | Expected Output | Test Approach |
|---|---|---|---|---|
| VB-EP-1 | Valid birthday with full month name | "13 November 1987" | (13, 11, 1987) | Equivalence Partitioning |
| VB-EP-2 | Valid birthday with month abbreviation | "13 Nov 1987" | (13, 11, 1987) | Equivalence Partitioning |
| VB-EP-3 | Valid birthday with numeric month | "13 11 1987" | (13, 11, 1987) | Equivalence Partitioning |
| VB-EP-4 | Invalid format (missing components) | "13 1987" | Raises ValueError | Equivalence Partitioning |
| VB-EP-5 | Invalid day (non-numeric) | "AA November 1987" | Raises ValueError | Equivalence Partitioning |
| VB-EP-6 | Invalid month (non-existent) | "13 Novem 1987" | Raises ValueError | Equivalence Partitioning |
| VB-EP-7 | Invalid year (non-numeric) | "13 November XXXX" | Raises ValueError | Equivalence Partitioning |
| VB-EP-8 | Day out of range for month (e.g., Feb 31) | "31 February 2000" | Raises ValueError | Equivalence Partitioning |

| | | | | |
|---|---|---|---|---|
| VB-EP-9 | Year out of valid range | "13 November 2026" | Raises ValueError | Equivalence Partitioning |
| VB-BVA-1 | Boundary: Earliest valid year | "01 January 1925" | (1, 1, 1925) | Boundary Value Analysis |
| VB-BVA-2 | Boundary: Latest valid year | "31 December 2025" | (31, 12, 2025) | Boundary Value Analysis |
| VB-BVA-3 | Just below earliest valid year | "31 December 1924" | Raises ValueError | Boundary Value Analysis |
| VB-BVA-4 | Just above latest valid year | "01 January 2026" | Raises ValueError | Boundary Value Analysis |

**2. Module: life_path_number**

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

**3. Module: lucky_color**

| Test ID | Description | Input (Life Path Number) | Expected Output | Test Approach |
|---|---|---|---|---|
| LC-EP-1 | Mapping for a regular number | 5 | "Sky Blue" | Equivalence Partitioning |
| LC-EP-2 | Mapping for master number 11 | 11 | "Silver" | Equivalence Partitioning |
| LC-EP-3 | Mapping for master number 22 | 22 | "White" | Equivalence Partitioning |
| LC-EP-4 | Mapping for master number 33 | 33 | "Crimson" | Equivalence Partitioning |

**4. Module: generation_checker**

| Test ID | Description | Input (Year) | Expected Output | Test Approach |
|---|---|---|---|---|
| GC-EP-1 | Silent Generation | 1940 | "Silent Generation" | Equivalence Partitioning |
| GC-EP-2 | Baby Boomers | 1960 | "Baby Boomers" | Equivalence Partitioning |
| GC-EP-3 | Generation X | 1970 | "Generation X" | Equivalence Partitioning |
| GC-EP-4 | Millennials | 1990 | "Millennials" | Equivalence Partitioning |

| | | | | |
|---|---|---|---|---|
| GC-EP-5 | Generation Z | 2000 | "Generation Z" | Equivalence Partitioning |
| GC-EP-6 | Generation Alpha | 2015 | "Generation Alpha" | Equivalence Partitioning |
| GC-EP-7 | Year outside valid range | 1900 | "Unknown" | Equivalence Partitioning |
| GC-BVA-1 | Boundary: Start of Silent Generation | 1901 | "Silent Generation" | Boundary Value Analysis |
| GC-BVA-2 | Boundary: End of Silent Generation | 1945 | "Silent Generation" | Boundary Value Analysis |
| GC-BVA-3 | Boundary: Start of Baby Boomers | 1946 | "Baby Boomers" | Boundary Value Analysis |
| GC-BVA-4 | Boundary: End of Baby Boomers | 1964 | "Baby Boomers" | Boundary Value Analysis |
| GC-BVA-5 | Boundary: Start of Generation X | 1965 | "Generation X" | Boundary Value Analysis |
| GC-BVA-6 | Boundary: End of Generation X | 1979 | "Generation X" | Boundary Value Analysis |
| GC-BVA-7 | Boundary: Start of Millennials | 1980 | "Millennials" | Boundary Value Analysis |
| GC-BVA-8 | Boundary: End of Millennials | 1994 | "Millennials" | Boundary Value Analysis |

| | | | | |
|---|---|---|---|---|
| GC-BVA-9 | Boundary: Start of Generation Z | 1995 | "Generation Z" | Boundary Value Analysis |
| GC-BVA-10 | Boundary: End of Generation Z | 2009 | "Generation Z" | Boundary Value Analysis |
| GC-BVA-11 | Boundary: Start of Generation Alpha | 2010 | "Generation Alpha" | Boundary Value Analysis |
| GC-BVA-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: validate_birthday (White-Box)

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

**2. Module: life_path_number (White-Box)**

| Test ID | Internal Component Tested | Input (day, month, year) | Expected Internal Behavior | Expected Outcome |
|---|---|---|---|---|
| WB-LPN-1 | Digit addition helper (add_digit) | 29 | Sums digits: 2 + 9 = 11 (preserving master number if applicable) | Intermediate sum: 11 |
| WB-LPN-2 | Recursive reduction (greater_formatter) | 42 | Reduces 4 + 2 = 6 | Returns 6 |
| WB-LPN-3 | Master number preservation | (29, 2, 1980) | Recognizes master number condition and preserves 22 | Returns 22 |
| WB-LPN-4 | Overall calculation for non-master input | (1, 1, 2000) | Adds reduced digits: 1 + 1 + 2 = 4 | Returns 4 |

**3. Module: lucky_color (White-Box)**

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

**4. Module: generation_checker (White-Box)**

| Test ID | Internal Logic Tested | Input (Year) | Expected Internal Range Check Behavior | Expected Outcome |
|---|---|---|---|---|
| WB-GC-1 | Check for Silent Generation range | 1940 | Verifies 1901 ≤ 1940 ≤ 1945 | "Silent Generation" |
| WB-GC-2 | Check for Baby Boomers range | 1960 | Verifies 1946 ≤ 1960 ≤ 1964 | "Baby Boomers" |
| WB-GC-3 | Check for Generation X range | 1970 | Verifies 1965 ≤ 1970 ≤ 1979 | "Generation X" |
| WB-GC-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 → validate_birthday → 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:

```
/
├── src/
│   ├── __init__.py
│   ├── validate_birthday.py
│   ├── life_path_number.py
│   ├── lucky_color.py
│   └── generation_checker.py
├── tests/
│   ├── __init__.py
│   ├── test_validate_birthday.py
│   ├── test_life_path_number.py
│   ├── test_lucky_color.py
│   └── test_generation_checker.py
└── main.py
```

### To Run Tests:

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

   Execute:

   python -m unittest discover tests

2. **Example Output:**

```
....................

----------------------------------------------------------------------

Ran 61 tests in 0.003s



OK
```

**All 61 tests pass, confirming that both the functional outcomes and internal paths are correctly implemented.**

## Traceability Matrix

| Module | Test Design Method | Test Cases Implemented | Comments |
|---|---|---|---|
| validate_birthday | EP, BVA, WB | 13+ cases covering valid formats, invalid inputs, and boundaries | Comprehensive date validation tested |
| life_path_number | EP, WB | 4 EP cases and 4 WB cases | Correct recursive reduction and master preservation validated |
| lucky_color | EP, WB | 4 EP cases; 2 WB cases | Accurate mapping for regular and master numbers confirmed |
| generation_checker | 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 |

# 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.

# Appendices

## Sample Screenshots

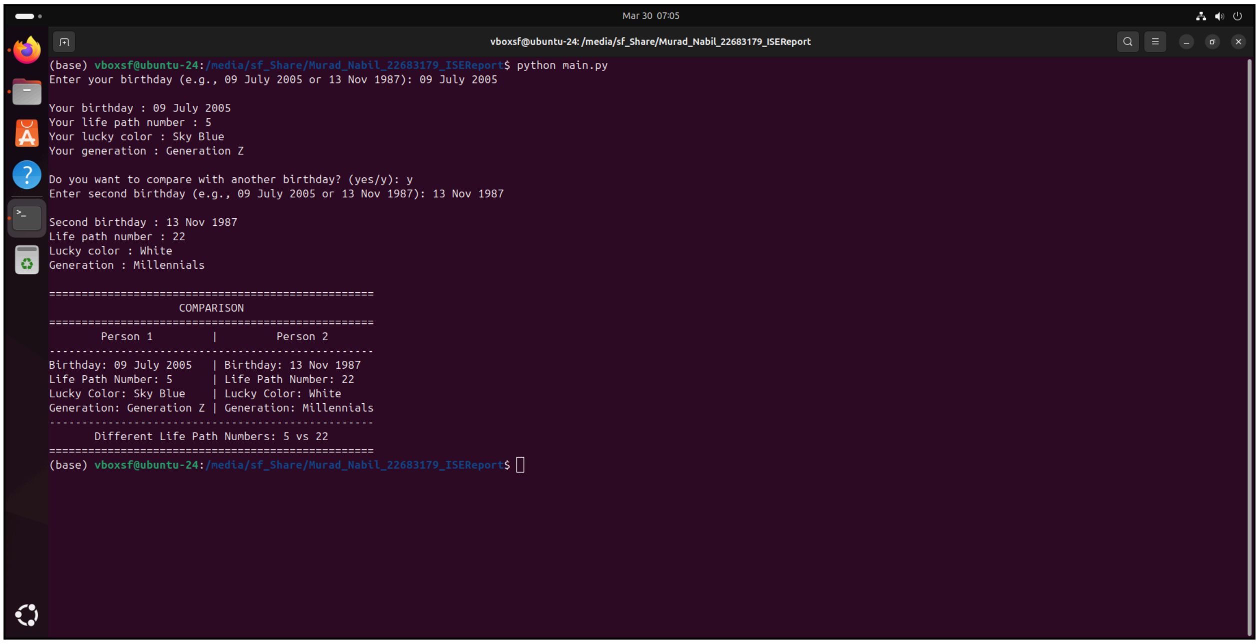


**Figure:** Program Output.

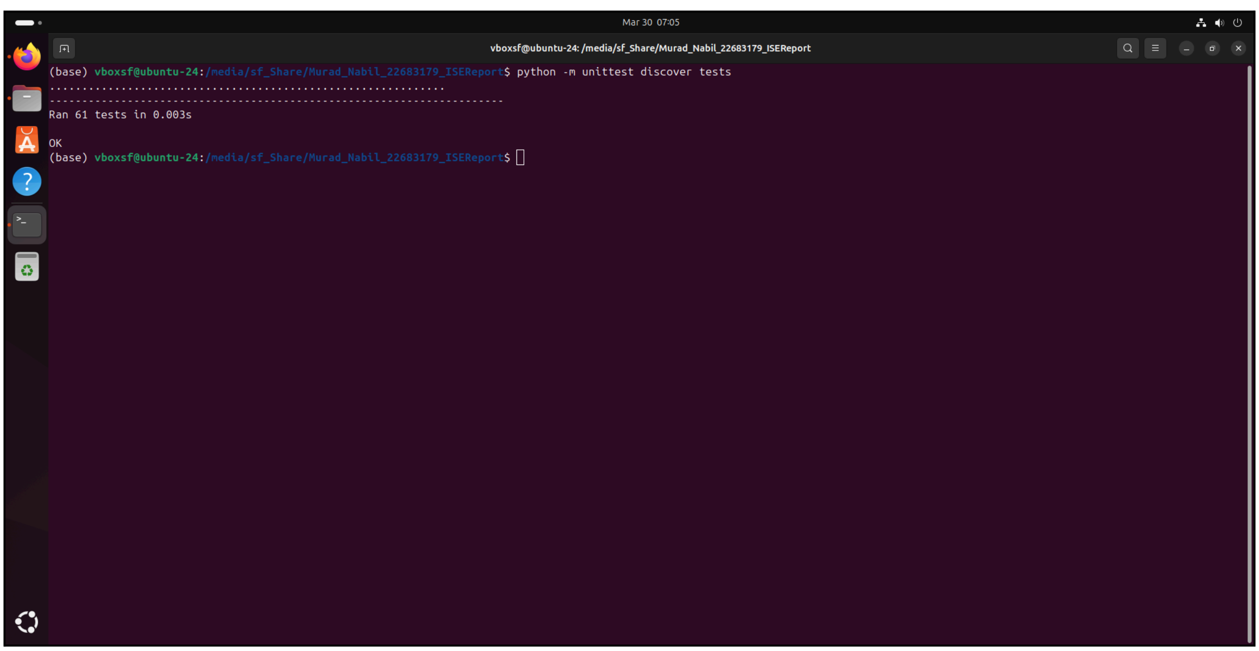


**Figure:** Unittesting.

## Git Log

**Git Repository:** https://github.com/Scythe6699/22683179_ISE_Assignment_2025.git

**Branch Plan:**

- main - Main development branch for stable code
- feature - For implementing features
- testing - For implementing tests
- documentation - For preparing documentation

**Git Log:**

Please check the **git_log.txt** for log files or type `git --no-pager log > git_log.txt` in the cloned repository folder.

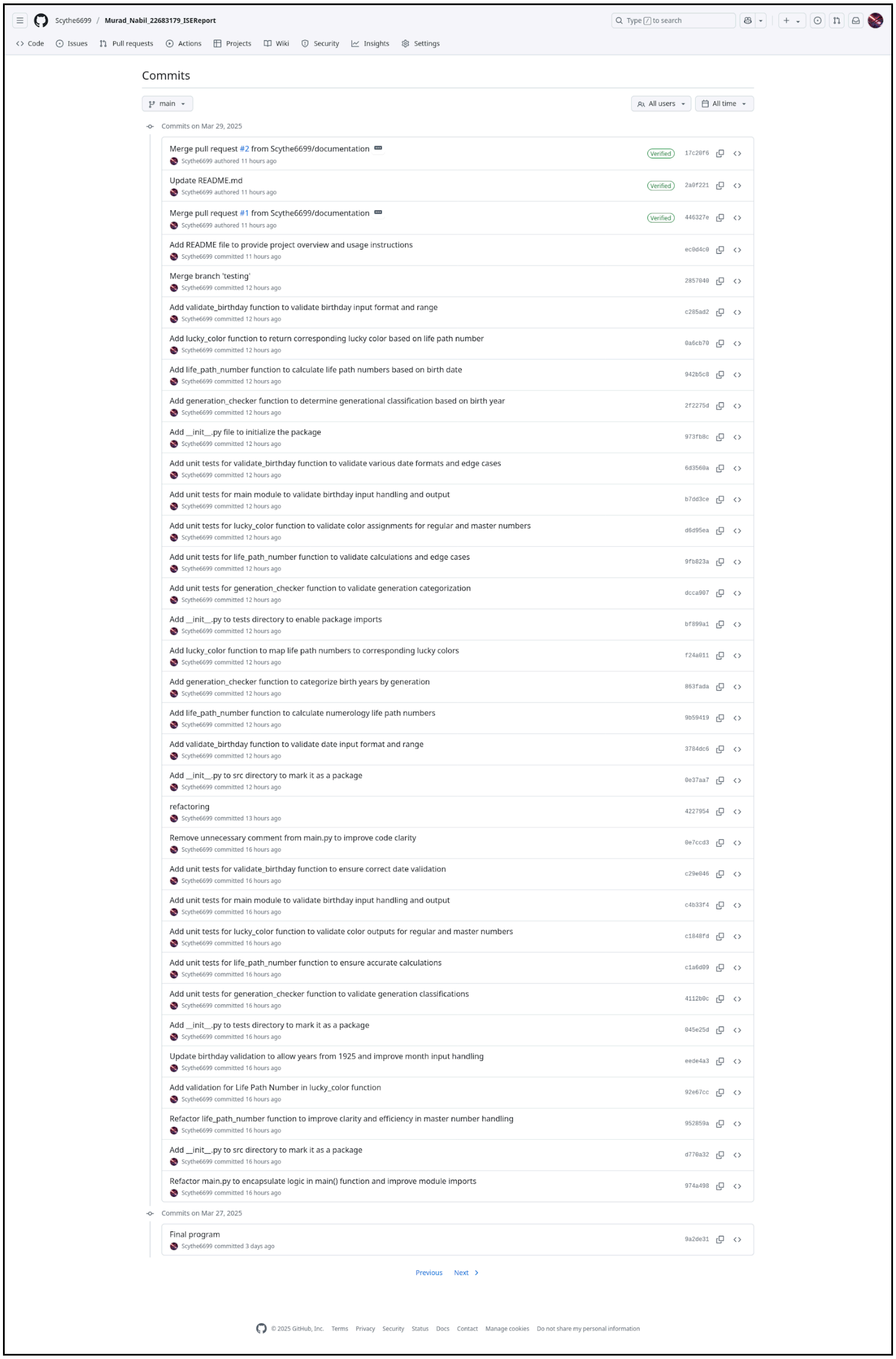

**Figure:** Git Commits 1.

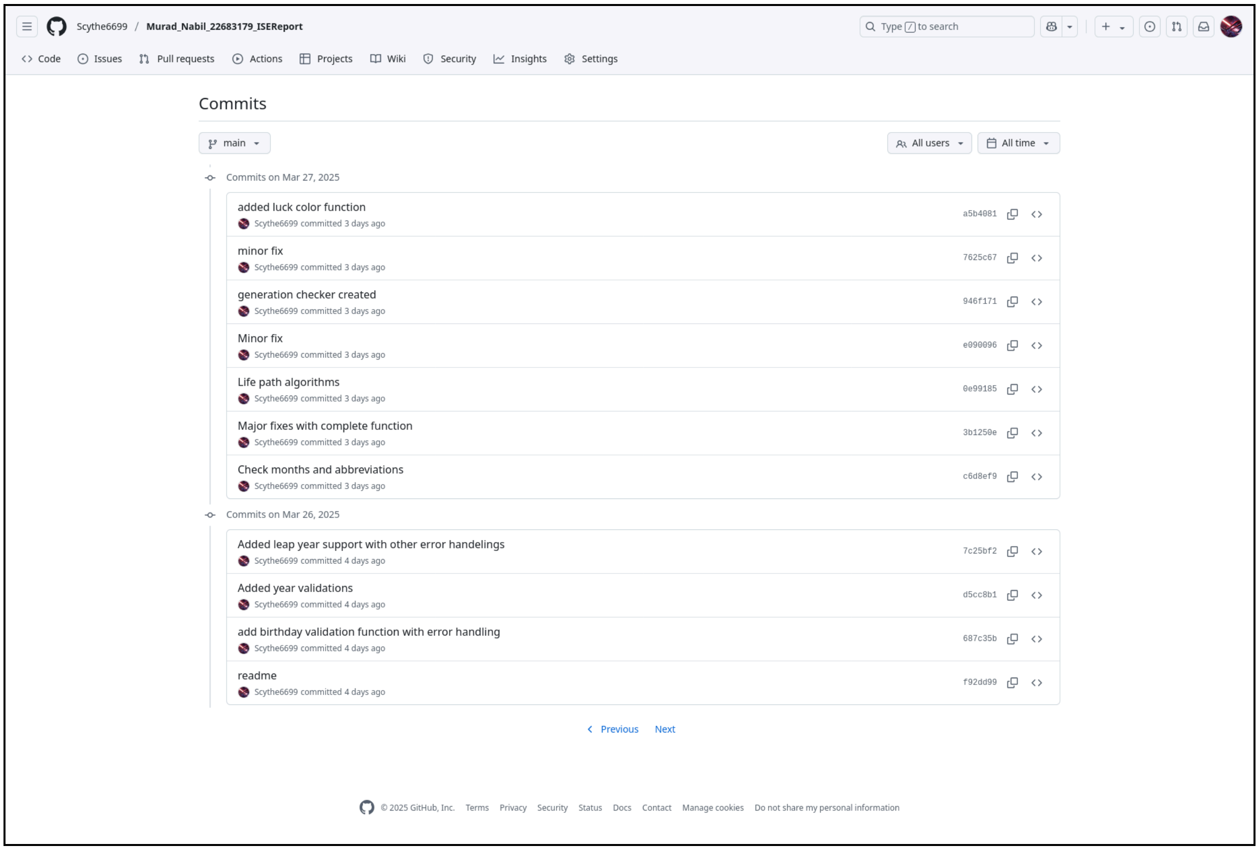


**Figure:** Git Commits 2.