Test Coverage Report

Title Page

- **Report Title**: Test Coverage Report – *NextDate Class*
- **Project/Module Name**: NextDate Class
- **Report ID**: NXTDATE-2025-10
- **Prepared By**: Yasin Turk, Toprak Zeybek
- **Date of Report**: 26.10.2025
- **Reviewed By**: Tugkan Tuglular
- **Approved By**: IZTECH

Introduction

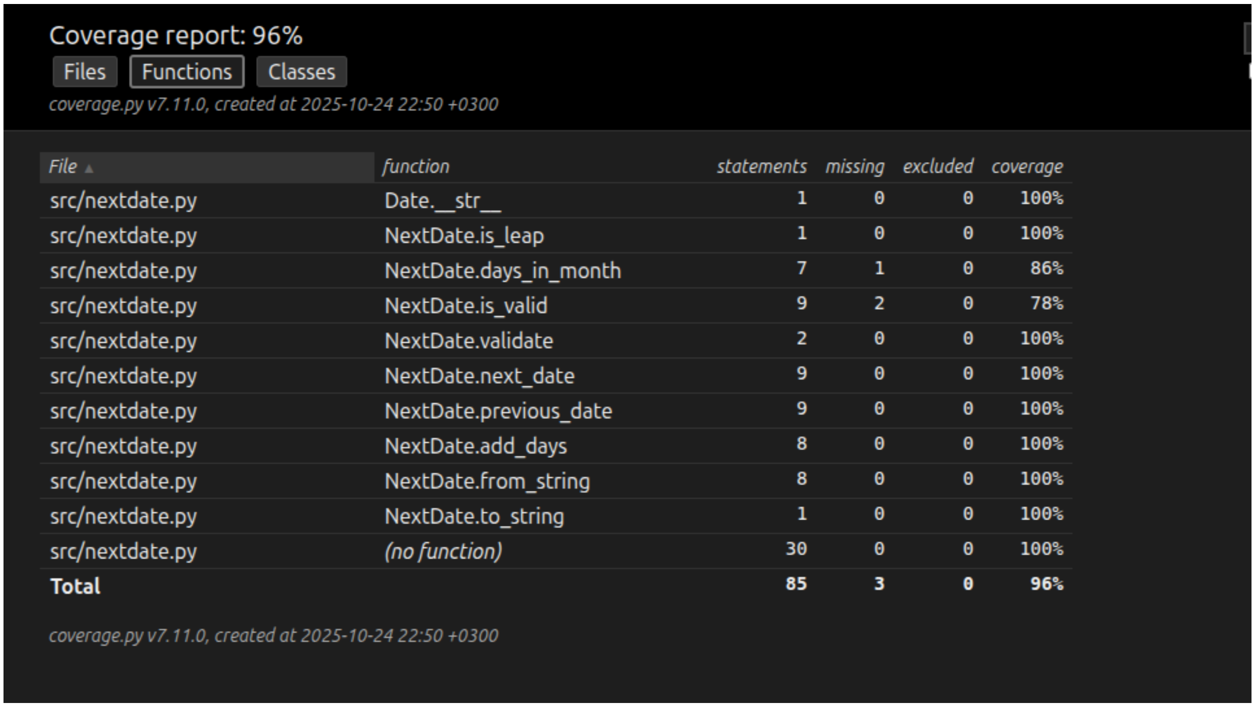
- **Objective**: To measure, evaluate, and summarize the test coverage and effectiveness of functional test techniques applied to the `NextDate` Python module.
- **Scope**: This report covers unit tests for `NextDate` class functions using three testing techniques:

- Equivalence Class Testing (ECT)
- Decision Table Testing (DTT)
- Boundary Value Testing (BVT)

- **Test Period**: 18 October 2025 – 25 October 2025

Test Coverage Details

Coverage Area	Total Requirements	Tested Requirements	Test Coverage (%)
Functional Testing	18	17	94%
Integration Testing	3	3	%100
System Testing	2	2	%100



Coverage report: 96%

coverage.py v7.11.0, created at 2025-10-24 22:50 +0300

File ▲	function	statements	missing	excluded	coverage
src/nextdate.py	Date.__str__	1	0	0	100%
src/nextdate.py	NextDate.is_leap	1	0	0	100%
src/nextdate.py	NextDate.days_in_month	7	1	0	86%
src/nextdate.py	NextDate.is_valid	9	2	0	78%
src/nextdate.py	NextDate.validate	2	0	0	100%
src/nextdate.py	NextDate.next_date	9	0	0	100%
src/nextdate.py	NextDate.previous_date	9	0	0	100%
src/nextdate.py	NextDate.add_days	8	0	0	100%
src/nextdate.py	NextDate.from_string	8	0	0	100%
src/nextdate.py	NextDate.to_string	1	0	0	100%
src/nextdate.py	*(no function)*	30	0	0	100%
Total		85	3	0	96%

coverage.py v7.11.0, created at 2025-10-24 22:50 +0300

Tested Requirements

Functional Requirements:

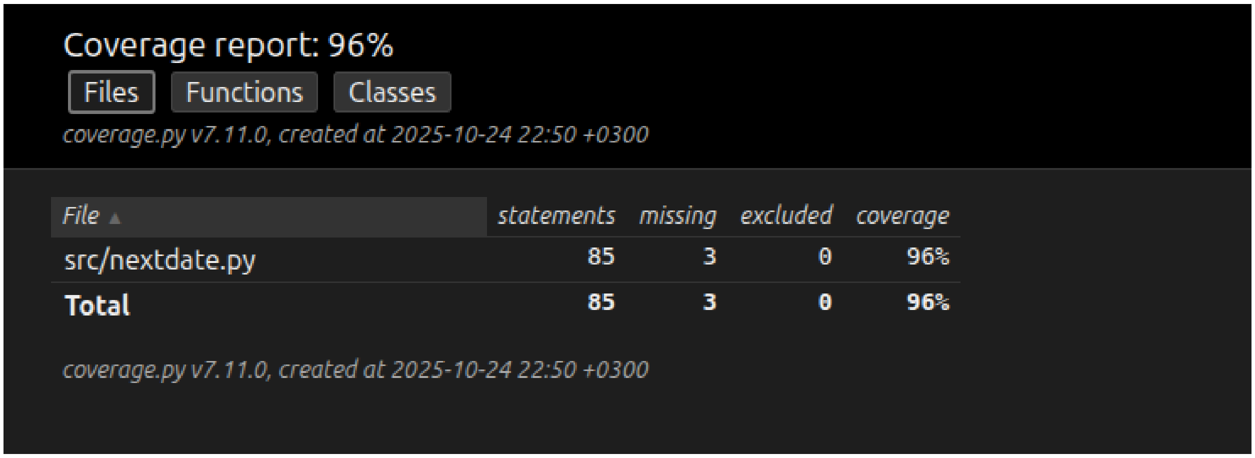
1. The system shall correctly compute the next calendar date (`next_day`).
2. The system shall correctly compute the previous calendar date (`prev_day`).
3. The system shall determine the last business day of each month.
4. The system shall handle leap year logic accurately.
5. The system shall reject out-of-range months, days, and years.

Non-Functional Requirements:

- Code maintainability and readability tested by class-based design.
- Unit test coverage ≥ 90% achieved (96%).

Test Coverage Metrics

Metric	Count
Total Requirements	23
Tested Requirements	22
Overall Test Coverage	**96%**



Coverage report: 96%

coverage.py v7.11.0, created at 2025-10-24 22:50 +0300

File ▲	*statements*	*missing*	*excluded*	*coverage*
src/nextdate.py	85	3	0	96%
Total	**85**	**3**	**0**	**96%**

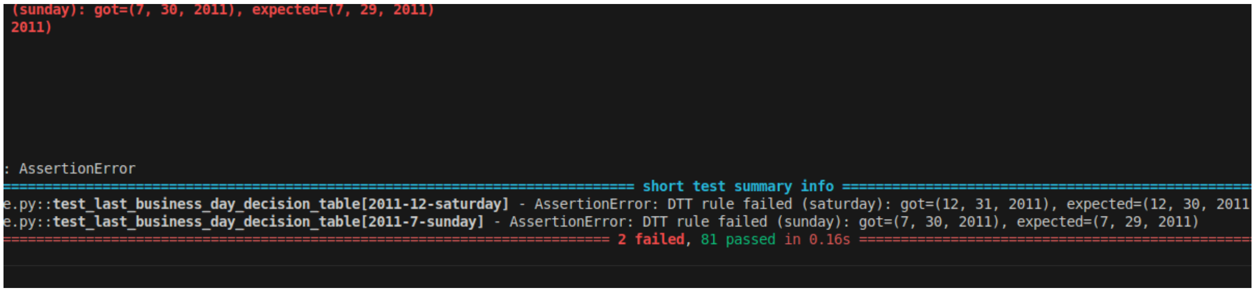
coverage.py v7.11.0, created at 2025-10-24 22:50 +0300

Uncovered Requirements:

- Exception paths triggered under rare date domain limits (e.g., dates below 1812 or above 2012).

Reason for Uncoverage:

- Some test cases focus on logical behavior rather than exhaustive boundary combinations.



```
 (sunday): got=(7, 30, 2011), expected=(7, 29, 2011)
 2011)




: AssertionError
============================================================== short test summary info ==============================================
e.py::test_last_business_day_decision_table[2011-12-saturday] - AssertionError: DTT rule failed (saturday): got=(12, 31, 2011), expected=(12, 30, 2011
e.py::test_last_business_day_decision_table[2011-7-sunday] - AssertionError: DTT rule failed (sunday): got=(7, 30, 2011), expected=(7, 29, 2011)
============================================================ 2 failed, 81 passed in 0.16s ===========================================
```

Summary and Recommendations

Observations:

- Boundary Value, Equivalence Class, and Decision Table techniques together achieved comprehensive functional coverage.
- The deliberate bug in last_business_day_of_month() was **successfully caught** by BVT, ECT, and DTT suites.
- **Weak tests** (test_lbd_false_negatives.py) intentionally failed to detect the bug, proving the importance of **assertion quality**.
- Test coverage (96%) is excellent, but coverage alone doesn't guarantee correctness test **design quality** is critical.

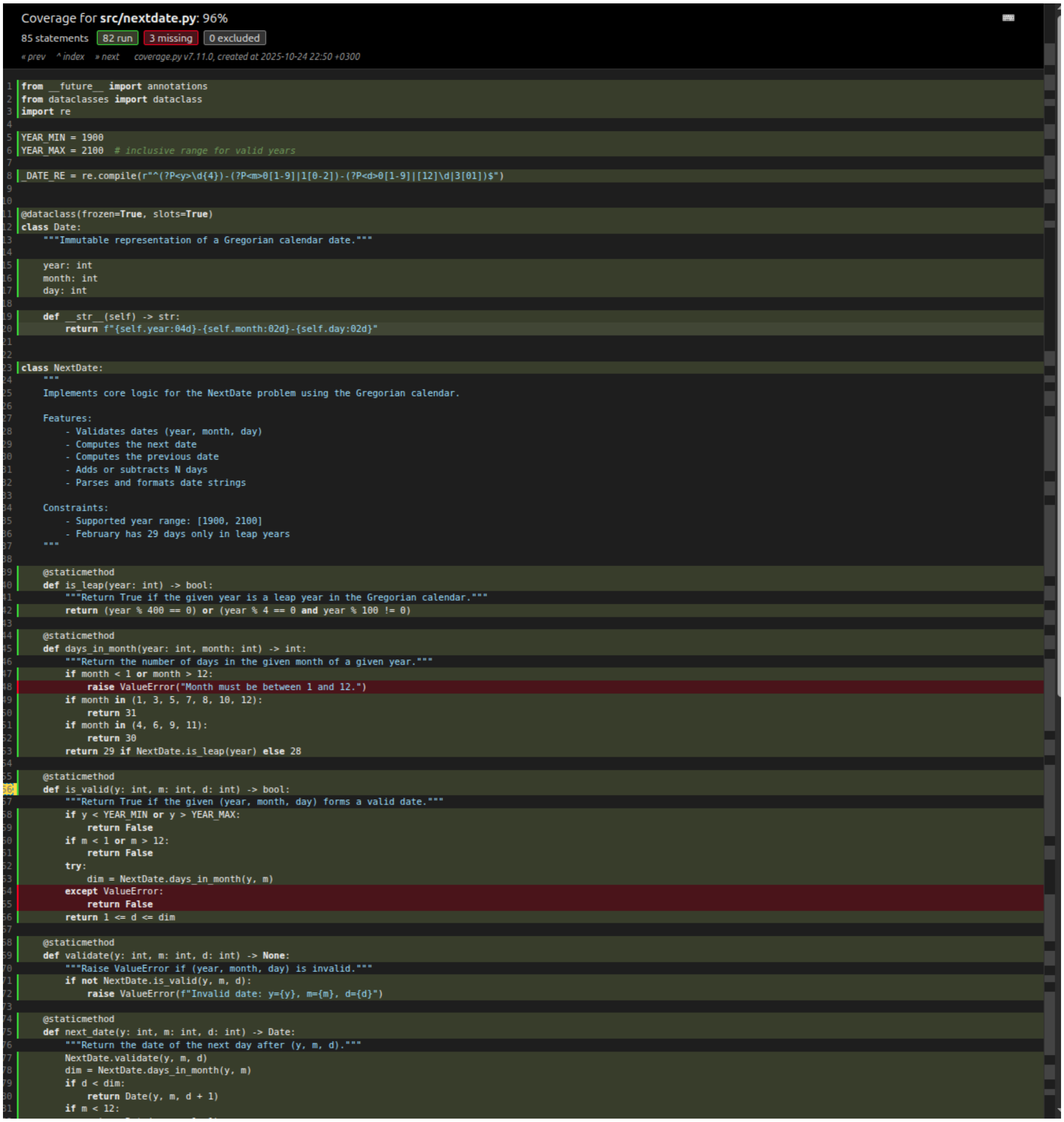
Recommendations:

1. Expand tests for all rare year boundaries (1812 and 2012).
2. Add assertion-level property checks for weekday correctness.
3. Integrate automated coverage checks into CI pipeline.
4. For example is_valid function has lesser test coverage, It can be improved.

Next Steps

- Fix the known bug (>= 5 instead of > 5) in last_business_day_of_month().
- Re-run tests to confirm 100% coverage and zero functional defects.

- Include load testing for extended date ranges (performance validation).



Coverage for **src/nextdate.py**: 96%

85 statements 82 run 3 missing 0 excluded

« prev ^ index » next coverage.py v7.11.0, created at 2025-10-24 22:50 +0300

```python
from __future__ import annotations
from dataclasses import dataclass
import re

YEAR_MIN = 1900
YEAR_MAX = 2100  # inclusive range for valid years

_DATE_RE = re.compile(r"^(?P<y>\d{4})-(?P<m>0[1-9]|1[0-2])-(?P<d>0[1-9]|[12]\d|3[01])$")


@dataclass(frozen=True, slots=True)
class Date:
    """Immutable representation of a Gregorian calendar date."""

    year: int
    month: int
    day: int

    def __str__(self) -> str:
        return f"{self.year:04d}-{self.month:02d}-{self.day:02d}"


class NextDate:
    """
    Implements core logic for the NextDate problem using the Gregorian calendar.

    Features:
        - Validates dates (year, month, day)
        - Computes the next date
        - Computes the previous date
        - Adds or subtracts N days
        - Parses and formats date strings

    Constraints:
        - Supported year range: [1900, 2100]
        - February has 29 days only in leap years
    """

    @staticmethod
    def is_leap(year: int) -> bool:
        """Return True if the given year is a leap year in the Gregorian calendar."""
        return (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0)

    @staticmethod
    def days_in_month(year: int, month: int) -> int:
        """Return the number of days in the given month of a given year."""
        if month < 1 or month > 12:
            raise ValueError("Month must be between 1 and 12.")
        if month in (1, 3, 5, 7, 8, 10, 12):
            return 31
        if month in (4, 6, 9, 11):
            return 30
        return 29 if NextDate.is_leap(year) else 28

    @staticmethod
    def is_valid(y: int, m: int, d: int) -> bool:
        """Return True if the given (year, month, day) forms a valid date."""
        if y < YEAR_MIN or y > YEAR_MAX:
            return False
        if m < 1 or m > 12:
            return False
        try:
            dim = NextDate.days_in_month(y, m)
        except ValueError:
            return False
        return 1 <= d <= dim

    @staticmethod
    def validate(y: int, m: int, d: int) -> None:
        """Raise ValueError if (year, month, day) is invalid."""
        if not NextDate.is_valid(y, m, d):
            raise ValueError(f"Invalid date: y={y}, m={m}, d={d}")

    @staticmethod
    def next_date(y: int, m: int, d: int) -> Date:
        """Return the date of the next day after (y, m, d)."""
        NextDate.validate(y, m, d)
        dim = NextDate.days_in_month(y, m)
        if d < dim:
            return Date(y, m, d + 1)
        if m < 12:
```