# Stability and Correctness of Python's Pickle Serialization

## PA1465 Software Testing

09 July 2025

Abdalrahman Mohammed, abmm22@student.bth.se

Simon Lindqvist, siln22@student.bth.se

Casper Aborres, caab22@student.bth

Rasmus Dunder, radn21@student.bt.se

Alexander Anders Michea Alfaro, almi21@student.bth.se

## Contents

# Introduction

In this project, the goal was to test whether Python's pickle module produces identical results for the same input across different operating systems and Python versions. We focused on stability by pickling various objects, hashing the resulting byte streams, and comparing these hashes across environments. This allowed us to check if pickle behaves consistently and correctly, regardless of system or version differences.

## Github repository

Link to repository: https://github.com/Ekorz-boop/PA1465-project-turn-in

# Method

## Test Suite Architecture and Strategy

The test suite was developed to systematically evaluate the consistency of Python's pickle serialization across different protocols and execution environments. Its architecture is composed of three primary components: the serialization and hashing mechanism, the test case suite, and the execution and comparison modules.

## Serialization and Hashing Approach

A dedicated function was implemented to serialize Python objects using the pickle module and convert the resulting binary data into a SHA-256 hash. This method enables consistent and environment-independent comparison by representing each object as a fixed hexadecimal hash. By comparing these hash values, the system can detect serialization discrepancies without relying on memory addresses or platform-specific representations.

## Test Case Structure

The suite contains a broad range of test cases designed to capture both typical and edge-case behavior. Normal test cases include primitive types (e.g., integers, floats, strings, booleans), container types (e.g., lists, tuples, dictionaries, sets), binary data, empty containers, nested structures, and simple custom objects. Edge test cases address less common scenarios, such as special floating-point values (NaN, infinities), including NumPy-specific representations of positive and negative zero, positive and negative infinity and nan, extremely large or small numbers, Unicode text with emojis and multi-language characters, complex numbers, large-scale data structures (including lists of custom objects and large dictionaries), and special types like bytearray and frozenset. A custom class (ComparisonClass) with user-defined equality logic (__eq__) was also included to evaluate the handling of object comparisons in serialization.

## Execution Strategy and Automation

The execution of test cases is automated via a GitHub Actions workflow. This workflow triggers on pushes and pull requests to the main branches and can also be run manually. The primary test job runs across a matrix of operating systems (Ubuntu, Windows, macOS) and Python versions (3.8, 3.9, 3.10, 3.11). For each combination in the matrix, the workflow checks out the codebase, sets up the specified Python environment, installs dependencies, and executes main.py. The main.py serializes each test case object using the custom hash function across all six supported pickle protocols (0 to 5). The outputs are recorded in a structured format, with robust error handling. The results, including environment metadata (OS and Python version derived from the matrix runner), are stored in uniquely named JSON files (e.g., OS_ubuntu-latest_PYTHON_3.8.json). Each such JSON file is then uploaded as a build artifact.

## Comparison and Analysis Methodology

A subsequent consolidate-and-commit job in the GitHub Actions workflow does the comparison and reporting. This job downloads all JSON result artifacts generated by the matrix test jobs into a common directory. It then runs comparison.py, which processes all the collected JSON result files. The script extracts hash values and environment details (OS and Python version from filenames) and performs pairwise comparisons to identify inconsistencies. The analysis targets protocol-level differences within the same environment and environment-level differences under the same protocol, categorized by test case. The outcome is compiled into multiple report formats: a raw JSON file (comparison_results.json) detailing all comparisons and a structured HTML report (comparison_report.html) for easier review of discrepancies. Finally, all individual JSON result files and the generated comparison reports are committed to a results/ directory in the repository.

# Results

## Traceability matrix

| **Test Category** | **Test Cases** | **Coverage** | **Protocol Support** | **Platform Stability** |
|---|---|---|---|---|
| simple_int | Integer value 42 | Complete | All protocols | Stable |
| simple_float | Float value 3.14 | Complete | All protocols | Stable |
| simple_string | String value “Hello, pickle!” | Complete | All protocols | Stable |
| simple_bool | Boolean value False | Complete | All protocols | Stable |
| simple_none | None | Complete | All protocols | Stable |
| simple_list | List containing [1, 2, 3, 4, 5] | Complete | All protocols | Stable |
| simple_tuple | Tuple containing (10, 20, 30) | Complete | All protocols | Stable |
| simple_dict | Dictionary with keys “key1”: “value1” and “key2”: 42 | Complete | All protocols | Stable |
| simple_set | Set containing {1, 2, 3, 4} | Complete | All protocols | Stable |
| bytes_data | Bytes value b"binary data" | Complete | All protocols | Stable |
| custom_object | Instance of ComparisonClass("test_object", 12345) | Complete | All protocols | Stable |
| nested_structure | Dictionary {"name": "nested", "data": [1, 2, {"inner": True}]} | Complete | All protocols | Stable |
| empty_containers | Empty list, empty dict, empty tuple, empty set | Complete | All protocols | Stable |
| inf | Floating point infinity (float("inf")) | Complete | All protocols | Stable |
| neg_inf | Negative infinity (float("-inf")) | Complete | All protocols | Stable |
| nan | Not-a-number (float("nan")) | Complete | All protocols | Stable |
| numpy_pos_zero | NumPy positive zero (np.float64(0.0)) | Complete | All protocols | Unstable on Python 3.8 |
| numpy_neg_zero | NumPy negative zero (np.float64(-0.0)) | Complete | All protocols | Unstable on Python 3.8 |
| numpy_pos_inf | NumPy positive infinity (np.float64(np.inf)) | Complete | All protocols | Unstable on Python 3.8 |
| numpy_neg_inf | NumPy negative infinity (np.float64(-np.inf)) | Complete | All protocols | Unstable on Python 3.8 |
| numpy_nan | NumPy NaN (np.float64(np.nan)) | Complete | All protocols | Unstable on Python 3.8 |
| very_large | Very large integer (10**1000) | Complete | All protocols | Stable |
| very_small | Very small float (1e-100) | Complete | All protocols | Stable |
| emoji | Unicode string containing emojis (“😃 🌍 🚀”) | Complete | All protocols | Stable |
| multi_language | Unicode string “English, 中文, Русский, العربية” | Complete | All protocols | Stable |
| complex_num | Complex number 3 + 4j | Complete | All protocols | Stable |
| custom_objects | List of ComparisonClass instances for i in range(5) | Complete | All protocols | Stable |
| large_dict | Dictionary with 1,000 entries ({str(i): i for i in range(1000)}) | Complete | All protocols | Stable |
| byte_array | Bytearray value bytearray(b"mutable bytes") | Complete | All protocols | Stable |
| frozen_set | Frozen set frozenset([1, 2, 3]) | Complete | All protocols | Stable |

# Discussion

## Output Consistency

Pickled outputs demonstrated high consistency across the tested Python environments (versions 3.8, 3.9, 3.10, and 3.11 on Ubuntu, Windows, and macOS) for most data types. The test suite specifically confirmed consistent serialization for these common cases. A notable observation was in the handling of NumPy float64 datatype. NumPy’s representations of positive and negative float, as well as positive and negative infinity and nan, produced distinct hash values in the Python 3.8 environment when compared to later versions (3.9, 3.10, 3.11) across all operating systems. Other non-NumPy floating-point values like NaN and infinities, however, were serialized consistently across all tested Python 3.8+ versions. These discrepancies with NumPy's signed zeros in Python 3.8 likely stem from minor differences in the CPython implementation or linked libraries affecting IEEE 754 floating-point value representation during serialization, a known area for subtle cross-version variations. Excluding the specific NumPy behaviour in Python 3.8, the test categories showed high output consistency from Python 3.8 through 3.11. Python 3.7 and earlier versions was not included in the automated test matrix due to compatibility limitations with current GitHub Actions runners (specifically, newer Ubuntu versions not fully supporting Python 3.7 and the move to ARM64 architecture for macOS runners making older x86_64 Python builds less straightforward to integrate).

## Coverage

The test suite provides comprehensive coverage for pickle serialization across Python versions 3.8, 3.9, 3.10, and 3.11, executed on Ubuntu, Windows, and macOS. It includes a wide array of Python's built-in types, encompassing primitives such as integers, floats, strings, booleans, and None; containers including lists, tuples, dictionaries, and both mutable and immutable sets (with empty versions of each); and binary data types such as bytes and bytearray. Complex structures are also included, for example nested dictionaries and lists. The suite addresses a variety of challenging scenarios, including floating-point specials like positive and negative infinity (float('inf') and float('-inf')), NaN, and NumPy's representations of positive and negative zero (np.float64(0.0) and np.float64(-0.0)). Numeric extremes (very large and very small numbers) are covered, as are textual variations that involve Unicode strings containing emojis and multi-language characters. Specialized types such as complex numbers, as well as large-scale data structures large dictionaries and lists populated with custom objects are tested. Custom objects include a ComparisonClass featuring custom equality logic, ensuring that user-defined object serialization and comparison are validated. Finally, all standard pickle protocols, from 0 through 5, are exercised for each test case to verify compatibility across protocols.

## Findings

For most test cases, spanning primitives, standard containers, custom objects, and most edge cases, pickle serialization is consistent across Python versions 3.8, 3.9, 3.10, and 3.11 on all tested operating systems (Ubuntu, Windows, and macOS). This high level of consistency indicates a stable serialization format for common Python objects within these modern Python releases. The primary inconsistency observed involves NumPy's np.float64 representations of float64. In Python 3.8, these values produced different SHA-256 hashes compared to Python 3.9, 3.10, and 3.11; this discrepancy was consistent across all operating systems running Python 3.8. Other floating-point edge cases not using NumPy, including nan and infinities, remained consistent across Python 3.8 and later versions. Within any single environment, that is, each combination of operating system and Python version, all pickle protocols (0 through 5) generally produced the same hash for a given object. This suggests reliable intra-protocol behaviour, and no significant protocol-dependent differences were detected in the initial comparisons. Excluding the NumPy anomalies in Python 3.8, Python 3.9, 3.10, and 3.11 demonstrated strong cross-platform consistency for pickle serialization.

## Limitations and Shortcomings

Python 3.7 and earlier versions were not included in the automated test matrix due to compatibility issues with modern GitHub Actions runners; for example, Ubuntu 24's support for older Python versions and macOS ARM64 architecture constraints made it impractical to test these earlier releases. Although Python 3.8 introduced significant language changes, that could cause differences compared to Python 3.7, these effects could not be empirically verified within the current test framework, thereby limiting the historical scope of the consistency analysis. The methodology relies on SHA-256 hashes of pickled byte streams. While this approach effectively detects any difference in the serialized output, it does not inherently distinguish between benign representational changes (for example, different yet functionally equivalent pickle encodings) and changes that are functionally significant. Given pickle's design, however, most byte-level variations are likely meaningful. Although operating system and Python version are tracked, more granular environmental factors, such as specific CPU architectures beyond the operating system level, micro-versions of Python if they are not fully captured, and underlying C library versions, are not explicitly controlled or reported, even though their effects are implicitly tested by running on standard GitHub runners. The scope of custom objects tested is limited to a single ComparisonClass featuring custom equality logic; more complex class hierarchies or those employing advanced features like custom getstate and setstate methods could reveal additional serialization nuances. Finally, the test suite focuses exclusively on output consistency and does not assess performance attributes such as speed or size of pickling and unpickling operations.

## Conclusion

The test suite successfully demonstrates a high degree of pickle serialization consistency for a wide range of Python objects and edge cases across Python versions 3.8, 3.9, 3.10, and 3.11 on Ubuntu, Windows, and macOS. The primary variation observed was in the serialization of NumPy's positive and negative zero floating-point values within the Python 3.8 environment, compared to later versions. Other data types, including common primitives, containers, and other floating-point edge cases, remained consistent across the tested Python versions and operating systems. Automated testing via GitHub Actions provides a robust mechanism for ongoing verification. Although the exclusion of Python 3.7 and earlier versions limits direct comparison with earlier behaviours, the findings affirm the reliability of pickle for data interchange in contemporary Python ecosystems (3.8 and above). Future work could explore methods to include older Python versions if runner compatibility improves or investigate more deeply the root cause of the NumPy variation in Python 3.8. Overall, the results instil confidence in pickle's stability for the tested configurations, with a specific point of attention regarding NumPy's float64 representations in Python 3.8.

## Contributions

All team members worked closely together throughout the project, assisting each other with various tasks. Everyone contributed to coding, testing, and drafting the report. We employed pair programming for the implementation, collaborating locally over Discord. AI tools were utilized to help write GitHub Workflow test scripts and check the grammar and structure enhancements in the final report. The entire project was completed both on-site at school and remotely over Discord, with all team members attending every meeting.