Python Code Style and PEP 8

PEP 8 is the official style guide for Python code. Following it makes your code more readable and consistent with the broader Python ecosystem.

Key PEP 8 guidelines:

Indentation: Use 4 spaces per indentation level. Never mix tabs and spaces.

Line Length: Limit all lines to 79 characters for code, 72 for comments and docstrings. Use parentheses for implicit line continuation.

Imports: Always put imports at the top of the file. Group them in this order:
1. Standard library imports
2. Related third-party imports
3. Local application/library specific imports

Naming Conventions:
- Variables and functions: snake_case
- Classes: CamelCase
- Constants: UPPER_SNAKE_CASE
- Private attributes: _leading_underscore
- Name-mangled attributes: __double_leading_underscore

Whitespace:
- No extra spaces inside parentheses, brackets, or braces
- One space around assignment operators
- No space before a colon in slices

Type Hints (PEP 484):
Python 3.5+ supports type hints. They make code more readable and enable static analysis tools like mypy to catch type errors before runtime.

Example:
```
def greet(name: str) -> str:
    return f"Hello, {name}"
```

Error Handling and Testing

Proper error handling and testing are essential for robust Python code.

Exception Handling:
- Use specific exception types, not bare except clauses
- Use try/except/else/finally for comprehensive error handling
- Create custom exception classes for domain-specific errors
- Use context managers (with statement) for resource management
- Never silently swallow exceptions

Example pattern:
```
try:
   result = process_data(input_data)
except ValueError as e:
   logger.error(f"Invalid data: {e}")
   raise
except ConnectionError:
   retry_with_backoff()
else:
   save_result(result)
finally:
   cleanup_resources()
```

Testing with pytest:
- Write tests in files named test_*.py
- Use descriptive test function names: test_user_creation_with_valid_email
- Use fixtures for shared setup code
- Use parametrize for testing multiple inputs
- Aim for high code coverage but focus on critical paths
- Use mocking to isolate units of code

Test structure (Arrange-Act-Assert):
```
def test_calculate_total():
   # Arrange
   items = [Item(price=10), Item(price=20)]
   # Act
   total = calculate_total(items)
   # Assert
   assert total == 30
```