Introduce Assertion

Add runtime checks to **validate assumptions** and catch programming errors early

Code Smell

```
# We need a condition that should be met in code.
def method(value): ... # value should be positive

=>

def method(value):
  assert  value > 0
```

Example: Sort

Before: We only comment the requirement.

```
def sort(self):
    """Selection sort implementation"""
    for x in range(len(self.data) - 1):
        m = x
        for y in range(x + 1, len(self.data)):
            if self.data[m] > self.data[y]:
                m = y
        # Here data[m] should be the minimum of data~ data[len(data)-1]
        v = self.data[m]
        self.data[m] = self.data[x]
        self.data[x] = v
        # Here data[0] ~ data[x+1] should already be ed
```

After: we add `assert` to make sure of it

```python
def sort(self):
    """Selection sort implementation with assertions"""
    for x in range(len(self.data) - 1):
        m = x
        for y in range(x + 1, len(self.data)): ...
        assert self._is_min(m, x, len(self.data) - 1)

        v = self.data[m]; ...
        assert self._is_sorted(0, x + 1)
```

Assert in Unit Test

The assertEqual (in UnitTest) uses the same assert function.

```python
import unittest
from SortSample import SortSample

class MainTest(unittest.TestCase):
    def test_main(self):
        sorter = SortSample([3, 1, 4, 1, 5, 9])
        sorter.sort()
        actual = str(sorter)
        expected = "[ 1, 1, 3, 4, 5, 9, ]"
        self.assertEqual(expected, actual)


...
if __name__ == "__main__":
    unittest.main()
```

```
..F
======================================================================
FAIL: test_zero (__main__.MainTest.test_zero)
----------------------------------------------------------------------
Traceback (most recent call last):
  method test_zero in MainTest.py at line 17
    self.assertEqual(expected, actual)
    ~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
AssertionError: '[ ]' != '[ , ]'
- [ ]
+ [ , ]
?  ++



----------------------------------------------------------------------
Ran 3 tests in 0.000s

FAILED (failures=1)
```

Tip - Use ASSERT control flag for Development

We can control the execution of assert using a flag.

```python
ASSERT = False  # Set to False to delete assertions

def main():
    x = -123
    if ASSERT: assert x > 0

if __name__ == "__main__":
    main()
```

Tip - Always use assertion

- `assert` is for development: it helps catch bugs and assumptions.
- In optimized mode (`-O` or `-OO`), all assert statements are ignored.

```
python -O myscript.py
```

Tip - Assertion cannot replace error processing

```python
def find_file(filepath):
    # BAD PRACTICE: Using assert for error handling
    assert os.path.exists(filepath), f"File not found: {filepath}"
    with open(filepath, 'r') as f:
        return f.read()
```

Always make code to prevent errors:

```python
def find_file(filepath):
    # Good practice: Explicit error handling
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File not found: {filepath}")
    with open(filepath, 'r') as f:
        return f.read()

# Usage
try:
    content = find_file("important.txt")
    print(content)
except FileNotFoundError as e:
    print(e)
```

Discussion

Why Assertion?

1. **Early error detection** - catches bugs closer to their source
2. **Documentation** - makes assumptions explicit
3. **Debugging aid** - helps identify where problems occur
4. **Code reliability** - increases confidence in code correctness
5. **Improves runtime performance** — assertions optimize performance