Laboratorium 14 - TDD

Marek Lechowicz

1. **Faza RED** - Zgodnie z metodyką TDD rozpoczynamy od napisania testu sprawdzającego istotne funkcjonalności:

```python
bubble_sort_testdata = [
    ([1, 6, 7, 9, 4, 15, -3], [-3, 1, 4, 6, 7, 9, 15]),
    ([3.5, 0.001, 69, 199, -0.01, 0, -1000], [-1000, -0.01, 0, 0.001, 3.5, 69, 199]),
    ([9, 20, 'poniedzialek', 99, 100], None),
    (1905, None)
]

@pytest.mark.parametrize('input_list, sorted_list', bubble_sort_testdata)
def test_bubble_sort(input_list, sorted_list):
    assert bubble_sort(input_list) == sorted_list
```

2. **Faza GREEN** - zapisujemy działający kod realizujący zadaną funkcjonalność - w tym wypadku sortowanie bąbelkowe:

```python
def bubble_sort(lst):
    if type(lst) is not list:
        return None
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if type(lst[j]) not in [int, float] or type(lst[j+1]) not in [int, float]:
                return None
            if lst[j] > lst[j+1]:
                temp = lst[j]
                lst[j] = lst[j+1]
                lst[j+1] = temp
    return lst
```

3. **Faza REFACTOR** - poprawa kodu poprzez zwiększenie czytelności i zmniejszenie złożoności, w tym wypadku będzie to pozbycie się zmiennej *temp* i użycie bardziej pythonowskiej zamiany

```python
def bubble_sort(lst):
    if type(lst) is not list:
        return None
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if type(lst[j]) not in [int, float] or type(lst[j+1]) not in [int, float]:
                return None
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst
```

4. Wnioski

Podejście TDD pozwala na prowadzenie projektu w ustrukturyzowany sposób poprzez podzielenie pracy na trzy fazy:

a) Utworzenie testów
b) Napisanie działającego kodu
c) Usprawnienie znalezionego rozwiązania

Taki sposób pracy wydaje się być szczególnie przydatny w bardziej rozbudowanych i grupowych projektach, ponieważ zdefiniowanie konkretnych wymagań pozwala uniknąć chaosu spowodowanego niedookreśleniem pewnych funkcjonalności.

Framework Pytest pozwala na dość szybkie i proste napisanie kodu testującego, a dekorator *@pytest.mark.parametrize* ułatwia przeprowadzenie testó dla wielu instancji.

Podczas pisania testów, tak jak podczas tworzenia całego projektu, należy pamiętać o zachowaniu porządku w strukturze plików - wszystkie testy powinny znaleźć się w folderze **test/**, a poszczególne pliki testowe powinny zawierać w swojej nazwie na początku lub na końcu słowo *test*.