



Politechnika Krakowska
Wydział Informatyki i Telekomunikacji

Studia Stacjonarne

Sprawozdanie z przedmiotu:

Zaawansowane Techniki Programowania

Laboratorium nr 6

Wykonał:

Jan Kopeć

Zadanie 1. Implementacja kalkulatora z wykorzystaniem TDD.

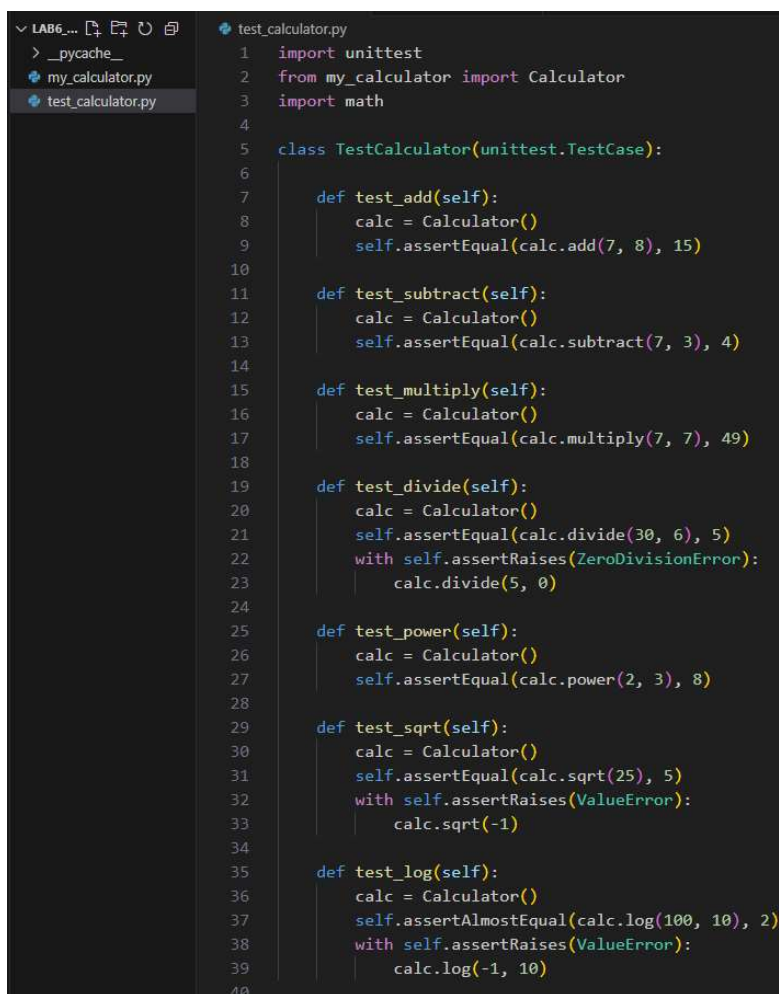
Opis zadania:

Należy stworzyć Kalkulator wykorzystując podejście TDD, kalkulator powinien obsługiwać następujące operacje:

- Dodawanie,
- Odejmowanie,
- Mnożenie,
- Dzielenie,
- Potęgowanie,
- Pierwiastkowanie,
- Logarytm o dowolnej podstawie.

Implementacja [Python 3.8.10, VS Code]

1. Napisanie testów jednostkowych dla każdej z funkcji



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files: `__pycache__`, `my_calculator.py`, and `test_calculator.py`. The code editor displays the content of `test_calculator.py`, which contains a `TestCalculator` class inheriting from `unittest.TestCase`. The class defines ten test methods: `test_add`, `test_subtract`, `test_multiply`, `test_divide`, `test_power`, `test_sqrt`, and `test_log`. Each test method creates a `Calculator` instance and asserts the result of a specific operation. The `test_divide`, `test_sqrt`, and `test_log` methods also use `with self.assertRaises` to verify that specific exceptions are raised for invalid inputs.

```
1 import unittest
2 from my_calculator import Calculator
3 import math
4
5 class TestCalculator(unittest.TestCase):
6
7     def test_add(self):
8         calc = Calculator()
9         self.assertEqual(calc.add(7, 8), 15)
10
11     def test_subtract(self):
12         calc = Calculator()
13         self.assertEqual(calc.subtract(7, 3), 4)
14
15     def test_multiply(self):
16         calc = Calculator()
17         self.assertEqual(calc.multiply(7, 7), 49)
18
19     def test_divide(self):
20         calc = Calculator()
21         self.assertEqual(calc.divide(30, 6), 5)
22         with self.assertRaises(ZeroDivisionError):
23             calc.divide(5, 0)
24
25     def test_power(self):
26         calc = Calculator()
27         self.assertEqual(calc.power(2, 3), 8)
28
29     def test_sqrt(self):
30         calc = Calculator()
31         self.assertEqual(calc.sqrt(25), 5)
32         with self.assertRaises(ValueError):
33             calc.sqrt(-1)
34
35     def test_log(self):
36         calc = Calculator()
37         self.assertAlmostEqual(calc.log(100, 10), 2)
38         with self.assertRaises(ValueError):
39             calc.log(-1, 10)
40
```

2. Uruchomienie testów (nie powinny przejść).

```
jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ python3 -m unittest test_calculator.py
EEEEEE

=====
ERROR: test_add (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 9, in test_add
    self.assertEqual(calc.add(7, 8), 15)
AttributeError: 'Calculator' object has no attribute 'add'

=====
ERROR: test_divide (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 21, in test_divide
    self.assertEqual(calc.divide(30, 6), 5)
AttributeError: 'Calculator' object has no attribute 'divide'

=====
ERROR: test_log (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 37, in test_log
    self.assertAlmostEqual(calc.log(100, 10), 2)
AttributeError: 'Calculator' object has no attribute 'log'

=====
ERROR: test_multiply (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 17, in test_multiply
    self.assertEqual(calc.multiply(7, 7), 49)
AttributeError: 'Calculator' object has no attribute 'multiply'

=====
ERROR: test_power (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 27, in test_power
    self.assertEqual(calc.power(2, 3), 8)
AttributeError: 'Calculator' object has no attribute 'power'

=====
ERROR: test_sqrt (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 31, in test_sqrt
    self.assertEqual(calc.sqrt(25), 5)
AttributeError: 'Calculator' object has no attribute 'sqrt'

=====
ERROR: test_subtract (test_calculator.TestCalculator)
=====
Traceback (most recent call last):
  File "/home/jankopec/LAB6_ZTP/test_calculator.py", line 13, in test_subtract
    self.assertEqual(calc.subtract(7, 3), 4)
AttributeError: 'Calculator' object has no attribute 'subtract'

=====
Ran 7 tests in 0.001s
FAILED (errors=7)
```

Testy nie przechodzą, gdyż mój kalkulator nie obsługuje jeszcze żadnych operacji:

```
LAB6_ZTP [WSL: UBUNTU-...] my_calculator.py
> _pycache_
my_calculator.py
1 class Calculator:
2     pass
```

3. Implementacja minimalnej wersji funkcji, aby test przeszedł.

Do kalkulatora dodaję operację dodawania:

```
my_calculator.py
1 import math
2
3 class Calculator:
4     def add(self, a, b):
5         return a + b
```

Wykonuję testy ponownie – tym razem otrzymuję 6 błędów co świadczy o tym, że funkcja add działa poprawnie.

```
-----
Ran 7 tests in 0.001s
FAILED (errors=6)
```

4. Powtórzenie cyklu dla każdej funkcji.

Implementuję pozostałe operacje:

```
my_calculator.py
1  import math
2
3  class Calculator:
4      def add(self, a, b):
5          return a + b
6
7      def subtract(self, a, b):
8          return a - b
9
10     def multiply(self, a, b):
11         return a * b
12
13     def divide(self, a, b):
14         if b == 0:
15             raise ZeroDivisionError("Nie można dzielić przez zero!")
16         return a / b
17
18     def power(self, a, b):
19         return a ** b
20
21     def sqrt(self, a):
22         if a < 0:
23             raise ValueError("Nie można obliczyć pierwiastka z liczby ujemnej!")
24         return math.sqrt(a)
25
26     def log(self, a, base):
27         if a <= 0 or base <= 0:
28             raise ValueError("Argumenty logarytmu muszą być dodatnie!")
29         return math.log(a, base)
```

Wykonuje testy:

```
jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ python3 -m unittest test_calculator.py
.....
-----
Ran 7 tests in 0.000s
OK
```

Wszystkie testy przechodzą.

Dodatkowo w środowisku VS Code konfiguruję testy jednostkowe za pomocą unittest, aby móc wizualnie śledzić wyniki testów:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TEST RESULTS  TERMINAL  PORTS
Received test ids from temp file.
test_add (test_calculator.TestCalculator) ... ok
test_divide (test_calculator.TestCalculator) ... ok
test_log (test_calculator.TestCalculator) ... ok
test_multiply (test_calculator.TestCalculator) ... ok
test_power (test_calculator.TestCalculator) ... ok
test_sqrt (test_calculator.TestCalculator) ... ok
test_subtract (test_calculator.TestCalculator) ... ok
-----
Ran 7 tests in 0.001s
OK
Finished running tests!
```

Running Tests for Workspace(s) /home/jankopec/LAB6_ZTP

- test_add
- test_divide
- test_log
- test_multiply
- test_power
- test_sqrt
- test_subtract

> 2 older results

Zadanie 2.

Należy napisać testy jednostkowe w wybranym języku, które pokrywają funkcje podanego kodu:

```
shopping_cart.py > ...
1 class ShoppingCart:
2     def __init__(self):
3         self.items = []
4
5     def add_item(self, name, price, quantity=1):
6         if price < 0 or quantity <= 0:
7             raise ValueError("Price and quantity must be positive.")
8         self.items.append({"name": name, "price": price, "quantity": quantity})
9
10    def remove_item(self, name):
11        for item in self.items:
12            if item["name"] == name:
13                self.items.remove(item)
14                return
15        raise ValueError("Item not found in the cart.")
16
17    def get_total_price(self):
18        return sum(item["price"] * item["quantity"] for item in self.items)
19
20    def clear_cart(self):
21        self.items = []
22
23    def get_items(self):
24        return self.items
```

Aby upewnić się, że testy pokrywają większość funkcjonalności, proszę skorzystać z narzędzi do analizy pokrycia kodu.

Implementacja [Python 3.8.10, VS Code, Coverage 7.6.1]

```

test_shopping_cart.py > TestShoppingCart > test_get_total_price
1  import unittest
2  from shopping_cart import ShoppingCart
3
4  class TestShoppingCart(unittest.TestCase):
5
6      def setUp(self):
7          self.cart = ShoppingCart()
8
9      def test_add_item(self):
10         self.cart.add_item("Laptop", 999.99, 1)
11         self.assertEqual(len(self.cart.get_items()), 1)
12         self.assertEqual(self.cart.get_items()[0]["name"], "Laptop")
13         self.assertEqual(self.cart.get_items()[0]["price"], 999.99)
14         self.assertEqual(self.cart.get_items()[0]["quantity"], 1)
15
16     def test_remove_item(self):
17         self.cart.add_item("Headphones", 150.75, 2)
18         self.cart.remove_item("Headphones")
19         self.assertEqual(len(self.cart.get_items()), 0)
20
21     def test_remove_item_not_found(self):
22         with self.assertRaises(ValueError):
23             self.cart.remove_item("Smartwatch")
24
25     def test_get_total_price(self):
26         self.cart.add_item("Laptop", 999.99, 1)
27         self.cart.add_item("Keyboard", 50.5, 3)
28         self.assertEqual(self.cart.get_total_price(), 1151.49)
29
30     def test_clear_cart(self):
31         self.cart.add_item("Tablet", 450.99, 1)
32         self.cart.add_item("Mouse", 25.5, 2)
33         self.cart.clear_cart()
34         self.assertEqual(len(self.cart.get_items()), 0)
35
36     def test_get_items(self):
37         self.cart.add_item("Tablet", 450.99, 1)
38         self.cart.add_item("Mouse", 25.5, 2)
39         items = self.cart.get_items()
40         self.assertEqual(len(items), 2)
41         self.assertEqual(items[0]["name"], "Tablet")
42         self.assertEqual(items[1]["name"], "Mouse")

```

Pobieram narzędzie coverage:

```

~/LAB6_ZTP$ pip3 install coverage
Successfully installed coverage-7.6.1

```

Uruchamiam testy:

```

jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ coverage run -m unittest test_shopping_cart.py
.....
-----
Ran 6 tests in 0.000s

OK

```

Generuję raport:

```

jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ coverage report -m
Name                               Stmts  Miss  Cover  Missing
-----
shopping_cart.py                   19      1   95%    7
test_shopping_cart.py              34      0  100%
-----
TOTAL                               53      1   98%

```

Z raportu wynika, że jedna linia kodu nie została pokryta testami.

Jest to linia 7:

```
7         raise ValueError("Price and quantity must be positive.")
```

Zatem dodaję 3 testy które sprawdzają czy program poprawnie wyrzuca wyjątek, gdy cena lub ilość jest niepoprawna:

```
44     def test_add_item_with_negative_price(self):
45         with self.assertRaises(ValueError):
46             self.cart.add_item("Laptop", -1000, 1)
47
48     def test_add_item_with_negative_quantity(self):
49         with self.assertRaises(ValueError):
50             self.cart.add_item("Laptop", 1000, -1)
51
52     def test_add_item_with_zero_quantity(self):
53         with self.assertRaises(ValueError):
54             self.cart.add_item("Laptop", 1000, 0)
```

Ponownie uruchamiam testy – tym razem pokrycie 100%:

```
jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ coverage run -m unittest test_shopping_cart.py
.....
-----
Ran 9 tests in 0.001s

OK
jankopec@DESKTOP-8PFJ905:~/LAB6_ZTP$ coverage report -m
Name                               Stmts  Miss  Cover   Missing
-----
shopping_cart.py                   19      0  100%
test_shopping_cart.py              43      0  100%
-----
TOTAL                             62      0  100%
```

3. Wnioski

W ramach laboratorium udało się zaimplementować prosty kalkulator oraz opracować odpowiednie testy jednostkowe dla każdej z funkcji. Zrealizowano podejście TDD, w którym testy były pisane przed implementacją funkcji, a każda funkcjonalność kalkulatora została przetestowana pod kątem poprawności. Przeprowadzono także analizę pokrycia kodu, przy pomocy narzędzia coverage. Udało się uzyskać 100% pokrycia testami dla kodu testowego oraz 100% dla implementacji koszyka zakupowego.