Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»
Отчет по лабораторной работе №4

Выполнил:
студент группы ИУ5-31Б
Бондаренко Денис Константинович

Подпись:_______________

Дата:_______________

Проверил:
преподаватель каф. ИУ5
Канев Антон Игоревич

Подпись:_____________

Дата:_____________

Москва, 2021 г.

Лабораторная работа №4
Описание задания

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. Вместо реализации паттерна Вы можете написать тесты для своей программы решения биквадратного уравнения. В этом случае, возможно, Вам потребуется доработать программу решения биквадратного уравнения, чтобы она была пригодна для модульного тестирования.
3. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

Текст программы

pattern.py

```python
from __future__ import annotations
from abc import ABC, abstractmethod


class TwixFactory(ABC):

    @abstractmethod
    def create_biscuit(self) -> AbstractBiscuit:
        pass

    @abstractmethod
    def create_caramel(self) -> AbstractCaramel:
        pass

    @abstractmethod
    def create_stick(self) -> AbstractStick:
        pass


class LeftFactory(TwixFactory):

    def create_biscuit(self) -> AbstractBiscuit:
        return LeftBiscuit()

    def create_caramel(self) -> AbstractCaramel:
        return LeftCaramel()

    def create_stick(self) -> AbstractStick:
        return LeftStick()


class RightFactory(TwixFactory):

    def create_biscuit(self) -> AbstractBiscuit:
        return RightBiscuit()
    def create_caramel(self) -> AbstractCaramel:
        return RightCaramel()

    def create_stick(self) -> AbstractStick:
        return RightStick()


class AbstractBiscuit(ABC):

    @abstractmethod
    def i_biscuit(self) -> str:
        pass


class LeftBiscuit(AbstractBiscuit):
    def i_biscuit(self) -> str:
        return "Left biscuit."


class RightBiscuit(AbstractBiscuit):
    def i_biscuit(self) -> str:
        return "Right biscuit."


class AbstractCaramel(ABC):

    @abstractmethod
    def i_caramel(self) -> None:
        pass

    @abstractmethod
    def side(self) -> None:
        pass

    @abstractmethod
    def on_the_biscuit(self, collaborator: AbstractBiscuit) -> None:

        pass


class LeftCaramel(AbstractCaramel):
    def i_caramel(self) -> str:
        return "Left caramel."

    def side(self) -> str:
        return "Left"

    def on_the_biscuit(self, collaborator: AbstractBiscuit) -> str:
        if isinstance(collaborator, RightBiscuit):
            return f"We are not the same."
        result = collaborator.i_biscuit()
        return f"Left caramel on the {result} We are the same."


class RightCaramel(AbstractCaramel):
    def i_caramel(self) -> str:
        return "Right caramel."

    def side(self) -> str:
        return "Right"

    def on_the_biscuit(self, collaborator: AbstractBiscuit):
        if isinstance(collaborator, LeftBiscuit):
            return f"We are not the same."
        result = collaborator.i_biscuit()
        return f"Right caramel on the {result} We are the same."


class AbstractStick(ABC):

    @abstractmethod
    def i_stick(self) -> str:
        pass

    def made_of(self):
        pass


class LeftStick(AbstractStick):
    def __init__(self):
        self.biscuit = LeftBiscuit()
        self.caramel = LeftCaramel()

    def i_stick(self) -> str:
        return f"{self.caramel.side()} stick."

    def made_of(self):
        return f"I'm {self.i_stick()} I'm made of
{self.caramel.on_the_biscuit(self.biscuit)}"


class RightStick(AbstractStick):
    def __init__(self):
        self.biscuit = RightBiscuit()
        self.caramel = RightCaramel()

    def i_stick(self) -> str:
        return f"{self.caramel.side()} stick."

    def made_of(self):
        return f"I'm {self.i_stick()} I'm made of
{self.caramel.on_the_biscuit(self.biscuit)}"


"""
def create_stick(biscuit, caramel):
    stick = [biscuit, caramel]
    return stick
"""


def client_code(factory: TwixFactory) -> None:

    biscuit = factory.create_biscuit()
    caramel = factory.create_caramel()
    stick = factory.create_stick()

    print(f"{biscuit.i_biscuit()}")
    print(f"{caramel.i_caramel()}")
    print(f"{caramel.on_the_biscuit(biscuit)}", end="\n")
    print(f"{stick.made_of()}", end="")


if __name__ == "__main__":

    print("Client: Testing client code with the left factory:")
    client_code(LeftFactory())

    print("\n")

    print("Client: Testing the same client code with the right factory:")
    client_code(RightFactory())

    print("\n")

    print("Testing the wrong components combination:")
    RB = RightBiscuit()
    LC = LeftCaramel()
    print(f"{LC.on_the_biscuit(RB)}", end="")
```

unit_test.py

```
import unittest
from pattern import *


class MyTest(unittest.TestCase):

    def setUp(self):
        self.rcaramel = RightCaramel()
        self.lcaramel = LeftCaramel()
        self.rbiscuit = RightBiscuit()
        self.lbiscuit = LeftBiscuit()
        self.rstick = RightStick()
        self.lstick = LeftStick()

    def test_same(self):
        self.assertEqual(self.lcaramel.on_the_biscuit(self.lbiscuit),
                         "Left caramel on the Left biscuit. We are the same.")

    def test_not_same_lcrb(self):
        self.assertEqual(self.lcaramel.on_the_biscuit(self.rbiscuit),
                         "We are not the same.")

    def test_not_same_rclb(self):
        self.assertEqual(self.rcaramel.on_the_biscuit(self.lbiscuit),
                         "We are not the same.")

    def test_rstick(self):
        self.assertEqual(self.rstick.made_of(), "I'm Right stick. I'm made of
Right caramel on the Right biscuit."
                                                " We are the same.")

    def test_lstick(self):
        self.assertEqual(self.lstick.made_of(), "I'm Left stick. I'm made of
Left caramel on the Left biscuit."
                                                " We are the same.")


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

my.feature

```
Feature: my
  Scenario: similarity check
```

```
    Given I have the components {left caramel} and {left biscuit}
    When I request for their similarity
    Then I expect the result to be {Left caramel on the Left biscuit. We are the
same.}

  Scenario: difference check
    Given I have the components {left caramel} and {right biscuit}
    When I request for their difference
    Then I expect the result to be {We are not the same.}

  Scenario: difference check 2
    Given I have the components {right caramel} and {left biscuit}
    When I request for their difference again
    Then I expect the result to be {We are not the same.} again

  Scenario: right stick check
    Given I have the {right stick}
    When I request for what it is made of
    Then I expect the result to be {I'm Right stick. I'm made of Right caramel
on the Right biscuit. We are the same.}

  Scenario: left stick check
    Given I have the {left stick}
    When I request for what it is made of again
    Then I expect the result to be {I'm Left stick. I'm made of Left caramel on
the Left biscuit. We are the same.}
```

my.py

```
from behave import *
from pattern import *


@given('I have the components {left caramel} and {left biscuit}')
def have_components(context, biscuit=LeftBiscuit(), caramel=LeftCaramel()):
    context.biscuit = biscuit
    context.caramel = caramel


@when('I request for their similarity')
def request_sim(context):
    context.result = context.caramel.on_the_biscuit(context.biscuit)


@then('I expect the result to be {Left caramel on the Left biscuit. We are the
same.}')
def expect_result(context):
    assert context.result == "Left caramel on the Left biscuit. We are the
same."


@given('I have the components {left caramel} and {right biscuit}')
def have_components(context, biscuit=RightBiscuit(), caramel=LeftCaramel()):
    context.biscuit = biscuit
    context.caramel = caramel


@when('I request for their difference')
def request_dif(context):
    context.result = context.caramel.on_the_biscuit(context.biscuit)
```

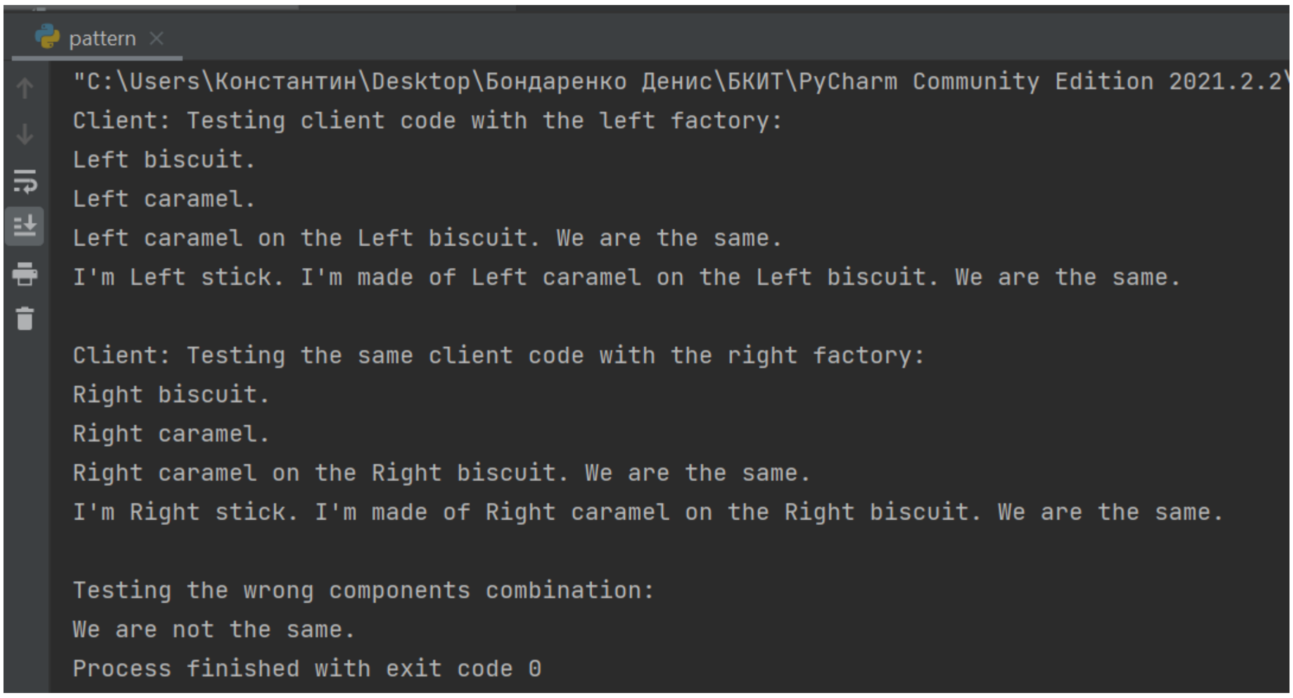
```python
@then('I expect the result to be {We are not the same.}')
def expect_result(context):
    assert context.result == "We are not the same."


@given('I have the components {right caramel} and {left biscuit}')
def have_components(context, biscuit=LeftBiscuit(), caramel=RightCaramel()):
    context.biscuit = biscuit
    context.caramel = caramel


@when('I request for their difference again')
def request_dif(context):
    context.result = context.caramel.on_the_biscuit(context.biscuit)


@then('I expect the result to be {We are not the same.} again')
def expect_result(context):
    assert context.result == "We are not the same."


@given('I have the {right stick}')
def have_stick(context, stick=RightStick()):
    context.stick = stick


@when('I request for what it is made of')
def request_made_of(context):
    context.result = context.stick.made_of()


@then('I expect the result to be {I\'m Right stick. I\'m made of Right caramel
on the Right biscuit. We are the same.}')
def expect_result(context):
    assert context.result == "I\'m Right stick. I\'m made of Right caramel on
the Right biscuit. We are the same."


@given('I have the {left stick}')
def have_stick(context, stick=LeftStick()):
    context.stick = stick


@when('I request for what it is made of again')
def request_made_of(context):
    context.result = context.stick.made_of()


@then('I expect the result to be {I\'m Left stick. I\'m made of Left caramel on
the Left biscuit. We are the same.}')
def expect_result(context):
    assert context.result == "I\'m Left stick. I\'m made of Left caramel on the
Left biscuit. We are the same."
```

mock.py

```
from pattern import *
from unittest import TestCase
from unittest.mock import patch


class MockTest(TestCase):
    @patch('pattern.LeftStick.made_of', return_value="I'm Left stick. I'm made
of Left caramel on the Left biscuit."
                                                    " We are the same.")
    def test_made_of(self, made_of):
        self.stick = LeftStick()
        self.assertEqual(self.stick.made_of(), "I'm Left stick. I'm made of Left
caramel on the Left biscuit."
                                              " We are the same.")

    @patch('pattern.RightStick.i_stick', return_value="Right stick.")
    def test_made_of2(self, i_stick):
        self.stick = RightStick()
        self.assertEqual(self.stick.made_of(), "I'm Right stick. I'm made of
Right caramel on the Right biscuit."
                                              " We are the same.")


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

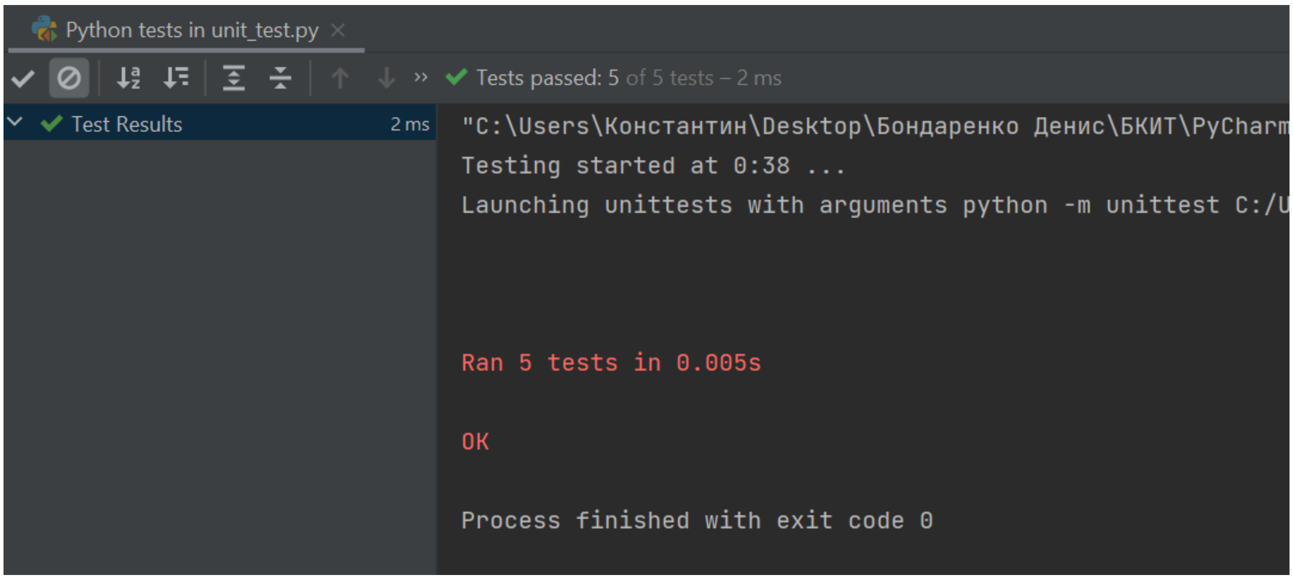
Экранные формы с примерами выполнения программы

pattern.py



```
"C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ\PyCharm Community Edition 2021.2.2\
Client: Testing client code with the left factory:
Left biscuit.
Left caramel.
Left caramel on the Left biscuit. We are the same.
I'm Left stick. I'm made of Left caramel on the Left biscuit. We are the same.

Client: Testing the same client code with the right factory:
Right biscuit.
Right caramel.
Right caramel on the Right biscuit. We are the same.
I'm Right stick. I'm made of Right caramel on the Right biscuit. We are the same.

Testing the wrong components combination:
We are not the same.
Process finished with exit code 0
```

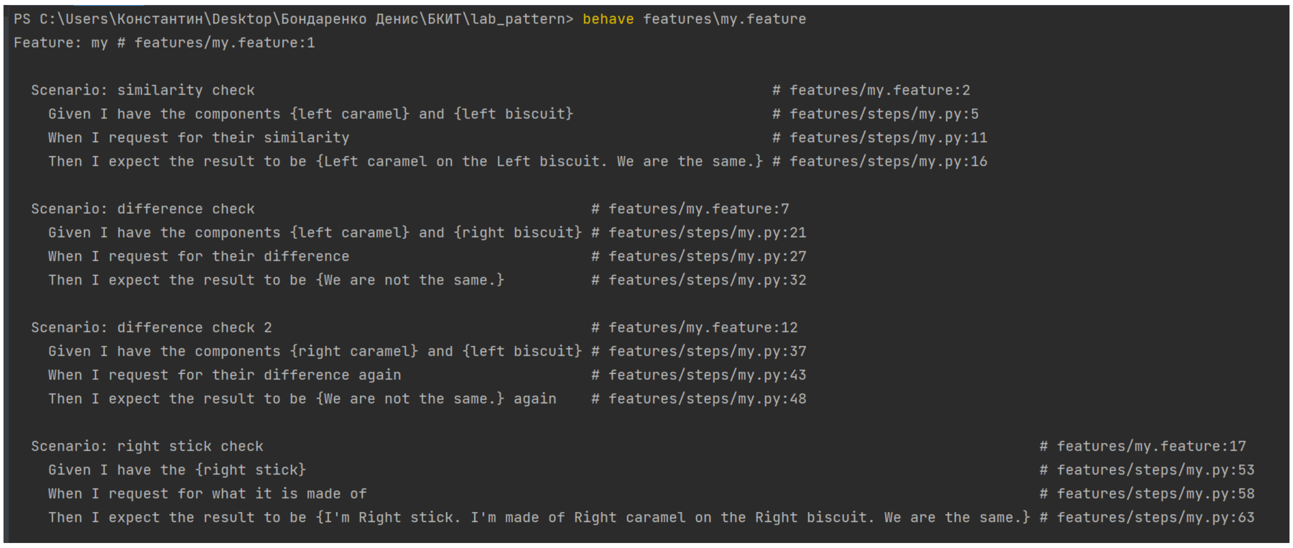
Python tests in unit_test.py



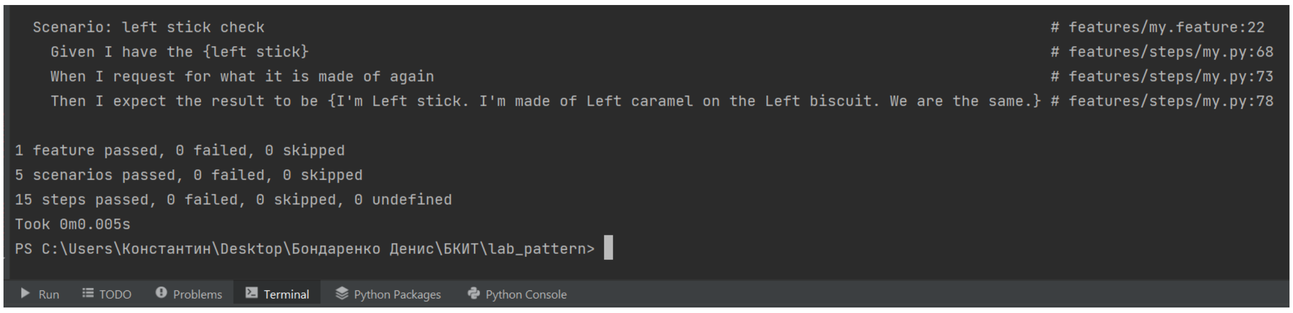
Python tests in unit_test.py
Tests passed: 5 of 5 tests – 2 ms
Test Results 2 ms

```
"C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ\PyCharm
Testing started at 0:38 ...
Launching unittests with arguments python -m unittest C:/U



Ran 5 tests in 0.005s

OK

Process finished with exit code 0
```

my.py и my.feature

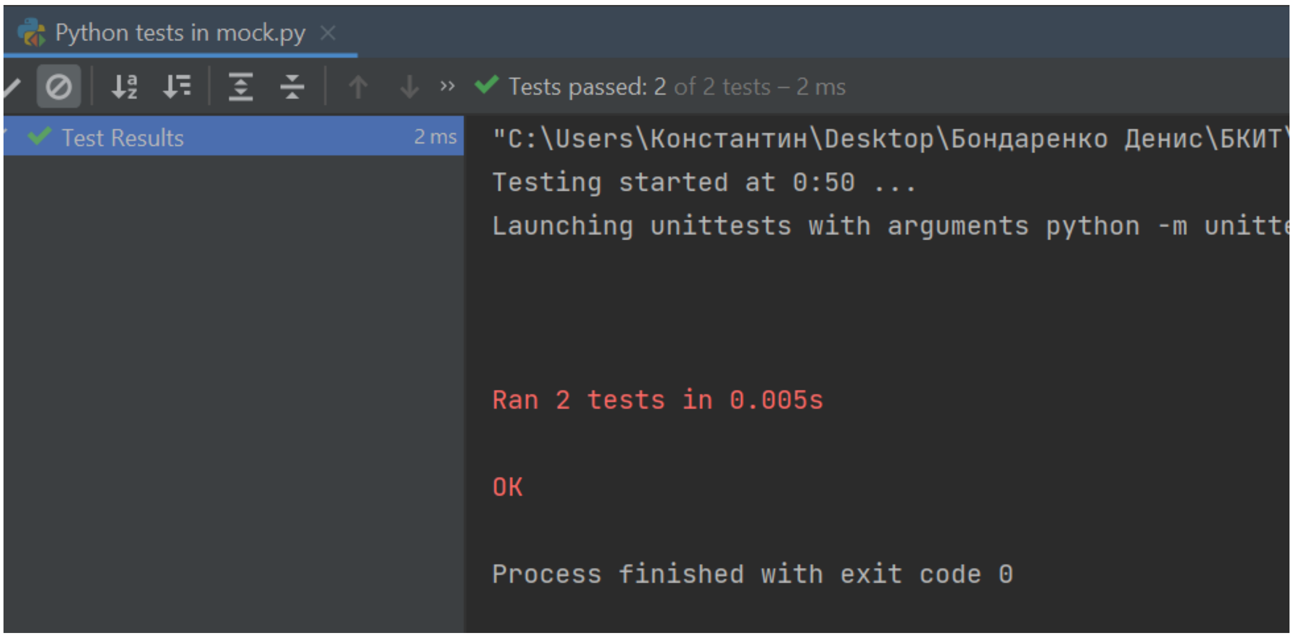


```
PS C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ\lab_pattern> behave features\my.feature
Feature: my # features/my.feature:1

  Scenario: similarity check                                                        # features/my.feature:2
    Given I have the components {left caramel} and {left biscuit}                   # features/steps/my.py:5
    When I request for their similarity                                             # features/steps/my.py:11
    Then I expect the result to be {Left caramel on the Left biscuit. We are the same.} # features/steps/my.py:16

  Scenario: difference check                                        # features/my.feature:7
    Given I have the components {left caramel} and {right biscuit} # features/steps/my.py:21
    When I request for their difference                             # features/steps/my.py:27
    Then I expect the result to be {We are not the same.}           # features/steps/my.py:32

  Scenario: difference check 2                                      # features/my.feature:12
    Given I have the components {right caramel} and {left biscuit} # features/steps/my.py:37
    When I request for their difference again                       # features/steps/my.py:43
    Then I expect the result to be {We are not the same.} again     # features/steps/my.py:48

  Scenario: right stick check                                                                             # features/my.feature:17
    Given I have the {right stick}                                                                        # features/steps/my.py:53
    When I request for what it is made of                                                                 # features/steps/my.py:58
    Then I expect the result to be {I'm Right stick. I'm made of Right caramel on the Right biscuit. We are the same.} # features/steps/my.py:63
```



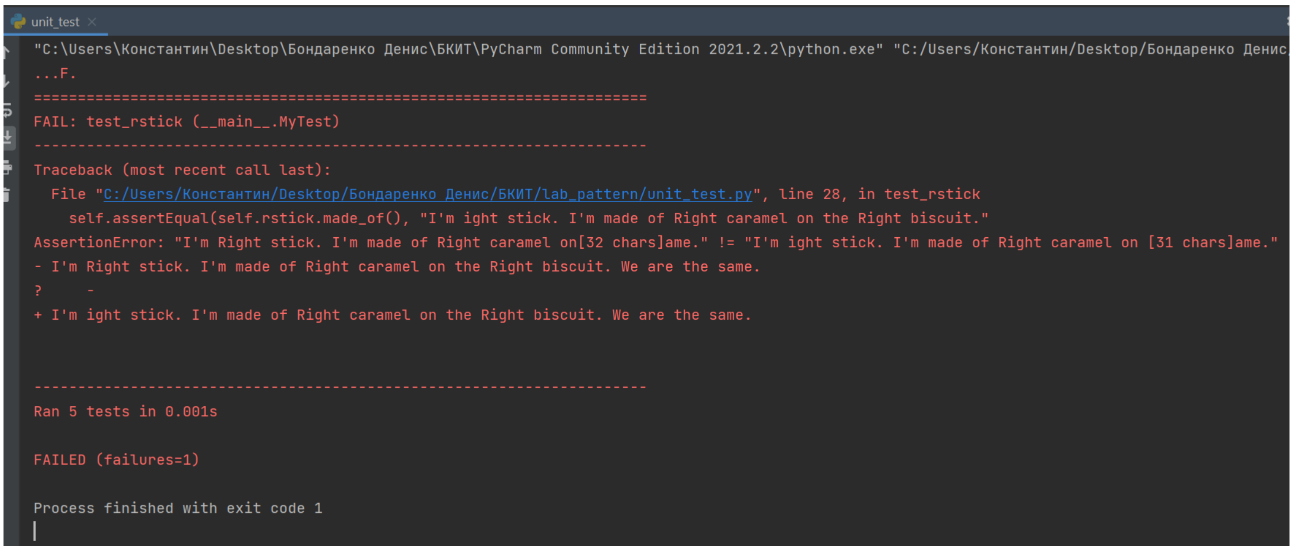
```
  Scenario: left stick check                                                                            # features/my.feature:22
    Given I have the {left stick}                                                                       # features/steps/my.py:68
    When I request for what it is made of again                                                         # features/steps/my.py:73
    Then I expect the result to be {I'm Left stick. I'm made of Left caramel on the Left biscuit. We are the same.} # features/steps/my.py:78

1 feature passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
15 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.005s
PS C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ\lab_pattern>
```

Python tests in mock.py



```
"C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ
Testing started at 0:50 ...
Launching unittests with arguments python -m unitte



Ran 2 tests in 0.005s

OK

Process finished with exit code 0
```

Python tests in unit_test.py при изменении ожидаемого результата в четвертом тесте



```
"C:\Users\Константин\Desktop\Бондаренко Денис\БКИТ\PyCharm Community Edition 2021.2.2\python.exe" "C:/Users/Константин/Desktop/Бондаренко Денис
...F.
======================================================================
FAIL: test_rstick (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Константин/Desktop/Бондаренко Денис/БКИТ/lab_pattern/unit_test.py", line 28, in test_rstick
    self.assertEqual(self.rstick.made_of(), "I'm ight stick. I'm made of Right caramel on the Right biscuit."
AssertionError: "I'm Right stick. I'm made of Right caramel on[32 chars]ame." != "I'm ight stick. I'm made of Right caramel on [31 chars]ame."
- I'm Right stick. I'm made of Right caramel on the Right biscuit. We are the same.
?     -
+ I'm ight stick. I'm made of Right caramel on the Right biscuit. We are the same.


----------------------------------------------------------------------
Ran 5 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 1
```