

**Министр науки и высшего образования Российской
Федерации**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет
ИТМО»**



**Факультет информационных технологий и
программирования**

Лабораторная работа №4

Написание Unit тестов. Github Actions.

**Выполнил студент группы № М3113
Рифад МД Абу Хуссаин**

Проверил:
Жуйков Артём Сергеевич

Санкт-Петербург

Part 01

1.

Создаю новую ветку, в которой впоследствии и буду вести всю работу

```
hussa@DESKTOP-P5GJON6 MINGW64 /c/ITM0/First Semester/Soft Skill/Lab04 (feat/tes
ts)
● $ git branch
    develop
    documentation
  * feat/tests
    main
    new_features_336131
```

2.

Функции для вычисления площади и периметра круга, квадрата и треугольника.
Функция `calc` рассчитывает эти значения в зависимости от типа фигуры и аргументов,
с проверкой ошибок.

```
import math

def area_circle(r):
    """Вычисление площади круга"""
    return math.pi * r * r

def perimeter_circle(r):
    """Вычисление периметра круга"""
    return 2 * math.pi * r

def area_square(a):
    """Вычисление площади квадрата"""
    return a * a

def perimeter_square(a):
    """Вычисление периметра квадрата"""
    return 4 * a

def area_triangle(a, b, c):
    """Вычисление площади треугольника по формуле Герона"""
    s = (a + b + c) / 2
    return math.sqrt(s * (s - a) * (s - b) * (s - c))

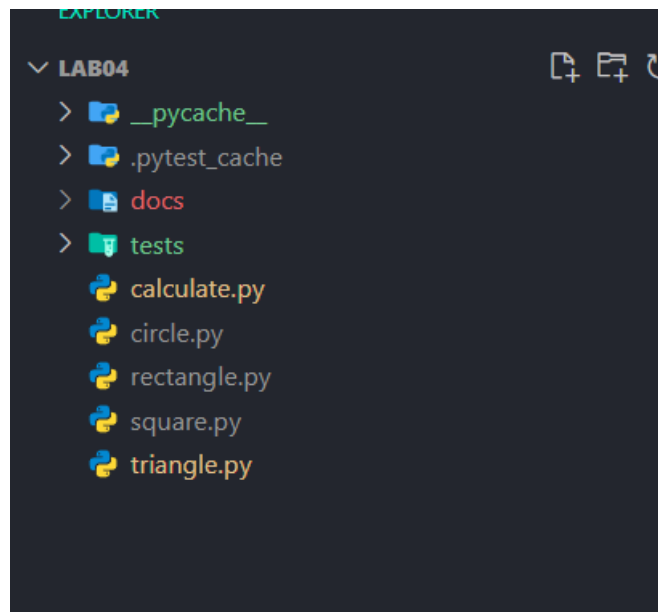
def perimeter_triangle(a, b, c):
```

```

    """Вычисление периметра треугольника"""
    return a + b + c

def calc(shape, *args):
    """Функция для вычисления площади или периметра заданной фигуры"""
    if shape == "circle":
        if len(args) != 1:
            raise ValueError("Круг должен иметь только один аргумент (радиус)")
        r = args[0]
        return {"area": area_circle(r), "perimeter": perimeter_circle(r)}
    elif shape == "square":
        if len(args) != 1:
            raise ValueError("Квадрат должен иметь только один аргумент (сторону)")
        a = args[0]
        return {"area": area_square(a), "perimeter": perimeter_square(a)}
    elif shape == "triangle":
        if len(args) != 3:
            raise ValueError("Треугольник должен иметь три аргумента (стороны)")
        a, b, c = args
        if a + b <= c or a + c <= b or b + c <= a:
            raise ValueError("Сумма двух сторон треугольника должна быть больше третьей")
        return {"area": area_triangle(a, b, c), "perimeter": perimeter_triangle(a, b, c)}
    else:
        raise ValueError(f"Неизвестная фигура: {shape}")

```



4.

Тесты с использованием pytest для вычисления площади и периметра круга, квадрата, треугольника и проверки функции `calc`, включая обработку ошибок для некорректных входных данных.

```
import pytest
import sys
import os
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

from calculate import (
    area_circle, perimeter_circle,
    area_square, perimeter_square,
    area_triangle, perimeter_triangle,
    calc
)

# Тесты для круга
def test_area_circle():
    assert area_circle(1) == pytest.approx(3.14159, 0.0001)
    assert area_circle(0) == 0
    assert area_circle(2) == pytest.approx(12.56637, 0.0001)
```

```
def test_perimeter_circle():
    assert perimeter_circle(1) == pytest.approx(6.28318, 0.0001)
    assert perimeter_circle(0) == 0
    assert perimeter_circle(2) == pytest.approx(12.56637, 0.0001)

# Тесты для квадрата
def test_area_square():
    assert area_square(2) == 4
    assert area_square(0) == 0
    assert area_square(5) == 25

def test_perimeter_square():
    assert perimeter_square(2) == 8
    assert perimeter_square(0) == 0
    assert perimeter_square(5) == 20

# Тесты для треугольника
def test_area_triangle():
    assert area_triangle(3, 4, 5) == pytest.approx(6, 0.0001)
    assert area_triangle(6, 8, 10) == pytest.approx(24, 0.0001)

def test_perimeter_triangle():
    assert perimeter_triangle(3, 4, 5) == 12
    assert perimeter_triangle(6, 8, 10) == 24

# Тесты для функции calc
def test_calc_circle():
    result = calc("circle", 1)
    assert result["area"] == pytest.approx(3.14159, 0.0001)
    assert result["perimeter"] == pytest.approx(6.28318, 0.0001)

def test_calc_square():
    result = calc("square", 4)
    assert result["area"] == 16
    assert result["perimeter"] == 16

def test_calc_triangle():
    result = calc("triangle", 3, 4, 5)
    assert result["area"] == pytest.approx(6, 0.0001)
    assert result["perimeter"] == 12

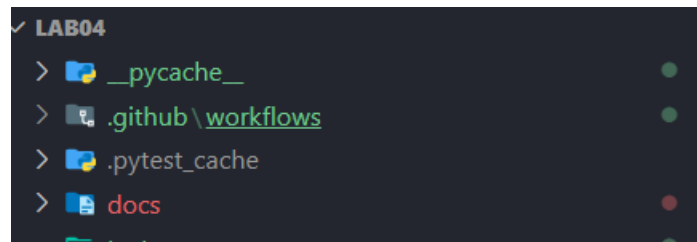
def test_calc_invalid_args():
    with pytest.raises(ValueError):
        calc("circle", 1, 2)
```

```
with pytest.raises(ValueError):  
    calc("triangle", 1, 2)  
with pytest.raises(ValueError):  
    calc("unknown_shape", 1)
```

Part 02

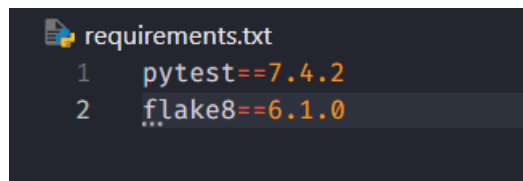
1.

Создайте файл воркфлоу



2.

Также нужно для определения зависимостей в проекте использовать отдельный файл **requirements.txt**



3.

Наконец создаю отдельную папку в проекте для различных рабочих пространств (workflows) и также создаю в ней файл с новым workflows для новых событий.

```

1  name: Python test
2
3  on:
4    push:
5      branches:
6        - master
7        - release/*
8    pull_request:
9      branches:
10       - master
11       - release/*
12       - develop
13
14  jobs:
15    test:
16      runs-on: ubuntu-latest
17
18      env:
19        FULL_NAME: "Hussain Rifad"
20        GROUP_NUMBER: "M3113"
21
22      steps:
23        # 1. Check out the repository code
24        - name: Checkout code
25          uses: actions/checkout@v3
26
27        # 2. Print personal info
28        - name: Print Personal Info
29          run: |
30            echo "Author: ${ env.NAME }"
31            echo "Group: ${ env.GROUP }"
32            echo "GitHub username: ${ github.actor }"
33
34        # 3. Set up Python 3.13
35        - name: Setup Python
36          uses: actions/setup-python@v4
37          with:
38            python-version: 3.13
39
40        # 4. Install dependencies
41        - name: Install Dependencies
42          run: |
43            python -m pip install --upgrade pip
44            pip install -r requirements.txt
45
46        # 5. Run unit tests
47        - name: Run Tests
48          run: pytest
49
50        # 6. Run linter for code style
51        - name: Lint Code
52          run: flake8 . --count --max-line-length=88 --show-source --statistics
53

```

4.

Коммит для проделанной работы

```
$ git commit -m"created new github workflow and added requirements.txt file"
[feat/tests da29af1] created new github workflow and added requirements.txt file
10 files changed, 206 insertions(+), 7 deletions(-)
```

Part 03


1. Push to github

```
$ git push --set-upstream origin feat/tests
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (17/17), 9.31 KiB | 1.33 MiB/s, done.
Total 17 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
```


2.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).


 base repository: smartigaorg/geometric_lib

base: main

 head repository: hussainrifad/geometric_lib

compare: feat/tests

✓ **Able to merge.** These branches can be automatically merged.



















**Add a title**

Feat/tests



Helpful resources
[GitHub Community Guidelines](#)

Add a description

WritePreview

H B I                  

Create new workflow and this repo is for testing. @Mindtable

 Markdown is supported  Paste, drop, or click to add files