**Python/Pytest**



**Задание 1: Начало работы с тестами**

Реализуйте кейс тестирования класса Calculation() который из себя представляет четыре функции:

def sum(self, a, b)

def sub(self, a, b)

def multiply(self, a, b)

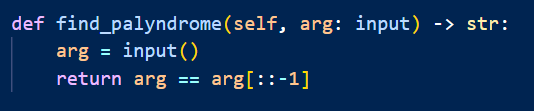
def div(self, a, b)

Каждая из функций должна иметь свой тестовый случай и все из них должны успешного его пройти (4/4 PASSED).

Реализуйте для данного кейса отдельный маркер с названием calc.

Так же добавьте в тестовом файле функцию test\_invisible() в котором в качестве утверждения будет 33 == 5, используйте маркер skip для того чтобы пропустить неверное утверждение (SKIPPED)  
На выходе у вас должно получиться (4 PASSED, 1 SKIPPED)

**Задание 2. Описание тестового случая**

****

Дан вышеописанный код в виде функции, реализуйте тестовый кейс для функции где происходит ввод строки на проверку является ли она палиндромом или нет. В случае если введенная вами строка True тест должен быть пройден (passed), если False – failed.

**Задание 3. Тестирование классов**

Реализуйте класс Text\_Greeting() внутри которого создайте следующие функции:

def greeting(self, name) – который будет принимать в себя строку (например return f”My name {name}’) и сравнивать полученное имя с ожидаемым в тестовом кейсе, реализуйте минимум три теста одного и того же кейса (через parametrize), где 2 – passed, 1 – failed.

def empty\_string(self, arg) – функция которая будет проверять пустая строка или нет, тест будет пройден если len(arg) > 0, если же нет – failed.

def register\_check(self, arg) – данная функция должна проверять строку на капитализацию символов (регистр), все что не будет написано в тестовом кейсе капсом (upper) будет failed.

def long\_check(self, arg) – функция, проверяющая длину строки с условием if len < 10, если < 10 – passed, если > 10 – failed.

def string\_type(self, arg) – функция должна принимать в себя аргумент и тестовый кейс правильный только при условии, если мы ввели любой тип данных кроме string (строки). Вводим строку – failed, любой другой тип данных – passed.

Тестовый кейс будет иметь такой вид:

Каждый тест будет иметь марку @greeting

def test\_greeting()

def test\_empty()

def test\_register()

def test\_long()

def test\_type()

Так же создадим новую функцию с заведомо неправильными данными (например ‘Aboba’ == ‘Anton’) и пропустим ее (марка skip)

def test\_empty() – skipped

**Задание 4. Тестирование функций**

Для начала работы с этим заданием создадим два новых файла:

pow.py и test\_pow.py (либо pow\_test.py)

В файле pow.py создадим несколько функций:

def pow2(a) где а \* а

def pow2\_math(a) где а в степени 2 (через библиотеку math)

def pow3\_math(a) где а в степени 3 (через библиотеку math)

def fibonacci(n) где нужно найти число Фибоначчи

В файле test\_pow у нас будут следующие функции:

Оба теста должны иметь марку pow2

def test\_pow2() - должен быть passed (возводя в степень)

def test\_pow2\_math() – должен быть passed (возводя в степень)

Следующие функции имеют марку pow3

def test\_pow3\_math() должен быть passed (возводя в степень)

def test\_fibonacci() - должен быть passed (проверяя число Фибоначчи)

def test\_notpow() – должен быть skipped, заранее неправильно поставлено условие (xfailed)

Итого 4 passed, 1 skipped, при этом они должны иметь разные марки для запуска.

