```
python -m venv testy_venv
testy_venv\Scripts\Activate.ps1
ały do zajec> python -m pip install pytest
```

- Tworzymy 3 pliki (funkcje.py, test_with_unittest.py, test_with_pytest.py)

funkcje.py

```python
def add(a,b):
    return a+b
```

test_with_unittest.py

```python
import funkcje
import unittest

# - Klasa powinna dziedziczyć po unittest.TestCase.
# - Metody testowe muszą mieć nazwę zaczynającą się od 'test' (np. test_add),
# inaczej nie zostaną uruchomione przy automatycznym wykrywaniu.

class Test_add(unittest.TestCase):
    def test_add(self):
        self.assertEqual(funkcje.add(2,4), 6)
        self.assertNotEqual(funkcje.add(2,4), 5)

# uruchom tylko jeśli bezposrednio startuje ten plik
if __name__ == '__main__':
    # uruchomienie runnera testów (wyszukuje testy i uruchamia)
    unittest.main()

# URUCHAMIANY TESTY
# - następnie zmieniamy returna w funckji na odejmowanie
#   i patrzymy jaki będzie wynik testów (FAIL)
#   *zwarac dokładnie który assert nie przeszedł
```

test_with_pytest.py

```python
import funkcje
import pytest
# - Metody testowe muszą mieć nazwę zaczynającą się od 'test' (np. test_add),
# inaczej nie zostaną uruchomione przy automatycznym wykrywaniu.

def test_add():
    assert funkcje.add(2,4) == 6
    assert not funkcje.add(2,3) == 6

# URUCHOMINIE: python -m <nazwa_modułu> <nazwa_programu>
# python -m pytest test_with_pytest.py
# python -m pytest test_with_pytest.py -v (więcej informacji)

# - następnie zmieniamy returna w funckji na odejmowanie
#   i patrzymy jaki będzie wynik testów (FAIL)
#   *zwarac dokładnie który assert nie przeszedł
```

Zadanie 1. Palindrom

Napisz funkcję sprawdzającą czy dane słowo jest palindromem (palindrom to słowo które pisane od przodu i od tyłu jest identyczne).
Napisz testy, które sprawdzą, czy słowa “kamilslimak” i “ala” są palindromami, a słowa “wiadro” i “kamyk” nimi nie są.

Funckje.py

```
def if_palindrom(word):
    return word == word[::-1]
```

Test_with_unittest.py

```
class Test_if_palindrom(unittest.TestCase):
    def test_palindrom(self):
        self.assertTrue(funkcje.if_palindrom("kamilslimak"))
        self.assertEqual(funkcje.if_palindrom("ala"), True)

    def test_not_palindrom(self):
        self.assertFalse(funkcje.if_palindrom("wiadro"))
        self.assertEqual(funkcje.if_palindrom("kamyk"), False)
```

Test_with_pytest.py

```
def test_if_palindrom():
    assert funkcje.if_palindrom("kamilslimak")
    assert funkcje.if_palindrom("ala")
    assert not funkcje.if_palindrom("wiadro")
    assert not funkcje.if_palindrom("kamyk")
```

Zadanie 2. Boki trójkąta
Napisz program, który sprawdzi, czy można utworzyć trójkąt z boków o podanej długości. Aby utworzyć trójkąt, suma dwóch krótszych boków musi być dłuższa niż najdłuższy bok.
Należy przetestować sytuację, w której zachodzi prawidłowy stosunek między bokami (2,3,4) sytuację, w której najdłuższy bok jest dłuższy od sumy dwóch pozostałych boków (1,1,2) i sytuację w której jeden z boków ma długość niedodatnią (0,-1,1)c

Funckje.py

```python
def if_valid_triangle(a, b, c):
    if min(a, b, c) <= 0:
        return False
    longest = max(a, b, c)
    return longest < (a + b + c - longest)
```

Test_with_unittest.py

```python
class TestIfValidTriangle(unittest.TestCase):
    def test_valid_triangle(self):
        self.assertEqual(funkcje.if_valid_triangle(2,3,4), True)
        self.assertEqual(funkcje.if_valid_triangle(5,9,5), True)

    def test_not_valid_triangle(self):
        self.assertEqual(funkcje.if_valid_triangle(1,1,2), False)
        self.assertEqual(funkcje.if_valid_triangle(-2,0,1), False)
```

Test_with_pytest.py

```python
def test_valid_triangle():
    assert funkcje.if_valid_triangle(2,3,4)
    assert funkcje.if_valid_triangle(5,9,5)

def test_invalid_triangle():
    assert not funkcje.if_valid_triangle(1,1,2)
    assert not funkcje.if_valid_triangle(-2,0,1)
```

```python
# Mockować = w teście zastąpić prawdziwą zależność (np. API, bazę, print) czymś kontrolowanym:
# żeby test był szybki
# powtarzalny (zawsze ten sam wynik)
# i żeby nie wymagał internetu / plików / bazy
```

Zadanie 3. Dzielenie
Napisz program dzielący dwie liczby. Program powinien wyświetlić wynik dzielenia, a w przypadku dzielenia przez 0 komunikat “Nie dziel przez 0”. (Te testy zrobimy wspólnie)

Funckje.py

```python
def divide(a,b): #(jedynym efektem działania tej funkcji jest tekst w konsoli)
    try:
        print(a/b)
    except:
        print("Nie dziel przez 0")

# ❌ nie zwraca wartości (return), ☑ tylko coś wypisuje na ekran
# 👉 Skoro funkcja nie zwraca wyniku, to nie możemy zrobić: assert divide(4,2) == 2
# bo divide(4,2) zwraca None.

# - dla (4,2) → "2.0\n"
# - dla (4,0) → "Nie dziel przez 0\n"

# musimy:
# 👉 przechwycić to, co trafiłoby do konsoli
# 👉 zapisać to do zmiennej
# 👉 porównać z oczekiwanym tekstem
```

Test_with_unittest.py

```python
from unittest.mock import patch # narzędzie do tym. podmiany czegoś w kodzie
from io import StringIO # obiekt zachowujący się jak plik/console, ale zapisuje tekst do pamięci

class TestDivideFunction(unittest.TestCase):
    # Na czas testu podmieniamy konsolę (miejsce, gdzie trafia print) na bufor w pamięci,
    # żeby móc sprawdzić, co zostało wydrukowane.
    @patch('sys.stdout', new_callable=StringIO)
    def test_divide(self, mock_stdout):
        funkcje.divide(4,2)
        self.assertEqual(mock_stdout.getvalue(), '2.0\n')

    @patch('sys.stdout', new_callable=StringIO)
    def test_divide_exception(self, mock_stdout):
        funkcje.divide(4,0)
        self.assertEqual(mock_stdout.getvalue(), 'Nie dziel przez 0\n')
```

Test_with_pytest.py

```python
def test_divide(capsys): # capsys przechwytuje ten print i zapisuje go “do pamięci”
    funkcje.divide(4,2)
#   capsys.readouterr() odczytuje przechwycony tekst:
#       - out = to, co poszło na stdout (printy)
#       - err = to, co poszło na stderr (błędy)
    out, err = capsys.readouterr()
    assert out == '2.0\n'

def test_divide_exception(capsys):
    funkcje.divide(4,0)
    out, err = capsys.readouterr()
    assert out == 'Nie dziel przez 0\n'
```

Zadanie 4. Sortowanie
Posortuj listę stringów według długości występujących w niej stringów.
Funkcja sort pozwala na ustalenie klucza według którego odbywa się sortowanie, w tym wypadku będzie to długość czyli len.
(Te testy zrobimy wspólnie)

Funckje.py

```
def sort_strings_by_length(list_of_strings):
    list_of_strings.sort(key=len)
    return list_of_strings
```

Test_with_unittest.py

```
class TestSortList(unittest.TestCase):
    def test_sorted_list(self):
        self.assertEqual(funkcje.sort_strings_by_length(['kot','pies','dziobak']),['kot','
pies','dziobak'])

    def test_reverse_list(self):
        self.assertEqual(funkcje.sort_strings_by_length(['dziobak','pies','kot']),['kot','
pies','dziobak'])

    def test_empty_list(self):
        self.assertEqual(funkcje.sort_strings_by_length([]), [])

    def test_mixed_list(self):
        self.assertEqual(funkcje.sort_strings_by_length(['kot','dziobak','pies']),['kot','
pies','dziobak'])
```

Test_with_pytest.py

```
import pytest
# 🔎 Co robi @pytest.mark.parametrize?
# To dekorator, który mówi:
#     - Uruchom tę samą funkcję testową kilka razy,
#     - za każdym razem z innymi danymi wejściowymi.

@pytest.mark.parametrize(
    "strings, expected",
    [
        (['kot', 'dziobak', 'pies'], ['kot', 'pies', 'dziobak']),
        (["minecraft", "roblox", "cs"], ['cs', 'roblox', 'minecraft'])
    ])

def test_sort_strings_by_len(strings, expected):
    assert funkcje.sort_strings_by_length(strings) == expected
```

Zadanie 5. Liczby pierwsze
Napisz program sprawdzający, czy liczba jest liczbą pierwszą. Liczba pierwsza to taka liczba naturalna, która dzieli się tylko przez 1 i samą siebie. Liczby 0 i 1 nie są liczbami pierwszymi.
Jeżeli liczba jest mniejsza od 2 wiemy że nie jest pierwsza - wynika to z definicji.
Następnie w pętli sprawdzamy czy znajdziemy dzielnik podanej liczby.
Jeżeli tak - wiemy, że liczba nie jest pierwsza,
jeżeli nie - liczba jest pierwsza, ponieważ nie ma żadnego innego dzielnika niż 1 i samą siebie.

Funckje.py

```python
import math
def if_prime(number):
    if number < 2:
        return False
    for i in range(2, int(math.sqrt(number)+1)):
        if number % i == 0:
            return False
    return True

# pierwiastek - największym możliwym dzielnikiem nieposiadającym jeszcze
# pary, każdy większy dzielnik jest już sparowany z dzielnikiem mniejszym
```

Test_with_unittest.py

```python
class TestIfPrime(unittest.TestCase):
    def test_is_prime(self):
        self.assertEqual(funkcje.if_prime(5), True)

    def test_is_not_prime(self):
        self.assertEqual(funkcje.if_prime(4), False)

    def test_1(self):
        self.assertEqual(funkcje.if_prime(1), False)

    def test_0(self):
        self.assertEqual(funkcje.if_prime(0), False)

    def test_negative(self):
        self.assertEqual(funkcje.if_prime(-1), False)
```

Test_with_pytest.py

```python
def test_prime_number():
    assert funkcje.if_prime(5)
    assert not funkcje.if_prime(4)
    assert not funkcje.if_prime(1)
    assert not funkcje.if_prime(0)
    assert not funkcje.if_prime(-1)
```

Zadanie 6. Średnia z listy
Napisz program (bez korzystania z funkcji mean) który policzy średnią wartość z listy. Program powinien zwrócić -1 jeżeli lista jest pusta oraz -2 jeżeli w liście znajdują się wartości nie będące liczbami.

Funckje.py

```
def calculate_mean(list_of_int):
    try:
        if len(list_of_int) == 0:
            return -1
        return sum(list_of_int) / len(list_of_int)
    except:
        return -2
```

Test_with_unittest.py

```
class TestCalculateMean(unittest.TestCase):
    def test_list(self):
        self.assertEqual(funkcje.calculate_mean([1,4,7]), 4)
        self.assertEqual(funkcje.calculate_mean([1,5,'kot']), -2)

    def test_empty_list(self):
        self.assertEqual(funkcje.calculate_mean([]), -1)
```

Test_with_pytest.py

```
def test_calculate_mean():
    assert funkcje.calculate_mean([1,3,8]) == 4

def test_calculate_mean_empty_list():
    assert funkcje.calculate_mean([]) == -1

def test_calculate_mean_incorect_data():
    assert funkcje.calculate_mean([1,2,'kot']) == -2
```