SPRAWOZDANIE

Imię i nazwisko: Jakub Kęsik

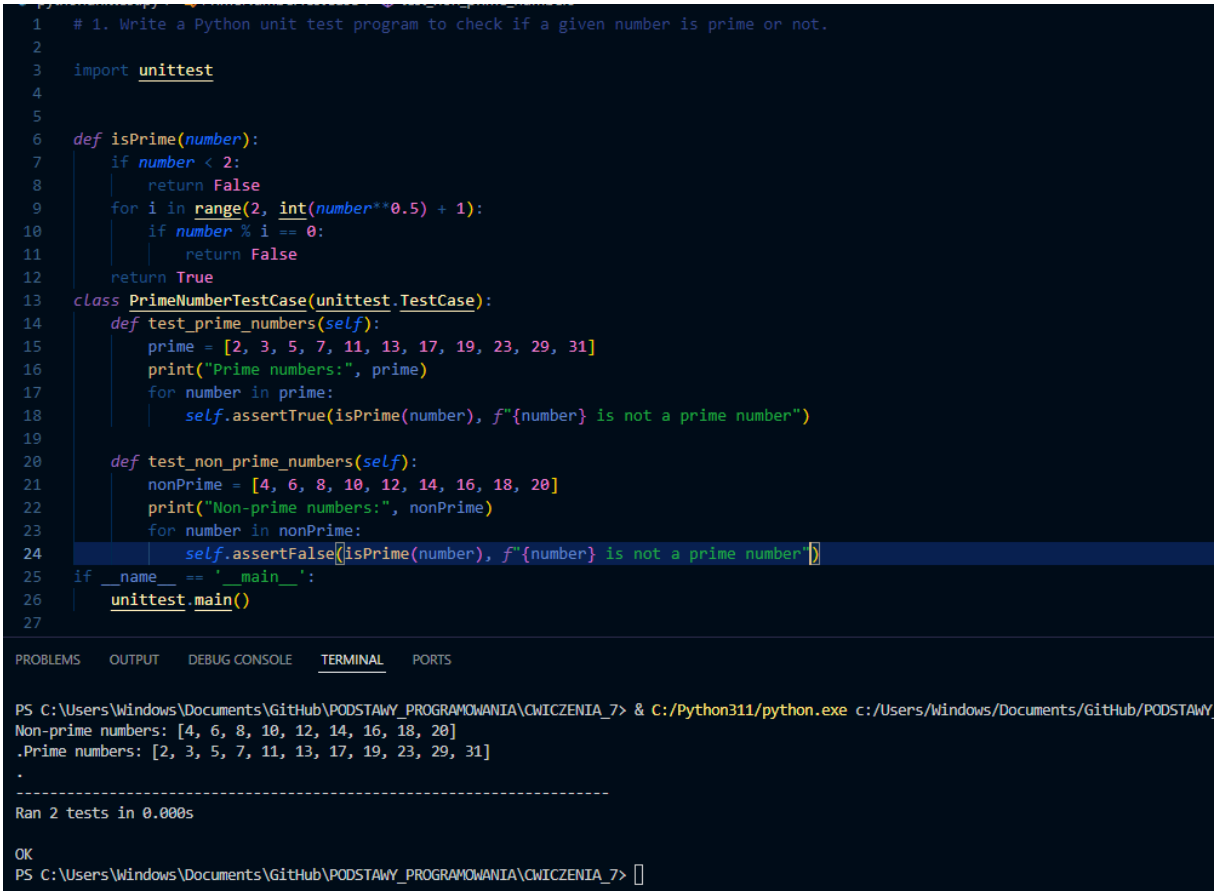
Nr albumu: 163040

Github: https://github.com/KasniQ/PODSTAWY_PROGRAMOWANIA

Python Unit Test

(pythonunittest.py)

Testy jednostkowe w Pythonie są realizowane przy użyciu modułu ‘unittest’, który pozwala na tworzenie, organizowanie i uruchamianie testów w celu weryfikacji poprawności kodu. Umożliwiają one sprawdzanie poszczególnych funkcji i metod, zapewniając, że działają one zgodnie z oczekiwaniami. Oferuje on różne metody asercji, które pomagają porównywać wyniki działania funkcji z przewidywanymi wartościami.



```python
# 1. Write a Python unit test program to check if a given number is prime or not.

import unittest


def isPrime(number):
    if number < 2:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True
class PrimeNumberTestCase(unittest.TestCase):
    def test_prime_numbers(self):
        prime = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
        print("Prime numbers:", prime)
        for number in prime:
            self.assertTrue(isPrime(number), f"{number} is not a prime number")

    def test_non_prime_numbers(self):
        nonPrime = [4, 6, 8, 10, 12, 14, 16, 18, 20]
        print("Non-prime numbers:", nonPrime)
        for number in nonPrime:
            self.assertFalse(isPrime(number), f"{number} is not a prime number")
if __name__ == '__main__':
    unittest.main()

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY
Non-prime numbers: [4, 6, 8, 10, 12, 14, 16, 18, 20]
.Prime numbers: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
.
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 2. Write a Python unit test program to check if a list is sorted in ascending order.

def ascendingOrderTest(myList):
    return all(myList[i] <= myList[i+1] for i in range(len(myList)-1))

class ascendingOrderTetser(unittest.TestCase):
    def testList(self):
        myList = [1,2,3,4,5,6,7]
        print("Sorted list: ", myList)
        self.assertTrue(ascendingOrderTest(myList), "List is sorted in ascending order")
    def testWrongList(self):
        myList = [1,5,6,2,4,3]
        print("Unsorted list: ", myList)
        self.assertFalse(ascendingOrderTest(myList), "List is unsorted")
if __name__ == '__main__':
    unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/Git
Sorted list:  [1, 2, 3, 4, 5, 6, 7]
.Unsorted list:  [1, 5, 6, 2, 4, 3]
.
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

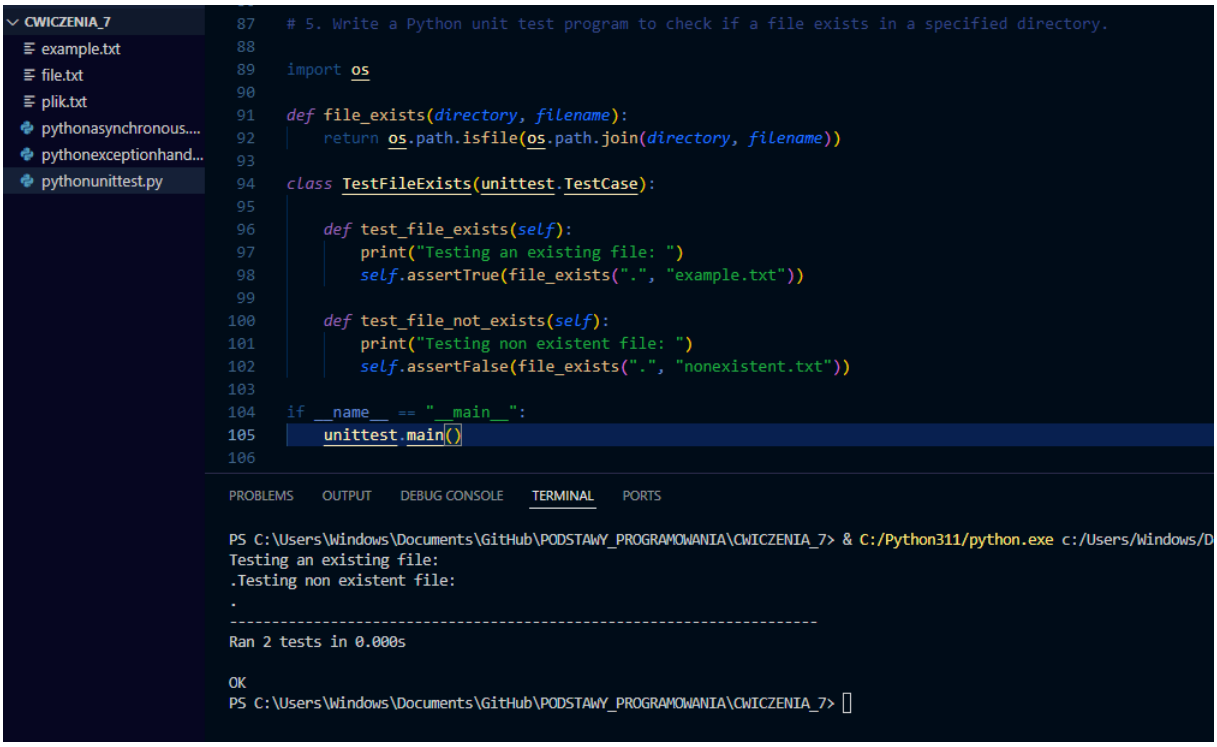
```python
# 3. Write a Python unit test program that checks if two lists are equal.

def equalLists(list1, list2):
    return list1 == list2

class testListsEquality(unittest.TestCase):
    def testEqualLists(self):
        print("\nEqual list test:\n", [5,582,37,238], "\n", [5,582,37,238])
        self.assertTrue(equalLists([5,582,37,238], [5,582,37,238]), "The lists are equal")

    def testUnequalLists(self):
        print("\nUnequal list test:\n", [1, 2, 3, 4], "\n", [4,5,6,778435])
        self.assertFalse(equalLists([1, 2, 3, 4], [4,5,6,778435]), "The lists are not equal")

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAW

Equal list test:
 [5, 582, 37, 238]
 [5, 582, 37, 238]
.
Unequal list test:
 [1, 2, 3, 4]
 [4, 5, 6, 778435]
.
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python

# 4. Write a Python unit test program to check if a string is a palindrome.

def isPalindrome(s):
    return s == s[::-1]

class testPalindrome(unittest.TestCase):

    def testIfPalindrome(self):
        palindrome = "kajak"
        print('Test palindrome: ', palindrome)
        self.assertTrue(isPalindrome(palindrome), "String is a palindrome")

    def testIfNotPalindrome(self):
        notAPalindrome = "hello"
        print("Testing non palindrome: ", notAPalindrome)
        self.assertFalse(isPalindrome(notAPalindrome), "String is not a palindrome")

if __name__ == "__main__":
    unittest.main()

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PR
Testing non palindrome:  hello
.Test palindrome:  kajak
.
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```



```python
# 6. Write a Python unit test that checks if a function handles floating-point calculations accurately.


def addFloats(a, b):
    return a + b

class testFloatingPoint(unittest.TestCase):
    def testAddFloats(self):
        print("Test:", addFloats(0.1, 0.2))
        self.assertAlmostEqual(addFloats(0.1, 0.2), 0.3, places=7)
        print("Test:", addFloats(1.1, 2.2))
        self.assertAlmostEqual(addFloats(1.1, 2.2), 3.3, places=7)
if __name__ == "__main__":
    unittest.main()


```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTA
Test: 0.3000000000000004
Test: 3.3000000000000003
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 7. Write a Python unit test program to check if a function handles multi-threading correctly.
import threading

def task():
    result = 0
    for i in range(1, 50):
        result += i
    return result

class multiThreadingTester(unittest.TestCase):
    def testMultiThreading(self):
        numberOfThreads = 10
        threads = []
        for _ in range(numberOfThreads):
            t = threading.Thread(target=task)
            threads.append(t)
            t.start()
        for t in threads:
            t.join()
        for t in threads:
            self.assertFalse(t.is_alive())
if __name__ == "__main__":
    unittest.main()


```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGR
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 8. Write a Python unit test program to check if a database connection is successful.

import sqlite3

class databaseConnectionTester(unittest.TestCase):
    def test_database_connection(self):
        conn = sqlite3.connect(':memory:')
        cursor = conn.cursor()
        cursor.execute("SELECT 10")
        result = cursor.fetchone()
        cursor.close()
        conn.close()
        self.assertEqual(result, (10,))
if __name__ == '__main__':
    unittest.main()

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Docume
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 9. Write a Python unit test program to check if a database query returns the expected results.

class testDatabaseQuery(unittest.TestCase):
    def setUp(self):
        self.conn = sqlite3.connect(':memory:')
        self.cursor = self.conn.cursor()
        self.cursor.execute("CREATE TABLE friends (id INTEGER PRIMARY KEY, name TEXT, surname TEXT)")
        self.cursor.execute("INSERT INTO friends (name, surname) VALUES ('Marcin', 'Kowalski')")
        self.cursor.execute("INSERT INTO friends (name, surname) VALUES ('Jan', 'Nowak')")
        self.conn.commit()

    def tearDown(self):
        self.cursor.close()
        self.conn.close()

    def test_database_query(self):
        self.cursor.execute("SELECT name, surname FROM friends ORDER BY name")
        results = self.cursor.fetchall()

        expected_results = [('Jan', 'Nowak'),  ('Marcin', 'Kowalski')]

        self.assertEqual(results, expected_results)

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

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 10. Write a Python unit test program to check if a function correctly parses and validates input data.

def parseAndValidate(data):
    if isinstance(data, str) and data.isnumeric():
        return int(data) > 0
    return False

class testInputParsing(unittest.TestCase):
    def test_valid_input(self):
        data = "21489214772184128934217 8"
        result = parseAndValidate(data)
        self.assertTrue(result)

    def test_invalid_input(self):
        data = "Euro 2024"
        result = parseAndValidate(data)
        self.assertFalse(result)

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

```

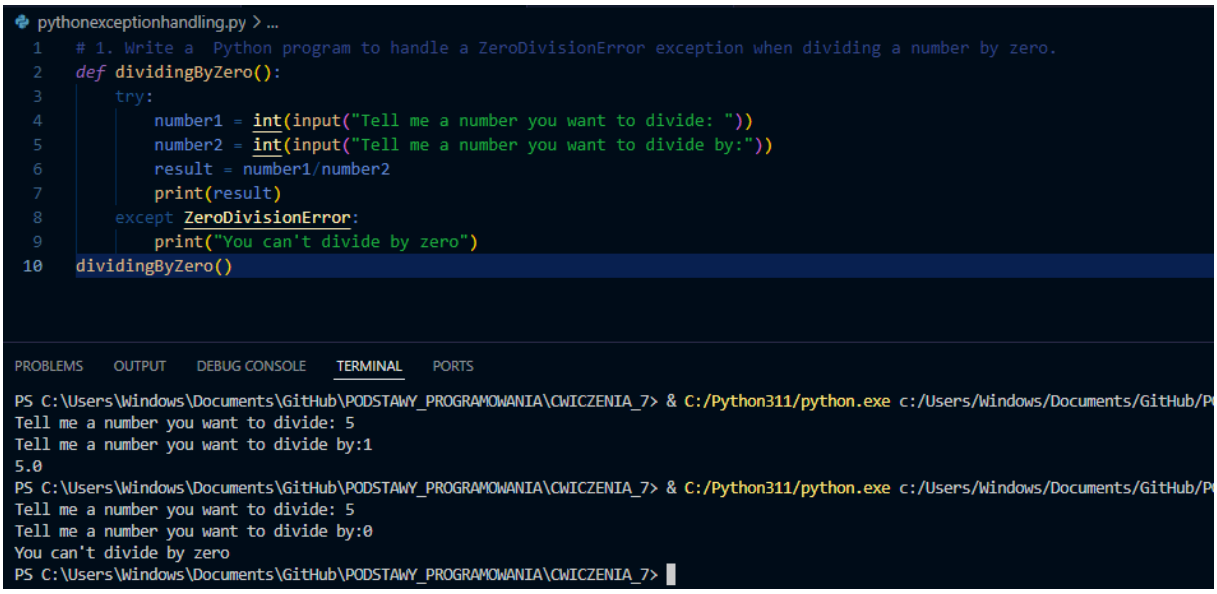
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

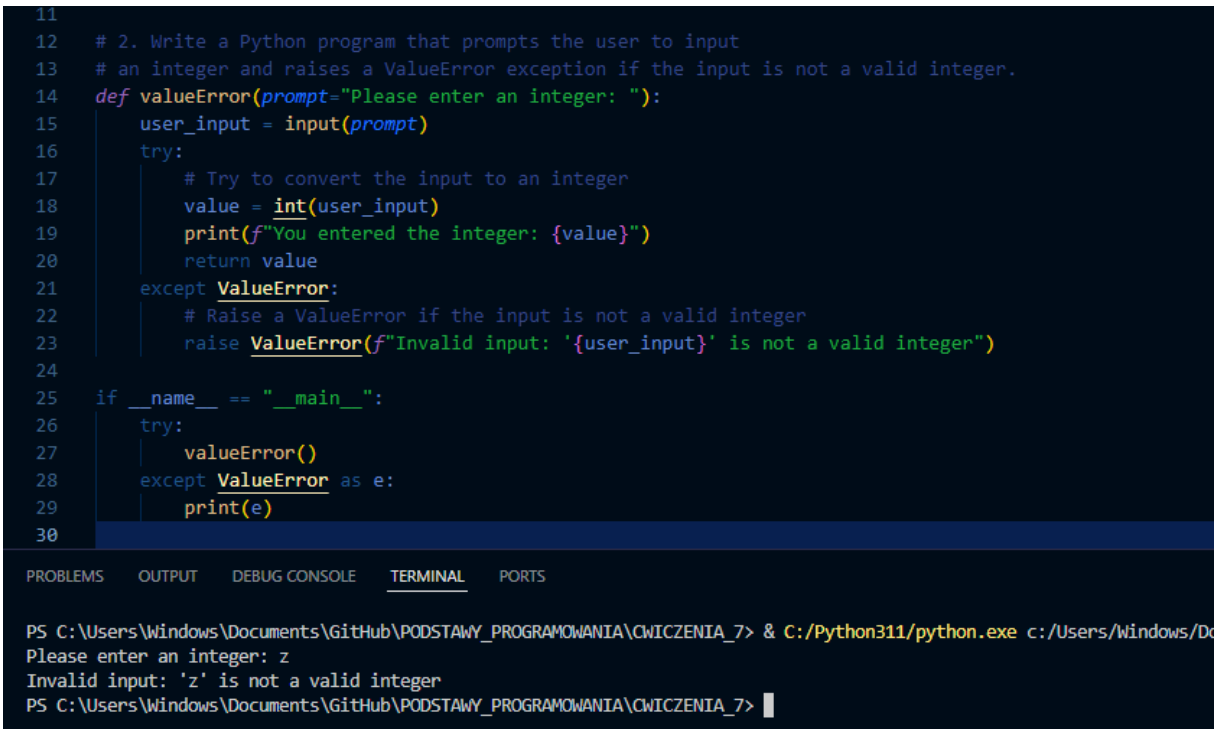
```
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

Python Exception Handling

(pythonexceptionhandling.py)

Obsługa wyjątków w Pythonie jest realizowana za pomocą konstrukcji ‘try’ i ‘except’, które pozwalają na przechwytywanie i reakcję na błędy podczas wykonywania programu. Dzięki temu możliwe jest zapobieganie awariom aplikacji i zapewnienie jej poprawnego działania w obliczu niespodziewanych sytuacji. Dodatkowo, konstrukcja ‘raise’ pozwala na wymuszenie wyjątków w celu sygnalizowania błędnych stanów.





```python
# 3. Write a  Python program that opens a file and handles a FileNotFoundError exception if the file does not exist.
def fileFinder(name):
    try:
        file = open(name, 'r')
        contents = file.read
        print("Content:")
        print(contents)
        file.close()
    except FileNotFoundError:
        print("File does not exist")
fileFinder("folder")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_P
File does not exist
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 4. Write a Python program that prompts the user to input
# two numbers and raises a TypeError exception if the inputs are not numerical.
def getInput(prompt):
    user_input = input(prompt)
    try:
        value = float(user_input)
        return value
    except ValueError:
        raise TypeError(f'{user_input} is not a number')

if __name__ == "__main__":
    try:
        num1 = getInput("First number:: ")
        num2 = getInput("Second number: ")
    except TypeError as e:
        print(e)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Win
First number:: 5
Second number: z
z is not a number
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 5. Write a Python program that opens a file and handles a PermissionError exception if there is a permission issue.

def fileOpener(file_name):
    try:
        with open(file_name, 'r') as file:
            content = file.read()
            print(content)
    except PermissionError:
        print(f'Permission denied')
        raise
fileOpener("plik.txt")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANI
kijdxkdjxkd
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 6. Write a  Python program that executes an operation on a list and handles an IndexError exception if the index is out of range.
def rangeError():
    myList = [1,2,3,4,5,6,7]
    index = int(input("Number:"))
    try:
        number = myList[index]
    except IndexError:
        print("Number not in range")
rangeError()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/CWICZENIA_
Number:10
Number not in range
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 7. Write a Python program that prompts the user to input a number and handles a KeyboardInterrupt exception if the user cancels the input.
def userCancel():
    try:
        number = int(input("Give me a number:"))
        print(number)
    except KeyboardInterrupt:
        print("Input has been cancelled by user")
userCancel()
```

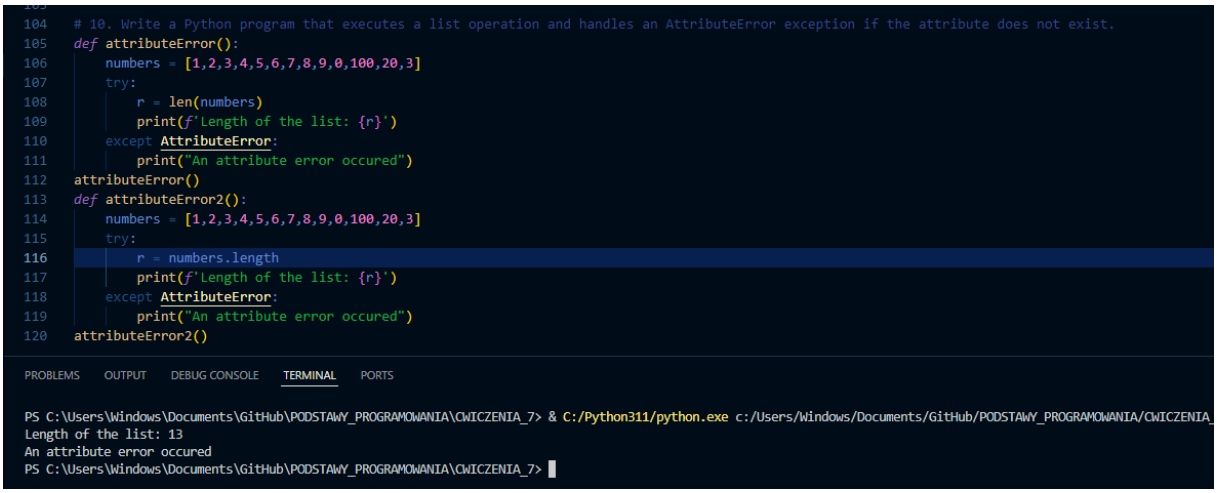
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/CWICZENIA_7/pythonexcep
Give me a number:Input has been cancelled by user
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/CWICZENIA_7/pythonexcep
Give me a number:5
5
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 8. Write a  Python program that executes division and handles an ArithmeticError exception if there is an arithmetic error.
def arithmeticError(number1, number2):
    try:
        result = number1/number2
        print(f'{number1} divided by {number2} gives: {result}')
    except ArithmeticError:
        print("Arithmetic error occured!")
arithmeticError(50,10)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/
Arithmetic error occured!
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/
50 divided by 10 gives: 5.0
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 9. Write a Python program that opens a file and handles a UnicodeDecodeError exception if there is an encoding issue.
def read_file(file_name, encoding='utf-8'):
    try:
        with open(file_name, 'r', encoding=encoding) as file:
            content = file.read()
            print(content)
    except UnicodeDecodeError:
        print(f"An UnicodeDecodeError appeared!")
read_file("example.txt")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRA
An UnicodeDecodeError appeared!
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```



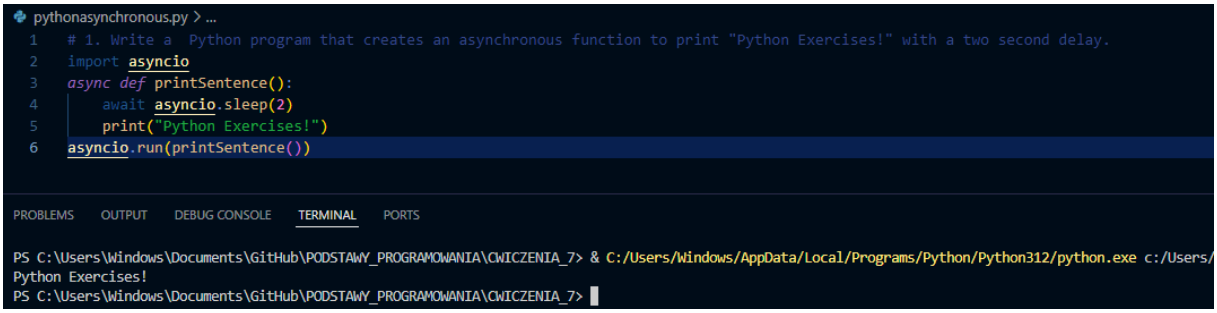
```python
# 10. Write a Python program that executes a list operation and handles an AttributeError exception if the attribute does not exist.
def attributeError():
    numbers = [1,2,3,4,5,6,7,8,9,0,100,20,3]
    try:
        r = len(numbers)
        print(f'Length of the list: {r}')
    except AttributeError:
        print("An attribute error occured")
attributeError()
def attributeError2():
    numbers = [1,2,3,4,5,6,7,8,9,0,100,20,3]
    try:
        r = numbers.length
        print(f'Length of the list: {r}')
    except AttributeError:
        print("An attribute error occured")
attributeError2()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGRAMOWANIA/CWICZENIA_
Length of the list: 13
An attribute error occured
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

Python Asynchronous

(pythonasynchronous.py)

Programowanie asynchroniczne w Pythonie jest realizowane za pomocą modułu ‘asyncio’, który umożliwia pisanie współbieżnego kodu w prosty i czytelny sposób. Kluczowe są słowa ‘async’ i ‘await’, które kolejno pozwalają na deklarowanie funkcji asynchronicznych oraz wstrzymywanie ich wykonania. Dzięki temu można efektywnie zarządzać zasobami i poprawiać wydajność aplikacji w przypadku operacji sieciowych czy odczytu/zapisu danych.



pythonasynchronous.py > ...

```python
# 1. Write a  Python program that creates an asynchronous function to print "Python Exercises!" with a two second delay.
import asyncio
async def printSentence():
    await asyncio.sleep(2)
    print("Python Exercises!")
asyncio.run(printSentence())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Users/Windows/AppData/Local/Programs/Python/Python312/python.exe c:/Users/
Python Exercises!
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 2. Write a Python program that creates three asynchronous functions and
# displays their respective names with different delays (1 second, 2 seconds, and 3 seconds).

async def functionOne():
    await asyncio.sleep(1)
    print("functionOne")

async def functionTwo():
    await asyncio.sleep(2)
    print("functionTwo")

async def functionThree():
    await asyncio.sleep(3)
    print("functionThree")

asyncio.run(functionOne())
asyncio.run(functionTwo())
asyncio.run(functionThree())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Users/Windows/AppData/Local/Programs/Python/Py
functionOne
functionTwo
functionThree
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 3. Write a Python program that creates an asyncio event loop and runs
# a coroutine that prints numbers from 1 to 7 with a delay of 1 second each.

async def numbers():
    for i in range(1,8):
        print(i)
        await asyncio.sleep(1)
asyncio.run(numbers())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Users/Windows/AppData/Local/Pr
1
2
3
4
5
6
7
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 4. Write a Python program that implements a coroutine to
# fetch data from two different URLs simultaneously using the "aiohttp" library.

import aiohttp

async def fetch_url(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()
async def main():
    urls = [
        'https://www.youtube.com/',
        'https://github.com/KasniQ/PODSTAWY_PROGRAMOWANIA/tree/main/CWICZENIA_7'
    ]
    url1 = asyncio.create_task(fetch_url(urls[0]))
    url2 = asyncio.create_task(fetch_url(urls[1]))
    data1 = await url1
    data2 = await url2
    print("Data from ",urls[0], len(data1), "bytes")
    print("Data from ",urls[1], len(data2), "bytes")
asyncio.run(main())

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/
Data from  https://www.youtube.com/ 540326 bytes
Data from  https://github.com/KasniQ/PODSTAWY_PROGRAMOWANIA/tree/main/CWICZENIA_7 242138 bytes
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 5. Write a  Python program that runs multiple asynchronous tasks
# concurrently using asyncio.gather() and measures the time taken.
import time

async def taskOne():
    print("Task 1")
    await asyncio.sleep(5)
    print("Task 1 Finished")

async def taskTwo():
    print("Task 2")
    await asyncio.sleep(2)
    print("Task 2 Finished")

async def taskThree():
    print("Task 3")
    await asyncio.sleep(2)
    print("Task 3 Finished")

async def main():
    start = time.time()
    await asyncio.gather(taskOne(), taskTwo(), taskThree())
    finish = time.time()
    timeTaken = finish - start
    print("Time taken: ", timeTaken)
asyncio.run(main())

```

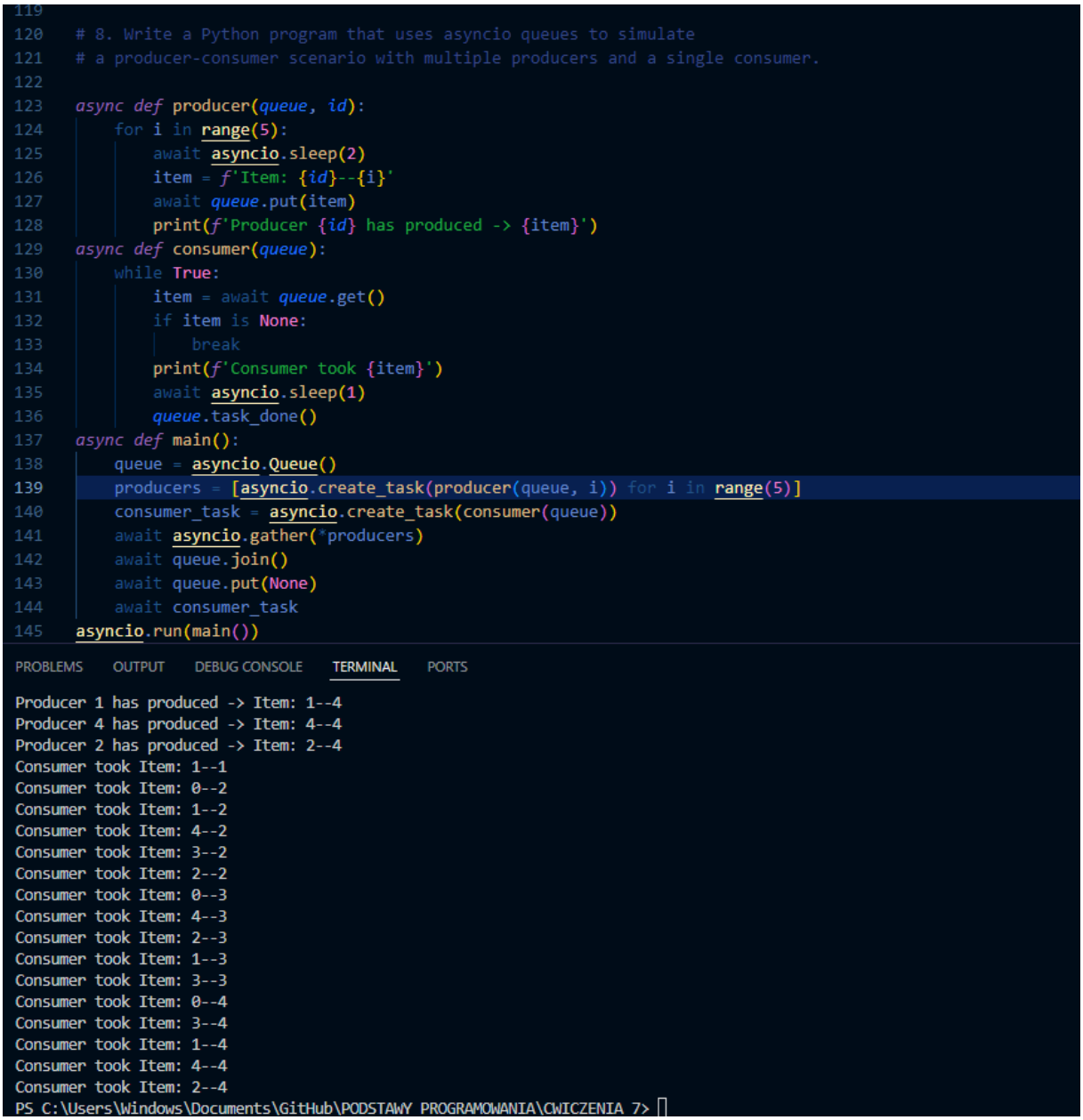
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PR
Task 1
Task 2
Task 3
Task 2 Finished
Task 3 Finished
Task 1 Finished
Time taken:  5.0086541175842285
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 6. Write a  Python program to create a coroutine that simulates
# a time-consuming task and use asyncio.CancelledError to handle task cancellation.
async def taskOne():
    print("Task started")
    try:
        for i in range(10):
            print(f"Thinking... {i+1}/10")
            await asyncio.sleep(1)
    except asyncio.CancelledError:
        print("Task cancelled")
        raise
async def main():
    task = asyncio.create_task(taskOne())
    await asyncio.sleep(5)
    task.cancel()
    try:
        await task
    except asyncio.CancelledError:
        print("The task has been cancelled due to environmental problems.")
asyncio.run(main())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documen
Task started
Thinking... 1/10
Thinking... 2/10
Thinking... 3/10
Thinking... 4/10
Thinking... 5/10
Task cancelled
The task has been cancelled due to environmental problems.
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```

```python
# 7. Write a Python program that implements a timeout for an asynchronous operation using asyncio.wait_for().
async def anotherTask(seconds):
    print(f'Starting task for {seconds} seconds')
    await asyncio.sleep(seconds)
    return f'Task has been completed in {seconds} seconds'
async def main():
    timeout = 8
    try:
        result = await asyncio.wait_for(anotherTask(9), timeout)
        print(result)
    except asyncio.TimeoutError:
        print(f'Timout error occured after {timeout} seconds')
asyncio.run(main())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7> & C:/Python311/python.exe c:/Users/Windows/Documents/GitHub/PODSTAWY_PROGR
Starting task for 9 seconds
Timout error occured after 8 seconds
PS C:\Users\Windows\Documents\GitHub\PODSTAWY_PROGRAMOWANIA\CWICZENIA_7>
```



```python
# 8. Write a Python program that uses asyncio queues to simulate
# a producer-consumer scenario with multiple producers and a single consumer.

async def producer(queue, id):
    for i in range(5):
        await asyncio.sleep(2)
        item = f'Item: {id}--{i}'
        await queue.put(item)
        print(f'Producer {id} has produced -> {item}')
async def consumer(queue):
    while True:
        item = await queue.get()
        if item is None:
            break
        print(f'Consumer took {item}')
        await asyncio.sleep(1)
        queue.task_done()
async def main():
    queue = asyncio.Queue()
    producers = [asyncio.create_task(producer(queue, i)) for i in range(5)]
    consumer_task = asyncio.create_task(consumer(queue))
    await asyncio.gather(*producers)
    await queue.join()
    await queue.put(None)
    await consumer_task
asyncio.run(main())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Producer 1 has produced -> Item: 1--4
Producer 4 has produced -> Item: 4--4
Producer 2 has produced -> Item: 2--4
Consumer took Item: 1--1
Consumer took Item: 0--2
Consumer took Item: 1--2
Consumer took Item: 4--2
Consumer took Item: 3--2
Consumer took Item: 2--2
Consumer took Item: 0--3
Consumer took Item: 4--3
Consumer took Item: 2--3
Consumer took Item: 1--3
Consumer took Item: 3--3
Consumer took Item: 0--4
Consumer took Item: 3--4
Consumer took Item: 1--4
Consumer took Item: 4--4
Consumer took Item: 2--4
PS C:\Users\Windows\Documents\GitHub\PODSTAWY PROGRAMOWANIA\CWICZENIA 7>
```

Spis treści

SPRAWOZDANIE 1

Python Unit Test 2

Python Exception Handling 8

Python Asynchronous 11