# Home page | WSB Merito University in Warsaw

# Podstawy Programowania – zadanie

Bartosz Gromek, nr albumu: 162584   
Grupa C3, Informatyka inż. niestacjonarnie  
Semestr 2

Poniżej umieściłem 28 zadań oddzielone tematycznie, wraz z odpowiedziami, linkiem do repozytorium, screenami konsoli oraz treścią tych zadań.

Repozytorium github: https://github.com/Rartosz/py-tasks-3

**Unit Test**

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

import unittest

def is\_prime(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\_numbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

        print("Prime numbers:", prime\_numbers)

        for number in prime\_numbers:

            self.assertTrue(is\_prime(number), f"{number} is not recognized as a prime number")

    def test\_non\_prime\_numbers(self):

        non\_prime\_numbers = [4, 6, 8, 10, 12, 14, 16, 18, 20]

        print("Non-prime numbers:", non\_prime\_numbers)

        for number in non\_prime\_numbers:

            self.assertFalse(is\_prime(number), f"{number} is incorrectly recognized as a prime number")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

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

import unittest

def is\_sorted\_ascending(lst):

    return all(lst[i] <= lst[i+1] for i in range(len(lst)-1))

class TestSortedAscending(unittest.TestCase):

    def test\_sorted\_list(self):

        lst = [1, 2, 3, 4, 5, 6, 7]

        print("Sorted list: ", lst)

        self.assertTrue(is\_sorted\_ascending(lst), "The list is not sorted in ascending order")

    def test\_unsorted\_list(self):

        lst = [5, 7, 2, 8, 1, 9]

        print("Unsorted list: ", lst)

        self.assertFalse(is\_sorted\_ascending(lst), "The list is sorted in ascending order")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

**3.** Write a Python unit test program that checks if two lists are equal.

import unittest

def lists\_are\_equal(nums1, nums2):

    return nums1 == nums2

class TestListsEquality(unittest.TestCase):

    def test\_equal\_lists(self):

        nums1 = [10, 20, 30, 40]

        nums2 = [10, 20, 30, 40]

        print("\nEqual list test:\n", nums1, "\n", nums2)

        self.assertTrue(lists\_are\_equal(nums1, nums2), "The lists are not equal")

    def test\_unequal\_lists(self):

        nums1 = [10, 20, 30, 40]

        nums2 = [30, 20, 10, 40]

        print("\nUnequal list test:\n", nums1, "\n", nums2)

        self.assertFalse(lists\_are\_equal(nums1, nums2), "The lists are equal")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie**

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

import unittest

def is\_palindrome(string):

    return string == string[::-1]

class TestPalindrome(unittest.TestCase):

    def test\_palindrome\_string(self):

        palindrome = "hello"

        print("Test palindrome:", palindrome)

        self.assertTrue(is\_palindrome(palindrome), "The string is not a palindrome")

    def test\_non\_palindrome\_string(self):

        non\_palindrome = "madam"

        print("Test non palindrome:", non\_palindrome)

        self.assertFalse(is\_palindrome(non\_palindrome), "The string is a palindrome")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

5. Write a Python unit test program to check if a file exists in a specified directory.

import os

import unittest

def file\_exists(directory, filename):

    file\_path = os.path.join(directory, filename)

    return os.path.exists(file\_path)

class TestFileExists(unittest.TestCase):

    def test\_existing\_file(self):

        directory = '/path/txt'

        filename = 'test1.txt'

        self.assertTrue(file\_exists(directory, filename), "The file does not exist in the specified directory")

    def test\_nonexistent\_file(self):

        directory = '/path/txt'

        filename = 'test2.txt'

        self.assertFalse(file\_exists(directory, filename), "The file exists in the specified directory")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

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

import unittest

class TestFloatingPointCalculations(unittest.TestCase):

    def test\_addition(self):

        result = 0.3 + 0.5

        self.assertAlmostEqual(result, 0.8, places=6)

    def test\_multiplication(self):

        result = 0.3 \* 0.5

        self.assertAlmostEqual(result, 0.15, places=6)

    def test\_division(self):

        result = 0.7 / 0.3

        self.assertAlmostEqual(result, 2.333333, places=6)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import unittest

import threading

def perform\_task():

    result = 0

    for i in range(1, 100000):

        result += i

    return result

class Test\_Multi\_Threading(unittest.TestCase):

    def test\_multi\_threading(self):

        num\_threads = 10

        threads = []

        for \_ in range(num\_threads):

            t = threading.Thread(target=perform\_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()

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import unittest

import sqlite3

class TestDatabaseConnection(unittest.TestCase):

    def test\_database\_connection(self):

        conn = sqlite3.connect(':memory:')

        cursor = conn.cursor()

        cursor.execute("SELECT 1")

        result = cursor.fetchone()

        cursor.close()

        conn.close()

        self.assertEqual(result, (1,))

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import unittest

import sqlite3

class TestDatabaseQuery(unittest.TestCase):

    def setUp(self):

        self.conn = sqlite3.connect(':memory:')

        self.cursor = self.conn.cursor()

        self.cursor.execute("CREATE TABLE employees (id INTEGER PRIMARY KEY, name TEXT, salary REAL)")

        self.cursor.execute("INSERT INTO employees (name, salary) VALUES ('Ylva Guiomar', 1800.0)")

        self.cursor.execute("INSERT INTO employees (name, salary) VALUES ('Scott Gregorius', 2100.0)")

        self.conn.commit()

    def tearDown(self):

        self.cursor.close()

        self.conn.close()

    def test\_database\_query(self):

        self.cursor.execute("SELECT name, salary FROM employees ORDER BY name")

        results = self.cursor.fetchall()

        expected\_results = [('Scott Gregorius', 2100.0), ('Ylva Guiomar', 1800.0)]

        self.assertEqual(results, expected\_results)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import unittest

def parse\_and\_validate\_input(data):

    if isinstance(data, str) and data.isnumeric():

        return int(data) > 0

    return False

class TestInputParsing(unittest.TestCase):

    def test\_valid\_input(self):

        data = "100"

        result = parse\_and\_validate\_input(data)

        self.assertTrue(result)

    def test\_invalid\_input(self):

        data = "Hello"

        result = parse\_and\_validate\_input(data)

        self.assertFalse(result)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**Exception Handling**

1. Write a Python program to handle a ZeroDivisionError exception when dividing a number by zero.

def divide\_numbers(x, y):

    try:

        result = x / y

        print("Result:", result)

    except ZeroDivisionError:

        print("The division by zero operation is not allowed.")

numerator = 100

denominator = 0

divide\_numbers(numerator, denominator)

****

**2.** Write a Python program that prompts the user to input an integer and raises a ValueError exception if the input is not a valid integer.

def get\_integer\_input(prompt):

    try:

        value = int(input(prompt))

        return value

    except ValueError:

        print("Error: Invalid input, input a valid integer.")

n = get\_integer\_input("Input an integer: ")

print("Input value:", n)

**Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie**

**3.** Write a Python program that opens a file and handles a FileNotFoundError exception if the file does not exist.

def open\_file(filename):

    try:

        file = open(filename, 'r')

        contents = file.read()

        print("File contents:")

        print(contents)

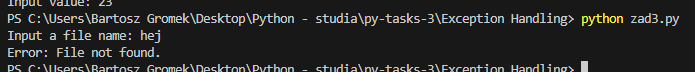
        file.close()

    except FileNotFoundError:

        print("Error: File not found.")

file\_name = input("Input a file name: ")

open\_file(file\_name)

****

**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 get\_numeric\_input(prompt):

    while True:

        try:

            value = float(input(prompt))

            return value

        except ValueError:

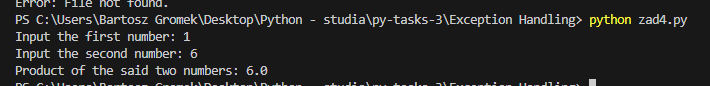
            print("Error: Invalid input. Please Input a valid number.")

n1 = get\_numeric\_input("Input the first number: ")

n2 = get\_numeric\_input("Input the second number: ")

result = n1 \* n2

print("Product of the said two numbers:", result)

****

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

def open\_file(filename):

    try:

        with open(filename, 'w') as file:

            contents = file.read()

            print("File contents:")

            print(contents)

    except PermissionError:

        print("Error: Permission denied to open the file.")

file\_name = input("Input a file name: ")

open\_file(file\_name)

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 test\_index(data, index):

    try:

        result = data[index]

        print("Result:", result)

    except IndexError:

        print("Error: Index out of range.")

nums = [1, 2, 3, 4, 5, 6, 7]

index = int(input("Input the index: "))

test\_index(nums, index)

****

7. Write a Python program that prompts the user to input a number and handles a KeyboardInterrupt exception if the user cancels the input..

try:

    n = int(input("Input a number: "))

    print("You entered:", n)

except KeyboardInterrupt:

    print("Input canceled by the user.")

****

8. Write a Python program that executes division and handles an ArithmeticError exception if there is an arithmetic error.

def division(dividend, divisor):

    try:

        result = dividend / divisor

        print("Result:", result)

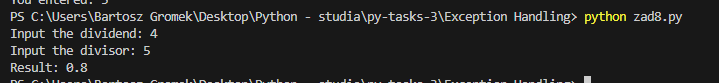
    except ArithmeticError:

        print("Error: Arithmetic error occurred!")

dividend = float(input("Input the dividend: "))

divisor = float(input("Input the divisor: "))

division(dividend, divisor)

****

9. Write a Python program that opens a file and handles a UnicodeDecodeError exception if there is an encoding issue.

def open\_file(filename):

    encoding = input("Input the encoding (ASCII, UTF-16, UTF-8) for the file: ")

    try:

        with open(filename, 'r', encoding=encoding) as file:

            contents = file.read()

            print("File contents:")

            print(contents)

    except UnicodeDecodeError:

        print("Error: Encoding issue occurred while reading the file.")

file\_name = input("Input the file name: ")

open\_file(file\_name)

10. Write a Python program that executes a list operation and handles an AttributeError exception if the attribute does not exist.

def test\_list\_operation(nums):

    try:

        r = len(nums)

        print("Length of the list:", r)

    except AttributeError:

        print("Error: The list does not have a 'length' attribute.")

nums = [1, 2, 3, 4, 5]

test\_list\_operation(nums)



**Asynchronous**

1. Write a Python program that creates an asynchronous function to print "Python Exercises!" with a two second delay

import asyncio

async def print\_delayed\_message():

    await asyncio.sleep(2)

    print("Python Exercises!")

async def main():

    await print\_delayed\_message()

asyncio.run(main())

****

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).

import asyncio

async def display\_name\_with\_delay(name, delay):

    await asyncio.sleep(delay)

    print(name)

async def main():

    tasks = [

        display\_name\_with\_delay("Asyn. function-1", 1),

        display\_name\_with\_delay("Asyn. function-2", 2),

        display\_name\_with\_delay("Asyn. function-3", 3)

    ]

    await asyncio.gather(\*tasks)

asyncio.run(main())

**Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie**

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.

import asyncio

async def display\_numbers():

    for i in range(1, 8):

        print(i)

        await asyncio.sleep(1)

asyncio.run(display\_numbers())

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

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

import asyncio

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():

    url\_1 = "https://www.wikipedia.org/"

    url\_2 = "https://www.google.com"

    task1 = asyncio.create\_task(fetch\_url(url\_1))

    task2 = asyncio.create\_task(fetch\_url(url\_2))

    data1 = await task1

    data2 = await task2

    print("Data from ",url\_1, len(data1), "bytes")

    print("Data from ",url\_2, len(data2), "bytes")

asyncio.run(main())

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import asyncio

import time

async def task1():

    print("Task-1 started")

    await asyncio.sleep(4)

    print("Task-1 completed")

async def task2():

    print("Task-2 started")

    await asyncio.sleep(1)

    print("Task-2 completed")

async def task3():

    print("Task-3 started\n")

    await asyncio.sleep(2)

    print("Task-3 completed")

async def main():

    start\_time = time.time()

    await asyncio.gather(task1(), task2(), task3())

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    print("\nAll tasks completed in {:.2f} seconds".format(elapsed\_time))

asyncio.run(main())

**Obraz zawierający tekst, oprogramowanie, Czcionka, Oprogramowanie multimedialne

Opis wygenerowany automatycznie**

6. Write a Python program to create a coroutine that simulates a time-consuming task and use asyncio.CancelledError to handle task cancellation.

import asyncio

import random

async def time\_consuming\_task():

    print('Time-consuming task started...')

    try:

        for i in range(1, 6):

            await asyncio.sleep(random.randint(1,5))

            print(f'Step {i} completed')

    except asyncio.CancelledError:

        print('Time consuming task was cancelled')

        raise

async def main():

    task = asyncio.create\_task(time\_consuming\_task())

    await asyncio.sleep(random.randint(1,3))

    task.cancel()

    try:

        await task

    except asyncio.CancelledError:

        print('Main coroutine caught task cancellation!')

asyncio.run(main())

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

7. Write a Python program that implements a timeout for an asynchronous operation using asyncio.wait\_for().

import asyncio

import time

async def time\_consuming\_task(duration):

    print(f'Starting long operation for {duration} seconds...')

    await asyncio.sleep(duration)

    return f'Long operation completed in {duration} seconds'

async def main():

    timeout =3

    try:

        result = await asyncio.wait\_for(time\_consuming\_task(8), timeout)

        print(result)

    except asyncio.TimeoutError:

        print(f'Timeout occurred after waiting for {timeout} seconds')

asyncio.run(main())

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

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

import asyncio

import random

async def producer(queue, id):

    for i in range(3):

        item = f"Item: {id}-{i}"

        await queue.put(item)

        print(f"Producer {id} produced-> {item}")

        await asyncio.sleep(random.uniform(0.1, 0.5))

async def consumer(queue):

    while True:

        item = await queue.get()

        if item is None:

            break

        print(f"Consumer consumed {item}")

        queue.task\_done()

async def main():

    queue = asyncio.Queue()

    producers = [asyncio.create\_task(producer(queue, i)) for i in range(3)]

    consumer\_task = asyncio.create\_task(consumer(queue))

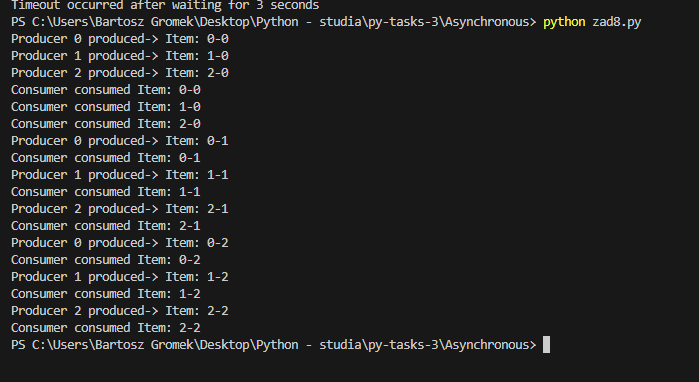
    await asyncio.gather(\*producers)

    await queue.join()

    await queue.put(None)

    await consumer\_task

asyncio.run(main())

****