Міністерство освіти і науки України

Державний університет “Житомирська політехніка”

Кафедра інженерії програмного забезпечення

Група: ВТ-21-1[1]

Програмування мовою Python

Лабораторна робота № 5

«Функції»

Виконав: Бабушко А. С.

Прийняв: Морозов Д. С.

					«Житомирська політехніка».22.121.01.000–Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Бабушко А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 00
Перевір.		Морозов Д.С.					
Керівник						ФІКТ Гр. ВТ-21-1[1]	
Н. контр.							
Затверд.							

Мета роботи: ознайомитися основами функціонального програмування і використання користувацьких функцій в мові Python

Хід роботи:

Завдання на лабораторну роботу:

Завдання 1. Користувач вводить дві сторони трьох прямокутників. Вивести їх площі.

Лістинг програми:

```
from time import perf_counter

""" Lab 5. Python. Andrii Babushko. Repository: https://github.com/AndriiBabushko/Python
"""


# task 1
def task_1_get_rectangles_areas(rectangles_sides):
    areas = []

    for i in range(0, len(rectangles_sides)):
        areas.append(rectangles_sides[i][0] * rectangles_sides[i][1])

    return areas


def enter_rectangles_sides(counter):
    rectangles_sides = []

    while counter != 0:
        try:
            side_a = float(input(f'Enter length of side a: '))
            side_b = float(input(f'Enter length of side b: '))
            rectangles_sides.append([side_a, side_b])
            if side_a < 0 or side_b < 0:
                raise ValueError(f'Sides менше 0!')
            else:
                pass
                print('Sides were entered correctly!')
                counter -= 1
        except ValueError as value_error:
            rectangles_sides.pop()
            print('ERROR:', value_error)

    return rectangles_sides


print('\nTASK 1!!!')
task_1_rectangles_sides = enter_rectangles_sides(3)
print(f'List of rectangles sides: {task_1_rectangles_sides}')
task_1_rectangles_areas = task_1_get_rectangles_areas(task_1_rectangles_sides)
for rectangle in range(0, len(task_1_rectangles_areas)):
    print(f'{rectangle + 1} rectangle\'s area: {task_1_rectangles_areas[rectangle]}')
```

Результат програми:

```
TASK 1!!!
Enter length of side a: 2
Enter length of side b: 10
Sides were entered correctly!
Enter length of side a: 4
Enter length of side b: 15
Sides were entered correctly!
Enter length of side a: 1
Enter length of side b: 13
Sides were entered correctly!
List of rectangles sides: [[2.0, 10.0], [4.0, 15.0], [1.0, 13.0]]
1 rectangle's area: 20.0
2 rectangle's area: 60.0
3 rectangle's area: 13.0
```

Завдання 2. Дано катети двох прямокутних трикутників. Написати функцію обчислення довжини гіпотенузи цих трикутників. Порівняти і вивести яка з гіпотенуз більше, а яка менше.

Лістинг програми:

```python
# task 2
def task_2_get_right_triangles_hypotenuses(right_triangles_legs):
    hypotenuses = []

    for i in range(0, len(right_triangles_legs)):
        hypotenuses.append(round((right_triangles_legs[i][0] ** 2 +
right_triangles_legs[i][1] ** 2) ** 0.5, 2))

    return hypotenuses


def enter_right_triangles_legs(counter):
    right_triangles_legs = []

    while counter != 0:
        try:
            leg_a = float(input(f'Enter leg a: '))
            leg_b = float(input(f'Enter leg b: '))
            right_triangles_legs.append([leg_a, leg_b])
            if leg_a < 0 or leg_b < 0:
                raise ValueError(f'Legs are less than 0!')
            else:
                pass
                print('Legs were entered correctly!')
                counter -= 1
        except ValueError as value_error:
            right_triangles_legs.pop()
            print('ERROR:', value_error)

    return right_triangles_legs


def compare_hypotenuses(hypotenuses):
    for i in range(0, len(hypotenuses) - 1):
        if hypotenuses[i] > hypotenuses[i + 1]:
            print(f'Hypotenuse №{i + 1}({hypotenuses[i]}) > Hypotenuse №{i +
```

```
2}({hypotenuses[i + 1]})')
        else:
            print(f'Hypotenuse №{i + 1}({hypotenuses[i]}) < Hypotenuse №{i +
2}({hypotenuses[i + 1]})')


print('\nTASK 2!!!')
task_2_right_triangles_legs = enter_right_triangles_legs(2)
print(f'List of right triangle legs: {task_2_right_triangles_legs}')
task_2_hypotenuses = task_2_get_right_triangles_hypotenuses(task_2_right_triangles_legs)
compare_hypotenuses(task_2_hypotenuses)
```

Результат програми:

```
TASK 2!!!
Enter leg a: 2
Enter leg b: 4
Legs were entered correctly!
Enter leg a: 3
Enter leg b: 8
Legs were entered correctly!
List of right triangle legs: [[2.0, 4.0], [3.0, 8.0]]
Hypotenuse №1(4.47) < Hypotenuse №2(8.54)
```

Завдання 3. Задано коло (x-a)2 + (y-b)2 = R2 і точки P (p1, p2), F (f1, f1), L (l1, l2). З'ясувати і вивести на екран, скільки точок лежить всередині кола. Перевірку, чи лежить точка всередині кола, оформити у вигляді функції.

Лістинг програми:

```
# task 3
def check_if_point_is_in_circle(checked_point, circle_points_and_radius):
    equation = (checked_point[0] - circle_points_and_radius[0]) ** 2 + (
            checked_point[1] - circle_points_and_radius[1]) ** 2

    if (circle_points_and_radius[2] ** 2) == equation:
        return True
    return False


def enter_circle_center_points_and_radius():
    while True:
        try:
            point_a = float(input(f'Enter circle center point a: '))
            point_b = float(input(f'Enter circle center point b: '))
            circle_radius = float(input(f'Enter radius: '))
            if circle_radius < 0:
                raise ValueError(f'Radius is less than 0!')
            else:
                pass
                print('Center circle points and radius were entered correctly!')
                break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return [point_a, point_b, circle_radius]


def enter_some_point(point):
    point_a = 0
    point_b = 0
```

```
    while True:
        try:
            if point == 0:
                point_a = float(input(f'Enter point p1 of P: '))
                point_b = float(input(f'Enter point p2 of P: '))
            if point == 1:
                point_a = float(input(f'Enter point f1 of F: '))
                point_b = float(input(f'Enter point f2 of F: '))
            if point == 2:
                point_a = float(input(f'Enter point l1 of L: '))
                point_b = float(input(f'Enter point l2 of L: '))
            break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return [point_a, point_b]


print('\nTASK 3!!!')
counter_point_in_circle = 0
counter_point_out_of_circle = 0
task_3_circle_center_and_radius = enter_circle_center_points_and_radius()
print(f'Center point O({task_3_circle_center_and_radius[0]},
{task_3_circle_center_and_radius[1]}).'
      f' Radius: {task_3_circle_center_and_radius[2]}')
for i in range(0, 3):
    task_3_some_point = enter_some_point(i)
    if check_if_point_is_in_circle(task_3_some_point, task_3_circle_center_and_radius):
        counter_point_in_circle += 1
    else:
        counter_point_out_of_circle += 1

print(f'Count of points which are in circle: {counter_point_in_circle}')
print(f'Count of points which are out of circle: {counter_point_out_of_circle}')
```

Результат програми:

```
TASK 3!!!
Enter circle center point a: 4
Enter circle center point b: 8
Enter radius: 6
Center circle points and radius were entered correctly!
Center point O(4.0, 8.0). Radius: 6.0
Enter point p1 of P: 1
Enter point p2 of P: 2
Enter point f1 of F: 6
Enter point f2 of F: -2
Enter point l1 of L: 1
Enter point l2 of L: -4
Count of points which are in circle: 0
Count of points which are out of circle: 3
```

Завдання 4. Дано числа X, Y, Z, T - довжини сторін чотирикутника. Обчислити його площу, якщо кут між сторонами довжиною X і Y - прямий.

Лістинг програми:

```python
# task 4
def enter_quadrangle_data():
    from math import sqrt
    quadrangle = {
        'x': 0,
        'y': 0,
        'z': 0,
        't': 0,
        'diagonal': 0
    }

    while True:
        try:
            quadrangle['x'] = float(input('Enter x: '))
            quadrangle['y'] = float(input('Enter y: '))
            quadrangle['z'] = float(input('Enter z: '))
            quadrangle['t'] = float(input('Enter t: '))
            quadrangle['diagonal'] = sqrt(quadrangle['x'] ** 2 + quadrangle['y'] ** 2)
            if quadrangle['x'] < 0 or quadrangle['y'] < 0 or quadrangle['z'] < 0 or
quadrangle['t'] < 0:
                raise ValueError(f'Some side length is less than 0!')
            else:
                pass
                print('Sides length were entered correctly!')
                break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return quadrangle


def get_first_square(x, y):
    return x * y * 0.5


def get_second_square(d, z, t):
    from math import sqrt

    p = (z + t + d) / 2

    return sqrt(p * (p - z) * (p - t) * (p - d))


print('\nTASK 4!!!')
task_4_quadrangle = enter_quadrangle_data()
task_4_square_of_quadrangle = round(
    get_first_square(task_4_quadrangle['x'], task_4_quadrangle['y']) +
    get_second_square(task_4_quadrangle['diagonal'], task_4_quadrangle['z'],
task_4_quadrangle['t'])
    , 2)
print(f'Square of quadrangle: {task_4_square_of_quadrangle}')
```

Результат програми:

```
TASK 4!!!
Enter x: 4
Enter y: 2
Enter z: 6
Enter t: 2
Sides length were entered correctly!
Square of quadrangle: 7.32
```

Завдання 5. Знайти всі натуральні числа, що не перевищують заданого n, які діляться на кожне із заданих користувачем чисел.

Лістинг програми:

```
# task 5
def task_5_get_natural_numbers(n):
    natural_numbers = []

    for number in range(1, n + 1):
        if n % number == 0:
            natural_numbers.append(number)

    return natural_numbers


def enter_n():
    while True:
        try:
            n = int(input('Enter n: '))
            if n > 0:
                pass
                print('Number "n" was entered correctly!')
                break
            else:
                raise ValueError(f'Number is less than 0!')
        except ValueError as value_error:
            print('ERROR:', value_error)

    return n


def enter_count_checked_numbers():
    while True:
        try:
            count_checked_numbers = int(input('How much do you want to check natural
numbers? '))
            if count_checked_numbers >= 1:
                pass
                break
            else:
                raise ValueError(f'Number is less than 0!')
        except ValueError as value_error:
            print('ERROR:', value_error)

    return count_checked_numbers


print('\nTASK 5!!!')
task_5_count_checked_numbers = enter_count_checked_numbers()
for i in range(0, task_5_count_checked_numbers):
    task_5_n = enter_n()
    print(f'{i + 1}) Entered N = {task_5_n}')
    task_5_natural_numbers = task_5_get_natural_numbers(task_5_n)
    print(f'List of natural numbers: {task_5_natural_numbers}')
```

Результат програми:

```
TASK 5!!!
How much do you want to check natural numbers? 3
Enter n: 50
Number "n" was entered correctly!
1) Entered N = 50
List of natural numbers: [1, 2, 5, 10, 25, 50]
Enter n: 30
Number "n" was entered correctly!
2) Entered N = 30
List of natural numbers: [1, 2, 3, 5, 6, 10, 15, 30]
Enter n: 26
Number "n" was entered correctly!
3) Entered N = 26
List of natural numbers: [1, 2, 13, 26]
```

Завдання 6. Скласти програму для знаходження чисел з інтервалу [M, N], що мають найбільшу кількість дільників.

Лістинг програми:

```
# task 6
def task_6_numbers_with_large_number_of_divisors():
    M_N_interval = enter_M_N_interval()
    print(f'[M, N] interval: {M_N_interval}')
    M = M_N_interval[0]
    N = M_N_interval[1]
    numbers_with_large_number_of_divisors = get_numbers_with_large_number_of_divisors(M, N)
    print(f'Dictionary of numbers with large number of divisors:
{numbers_with_large_number_of_divisors}')


def get_numbers_with_large_number_of_divisors(M, N):
    numbers_with_large_number_of_divisors = {}

    for number in range(M, N):
        counter = 0
        for divisor in range(1, N):
            if number % divisor == 0:
                counter += 1
        numbers_with_large_number_of_divisors.update({number: counter})

    values_list_of_dictionary = numbers_with_large_number_of_divisors.values()
    max_counter_from_list = max(values_list_of_dictionary)

    number_with_max_divisors = {i for i in numbers_with_large_number_of_divisors if
                                numbers_with_large_number_of_divisors[i] ==
max_counter_from_list}

    return number_with_max_divisors


def enter_M_N_interval():
    while True:
        try:
            M = int(input('Enter started M point of interval: '))
            N = int(input('Enter finished N point of interval: '))
            if M >= N:
```

```
                raise ValueError(f'M is greater than N!')
            else:
                pass
                print('Points of interval were entered correctly!')
                break
        except ValueError as value_err:
            print(f'ERROR: {value_err}')

    return [M, N]


print('\nTASK 6!!!')
task_6_numbers_with_large_number_of_divisors()
```

Результат програми:

```
TASK 6!!!
Enter started M point of interval: 2
Enter finished N point of interval: 88
Points of interval were entered correctly!
[M, N] interval: [2, 88]
Dictionary of numbers with large number of divisors: {72, 60, 84}
```

Завдання 7. Написати функцію для пошуку всіх простих чисел від 0 до N з можливістю вибору формату представлення результату (списком; рядками в стовпчик; просто вивести кількість простих чисел.

Лістинг програми:

```
# task 7
def task_7_result_in_appropriate_format():
    task_7_n = enter_n()
    print(f'Entered N: {task_7_n}')
    task_7_entered_format = enter_format()
    output_result_in_appropriate_format(task_7_n, task_7_entered_format)


def enter_format():
    while True:
        try:
            print(f'Ways for output result: "list", "by strings", "count primes"')
            format_for_output = str(input('Enter one of the variants of output here: '))
            if format_for_output != 'list' and format_for_output != 'by strings' and
format_for_output != 'count primes':
                raise ValueError('Format was entered incorrectly!')
            else:
                pass
                print('Format was entered correctly!')
                break
        except ValueError as value_err:
            print(f'ERROR: {value_err}')

    return format_for_output


def output_result_in_appropriate_format(N, some_format):
    if some_format == 'list':
        output_primary_list = []

        for number in range(0, N):
            if is_prime(number):
                output_primary_list.append(number)

        print(f'List of primary numbers: {output_primary_list}')
```

```
    elif some_format == 'by strings':
        output_string = 'Primary numbers:\n'
        counter = 1

        for number in range(0, N):
            if is_prime(number):
                output_string += f'{counter}) {number}\n'
                counter += 1

        print(f'{output_string}')
    else:
        counter = 0

        for number in range(0, N):
            if is_prime(number):
                counter += 1

        print(f'Total count of primary numbers in interval [0, N]: {counter}')


def is_prime(number):
    for delimiter in range(2, (number // 2) + 1):
        if number % delimiter == 0:
            return False

    return True


print('\nTASK 7!!!')
task_7_result_in_appropriate_format()
```

Результат програми:

```
TASK 7!!!
Enter n: 10
Number "n" was entered correctly!
Entered N: 10
Ways for output result: "list", "by strings", "count primes"
Enter one of the variants of output here: list
Format was entered correctly!
List of primary numbers: [0, 1, 2, 3, 5, 7]
```

Завдання 8. Дано список з випадкових натуральних чисел довільної довжини. Написати програму, що формуватиме з заданого другий список, що міститиме тільки значення від MIN+bottom до MAX-upper. Де MIN і MAX – відповідно найменше і найбільше число в списку, a botton і upper – нижня і верхня межа значень вибірки нового списку. Програма має містити обробку винятків на випадок введення символів невірного типу, дробових чисел, вихід за межі мінімального і максимального значення.

Лістинг програми:

```
# task 8
def task_8_new_list_from_another():
    task_8_created_random_int_list = create_random_integers_list(get_random_int_number(10,
25))
    print(f'New created list: {task_8_created_random_int_list}')
    min_number = min(task_8_created_random_int_list)
    max_number = max(task_8_created_random_int_list)
    bottom_and_upper = get_bottom_and_upper(min_number, max_number)
```

```
    task_8_new_list_from_created_list =
get_new_list_from_another(task_8_created_random_int_list, min_number,
                                                          max_number,
bottom_and_upper[0], bottom_and_upper[1])
    print(f'New list from recently created one: {task_8_new_list_from_created_list}')


def get_new_list_from_another(some_list, min_number, max_number, bottom, upper):
    new_list = []

    for number in some_list:
        if min_number + bottom <= number <= max_number - upper:
            new_list.append(number)

    return new_list


def get_bottom_and_upper(min_number, max_number):
    while True:
        try:
            bottom = int(input('Enter bottom value: '))
            upper = int(input('Enter upper value: '))
            if min_number + bottom > max_number:
                raise ValueError(f'Min + bottom({min_number + bottom}) >
Max({max_number})')
            elif max_number - upper < min_number:
                raise ValueError(f'Max + upper({max_number + upper}) < Min({min_number})')
            elif bottom != int(bottom) and upper != int(upper):
                raise ValueError(f'Bottom or upper is float number!')
            else:
                pass
                print('Numbers were entered correctly!')
                break
        except ValueError as value_err:
            print(f'ERROR: {value_err}')

    return [bottom, upper]


def get_random_int_number(min_random, max_random):
    import random as random
    return round((random.random() * (max_random - min_random) + min_random))


def create_random_integers_list(count_integers):
    created_list = []

    for counter in range(0, count_integers):
        created_list.append(get_random_int_number(1, 999))

    return created_list


print('\nTASK 8!!!')
task_8_new_list_from_another()
```

Результат програми:

```
TASK 8!!!
New created list: [607, 659, 579, 203, 31, 526, 875, 530, 263, 145, 23, 569, 398, 30, 521, 647, 776, 495, 35, 782, 351]
Enter bottom value: 120
Enter upper value: 250
Numbers were entered correctly!
New list from recently created one: [607, 579, 203, 526, 530, 263, 145, 569, 398, 521, 495, 351]
```

Завдання 9. Для завдань 6 – 8 написати декоратор, що дозволить визначати час виконання програми. Виконати перевірку часу виконання написаних функцій для 10**n елементів при n <=6 з кроком в n. Тобто визначити час виконання функцій для десятків, сотень, тисяч і так до мільйону елементів.

Лістинг програми:

```
# task 9
def task_9_task_6_decorator(n, task_6_func):
    print('\nLaunch time check decorator of task 6!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            task_6_func(get_random_int_number(0, 500), get_random_int_number(500, 1000))
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


def task_9_task_7_decorator(n, task_7_func):
    print('\nLaunch time check decorator of task 7!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            n = get_random_int_number(1, 100)
            formats = ['list', 'count primes']
            entered_format = formats[get_random_int_number(0, 1)]
            task_7_func(n, entered_format)
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


def task_9_task_8_decorator(n, task_8_func):
    print('\nLaunch time check decorator of task 8!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            created_random_int_list = create_random_integers_list(get_random_int_number(10,
25))
            min_number = min(created_random_int_list)
            max_number = max(created_random_int_list)
            bottom = get_random_int_number(min_number, int(max_number / 2))
            upper = get_random_int_number(min_number, int(max_number / 2))
            task_8_func(created_random_int_list, min_number, max_number, bottom, upper)
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


print('\nTASK 9!!!')
task_9_task_6_decorator(6, get_numbers_with_large_number_of_divisors)
task_9_task_7_decorator(6, output_result_in_appropriate_format)
task_9_task_8_decorator(6, get_new_list_from_another)
```

Результат програми:

```
TASK 9!!!


Launch time check decorator of task 6!
Time of executing function 10 times: 0.17547 sec.


Launch time check decorator of task 7!
Total count of primary numbers in interval [0, N]: 9
Total count of primary numbers in interval [0, N]: 6
List of primary numbers: [0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]
Total count of primary numbers in interval [0, N]: 15
List of primary numbers: [0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
Total count of primary numbers in interval [0, N]: 26
List of primary numbers: [0, 1]
Total count of primary numbers in interval [0, N]: 13
Total count of primary numbers in interval [0, N]: 13
Time of executing function 10 times: 0.00031 sec.


Launch time check decorator of task 8!
Time of executing function 10 times: 0.00017 sec.
```

Увесь лістинг програми:

```python
from time import perf_counter

""" Lab 5. Python. Andrii Babushko. Repository: https://github.com/AndriiBabushko/Python
"""


# task 1
def task_1_get_rectangles_areas(rectangles_sides):
    areas = []

    for i in range(0, len(rectangles_sides)):
        areas.append(rectangles_sides[i][0] * rectangles_sides[i][1])

    return areas


def enter_rectangles_sides(counter):
    rectangles_sides = []

    while counter != 0:
        try:
            side_a = float(input(f'Enter length of side a: '))
            side_b = float(input(f'Enter length of side b: '))
            rectangles_sides.append([side_a, side_b])
            if side_a < 0 or side_b < 0:
                raise ValueError(f'Sides менше 0!')
            else:
                pass
                print('Sides were entered correctly!')
                counter -= 1
        except ValueError as value_error:
            rectangles_sides.pop()
```

```
            print('ERROR:', value_error)

    return rectangles_sides


print('\nTASK 1!!!')
task_1_rectangles_sides = enter_rectangles_sides(3)
print(f'List of rectangles sides: {task_1_rectangles_sides}')
task_1_rectangles_areas = task_1_get_rectangles_areas(task_1_rectangles_sides)
for rectangle in range(0, len(task_1_rectangles_areas)):
    print(f'{rectangle + 1} rectangle\'s area: {task_1_rectangles_areas[rectangle]}')


# task 2
def task_2_get_right_triangles_hypotenuses(right_triangles_legs):
    hypotenuses = []

    for i in range(0, len(right_triangles_legs)):
        hypotenuses.append(round((right_triangles_legs[i][0] ** 2 +
right_triangles_legs[i][1] ** 2) ** 0.5, 2))

    return hypotenuses


def enter_right_triangles_legs(counter):
    right_triangles_legs = []

    while counter != 0:
        try:
            leg_a = float(input(f'Enter leg a: '))
            leg_b = float(input(f'Enter leg b: '))
            right_triangles_legs.append([leg_a, leg_b])
            if leg_a < 0 or leg_b < 0:
                raise ValueError(f'Legs are less than 0!')
            else:
                pass
                print('Legs were entered correctly!')
                counter -= 1
        except ValueError as value_error:
            right_triangles_legs.pop()
            print('ERROR:', value_error)

    return right_triangles_legs


def compare_hypotenuses(hypotenuses):
    for i in range(0, len(hypotenuses) - 1):
        if hypotenuses[i] > hypotenuses[i + 1]:
            print(f'Hypotenuse №{i + 1}({hypotenuses[i]}) > Hypotenuse №{i +
2}({hypotenuses[i + 1]})')
        else:
            print(f'Hypotenuse №{i + 1}({hypotenuses[i]}) < Hypotenuse №{i +
2}({hypotenuses[i + 1]})')


print('\nTASK 2!!!')
task_2_right_triangles_legs = enter_right_triangles_legs(2)
print(f'List of right triangle legs: {task_2_right_triangles_legs}')
task_2_hypotenuses = task_2_get_right_triangles_hypotenuses(task_2_right_triangles_legs)
compare_hypotenuses(task_2_hypotenuses)


# task 3
def check_if_point_is_in_circle(checked_point, circle_points_and_radius):
    equation = (checked_point[0] - circle_points_and_radius[0]) ** 2 + (
            checked_point[1] - circle_points_and_radius[1]) ** 2

    if (circle_points_and_radius[2] ** 2) == equation:
        return True
```

```
    return False


def enter_circle_center_points_and_radius():
    while True:
        try:
            point_a = float(input(f'Enter circle center point a: '))
            point_b = float(input(f'Enter circle center point b: '))
            circle_radius = float(input(f'Enter radius: '))
            if circle_radius < 0:
                raise ValueError(f'Radius is less than 0!')
            else:
                pass
                print('Center circle points and radius were entered correctly!')
                break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return [point_a, point_b, circle_radius]


def enter_some_point(point):
    point_a = 0
    point_b = 0

    while True:
        try:
            if point == 0:
                point_a = float(input(f'Enter point p1 of P: '))
                point_b = float(input(f'Enter point p2 of P: '))
            if point == 1:
                point_a = float(input(f'Enter point f1 of F: '))
                point_b = float(input(f'Enter point f2 of F: '))
            if point == 2:
                point_a = float(input(f'Enter point l1 of L: '))
                point_b = float(input(f'Enter point l2 of L: '))
            break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return [point_a, point_b]


print('\nTASK 3!!!')
counter_point_in_circle = 0
counter_point_out_of_circle = 0
task_3_circle_center_and_radius = enter_circle_center_points_and_radius()
print(f'Center point O({task_3_circle_center_and_radius[0]},
{task_3_circle_center_and_radius[1]}).'
      f' Radius: {task_3_circle_center_and_radius[2]}')
for i in range(0, 3):
    task_3_some_point = enter_some_point(i)
    if check_if_point_is_in_circle(task_3_some_point, task_3_circle_center_and_radius):
        counter_point_in_circle += 1
    else:
        counter_point_out_of_circle += 1

print(f'Count of points which are in circle: {counter_point_in_circle}')
print(f'Count of points which are out of circle: {counter_point_out_of_circle}')


# task 4
def enter_quadrangle_data():
    from math import sqrt
    quadrangle = {
        'x': 0,
        'y': 0,
        'z': 0,
        't': 0,
```

```
        'diagonal': 0
    }

    while True:
        try:
            quadrangle['x'] = float(input('Enter x: '))
            quadrangle['y'] = float(input('Enter y: '))
            quadrangle['z'] = float(input('Enter z: '))
            quadrangle['t'] = float(input('Enter t: '))
            quadrangle['diagonal'] = sqrt(quadrangle['x'] ** 2 + quadrangle['y'] ** 2)
            if quadrangle['x'] < 0 or quadrangle['y'] < 0 or quadrangle['z'] < 0 or
quadrangle['t'] < 0:
                raise ValueError(f'Some side length is less than 0!')
            else:
                pass
                print('Sides length were entered correctly!')
                break
        except ValueError as value_error:
            print('ERROR:', value_error)

    return quadrangle


def get_first_square(x, y):
    return x * y * 0.5


def get_second_square(d, z, t):
    from math import sqrt

    p = (z + t + d) / 2

    return sqrt(p * (p - z) * (p - t) * (p - d))


print('\nTASK 4!!!')
task_4_quadrangle = enter_quadrangle_data()
task_4_square_of_quadrangle = round(
    get_first_square(task_4_quadrangle['x'], task_4_quadrangle['y']) +
    get_second_square(task_4_quadrangle['diagonal'], task_4_quadrangle['z'],
task_4_quadrangle['t'])
    , 2)
print(f'Square of quadrangle: {task_4_square_of_quadrangle}')


# task 5
def task_5_get_natural_numbers(n):
    natural_numbers = []

    for number in range(1, n + 1):
        if n % number == 0:
            natural_numbers.append(number)

    return natural_numbers


def enter_n():
    while True:
        try:
            n = int(input('Enter n: '))
            if n > 0:
                pass
                print('Number "n" was entered correctly!')
                break
            else:
                raise ValueError(f'Number is less than 0!')
        except ValueError as value_error:
            print('ERROR:', value_error)
```

```
    return n


def enter_count_checked_numbers():
    while True:
        try:
            count_checked_numbers = int(input('How much do you want to check natural
numbers? '))
            if count_checked_numbers >= 1:
                pass
                break
            else:
                raise ValueError(f'Number is less than 0!')
        except ValueError as value_error:
            print('ERROR:', value_error)

    return count_checked_numbers


print('\nTASK 5!!!')
task_5_count_checked_numbers = enter_count_checked_numbers()
for i in range(0, task_5_count_checked_numbers):
    task_5_n = enter_n()
    print(f'{i + 1}) Entered N = {task_5_n}')
    task_5_natural_numbers = task_5_get_natural_numbers(task_5_n)
    print(f'List of natural numbers: {task_5_natural_numbers}')


# task 6
def task_6_numbers_with_large_number_of_divisors():
    M_N_interval = enter_M_N_interval()
    print(f'[M, N] interval: {M_N_interval}')
    M = M_N_interval[0]
    N = M_N_interval[1]
    numbers_with_large_number_of_divisors = get_numbers_with_large_number_of_divisors(M, N)
    print(f'Dictionary of numbers with large number of divisors:
{numbers_with_large_number_of_divisors}')


def get_numbers_with_large_number_of_divisors(M, N):
    numbers_with_large_number_of_divisors = {}

    for number in range(M, N):
        counter = 0
        for divisor in range(1, N):
            if number % divisor == 0:
                counter += 1
        numbers_with_large_number_of_divisors.update({number: counter})

    values_list_of_dictionary = numbers_with_large_number_of_divisors.values()
    max_counter_from_list = max(values_list_of_dictionary)

    number_with_max_divisors = {i for i in numbers_with_large_number_of_divisors if
                                numbers_with_large_number_of_divisors[i] ==
max_counter_from_list}

    return number_with_max_divisors


def enter_M_N_interval():
    while True:
        try:
            M = int(input('Enter started M point of interval: '))
            N = int(input('Enter finished N point of interval: '))
            if M >= N:
                raise ValueError(f'M is greater than N!')
            else:
                pass
                print('Points of interval were entered correctly!')
```

```
                break
            except ValueError as value_err:
                print(f'ERROR: {value_err}')

    return [M, N]


print('\nTASK 6!!!')
task_6_numbers_with_large_number_of_divisors()


# task 7
def task_7_result_in_appropriate_format():
    task_7_n = enter_n()
    print(f'Entered N: {task_7_n}')
    task_7_entered_format = enter_format()
    output_result_in_appropriate_format(task_7_n, task_7_entered_format)


def enter_format():
    while True:
        try:
            print(f'Ways for output result: "list", "by strings", "count primes"')
            format_for_output = str(input('Enter one of the variants of output here: '))
            if format_for_output != 'list' and format_for_output != 'by strings' and
format_for_output != 'count primes':
                raise ValueError('Format was entered incorrectly!')
            else:
                pass
                print('Format was entered correctly!')
                break
        except ValueError as value_err:
            print(f'ERROR: {value_err}')

    return format_for_output


def output_result_in_appropriate_format(N, some_format):
    if some_format == 'list':
        output_primary_list = []

        for number in range(0, N):
            if is_prime(number):
                output_primary_list.append(number)

        print(f'List of primary numbers: {output_primary_list}')
    elif some_format == 'by strings':
        output_string = 'Primary numbers:\n'
        counter = 1

        for number in range(0, N):
            if is_prime(number):
                output_string += f'{counter}) {number}\n'
                counter += 1

        print(f'{output_string}')
    else:
        counter = 0

        for number in range(0, N):
            if is_prime(number):
                counter += 1

        print(f'Total count of primary numbers in interval [0, N]: {counter}')


def is_prime(number):
    for delimiter in range(2, (number // 2) + 1):
        if number % delimiter == 0:
```

```
            return False

    return True


print('\nTASK 7!!!')
task_7_result_in_appropriate_format()


# task 8
def task_8_new_list_from_another():
    task_8_created_random_int_list = create_random_integers_list(get_random_int_number(10,
25))
    print(f'New created list: {task_8_created_random_int_list}')
    min_number = min(task_8_created_random_int_list)
    max_number = max(task_8_created_random_int_list)
    bottom_and_upper = get_bottom_and_upper(min_number, max_number)
    task_8_new_list_from_created_list =
get_new_list_from_another(task_8_created_random_int_list, min_number,
                                                                max_number,
bottom_and_upper[0], bottom_and_upper[1])
    print(f'New list from recently created one: {task_8_new_list_from_created_list}')


def get_new_list_from_another(some_list, min_number, max_number, bottom, upper):
    new_list = []

    for number in some_list:
        if min_number + bottom <= number <= max_number - upper:
            new_list.append(number)

    return new_list


def get_bottom_and_upper(min_number, max_number):
    while True:
        try:
            bottom = int(input('Enter bottom value: '))
            upper = int(input('Enter upper value: '))
            if min_number + bottom > max_number:
                raise ValueError(f'Min + bottom({min_number + bottom}) >
Max({max_number})')
            elif max_number - upper < min_number:
                raise ValueError(f'Max + upper({max_number + upper}) < Min({min_number})')
            elif bottom != int(bottom) and upper != int(upper):
                raise ValueError(f'Bottom or upper is float number!')
            else:
                pass
                print('Numbers were entered correctly!')
                break
        except ValueError as value_err:
            print(f'ERROR: {value_err}')

    return [bottom, upper]


def get_random_int_number(min_random, max_random):
    import random as random
    return round((random.random() * (max_random - min_random) + min_random))


def create_random_integers_list(count_integers):
    created_list = []

    for counter in range(0, count_integers):
        created_list.append(get_random_int_number(1, 999))

    return created_list
```

```
print('\nTASK 8!!!')
task_8_new_list_from_another()


# task 9
def task_9_task_6_decorator(n, task_6_func):
    print('\nLaunch time check decorator of task 6!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            task_6_func(get_random_int_number(0, 500), get_random_int_number(500, 1000))
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


def task_9_task_7_decorator(n, task_7_func):
    print('\nLaunch time check decorator of task 7!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            n = get_random_int_number(1, 100)
            formats = ['list', 'count primes']
            entered_format = formats[get_random_int_number(0, 1)]
            task_7_func(n, entered_format)
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


def task_9_task_8_decorator(n, task_8_func):
    print('\nLaunch time check decorator of task 8!')
    for i in range(1, 7, n):
        time_started = perf_counter()
        for repeat in range(1, 10 ** i):
            created_random_int_list = create_random_integers_list(get_random_int_number(10,
25))
            min_number = min(created_random_int_list)
            max_number = max(created_random_int_list)
            bottom = get_random_int_number(min_number, int(max_number / 2))
            upper = get_random_int_number(min_number, int(max_number / 2))
            task_8_func(created_random_int_list, min_number, max_number, bottom, upper)
        time_finished = perf_counter()
        time = time_finished - time_started
        print(f'Time of executing function {10 ** i} times: {round(time, 5)} sec.')


print('\nTASK 9!!!')
task_9_task_6_decorator(6, get_numbers_with_large_number_of_divisors)
task_9_task_7_decorator(6, output_result_in_appropriate_format)
task_9_task_8_decorator(6, get_new_list_from_another)
```

Висновок: під час виконання лабораторної роботи було отримано навички написання власних функцій та організації коду за допомогою них.