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.

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

***Хід роботи:***

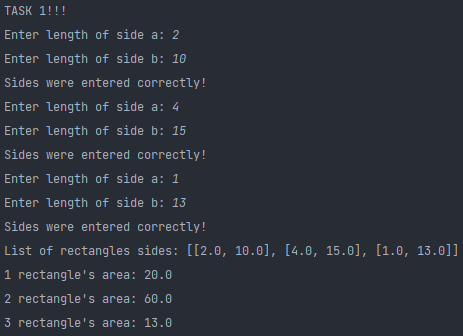
***Завдання на лабораторну роботу:***

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:***

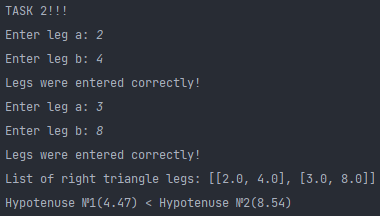
******

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:***

*# 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:***

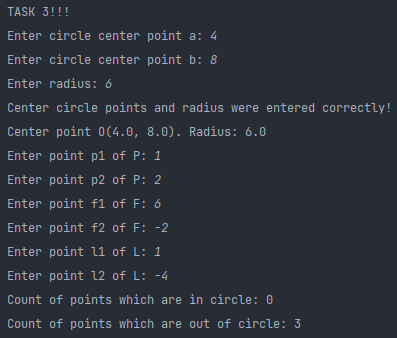
******

Завдання 3. Задано коло (x-a)2 + (y-b)2 = R2 і точки Р (р1, р2), 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:***

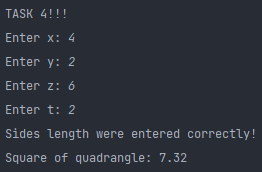
******

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

***Лістинг програми:***

*# 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:***

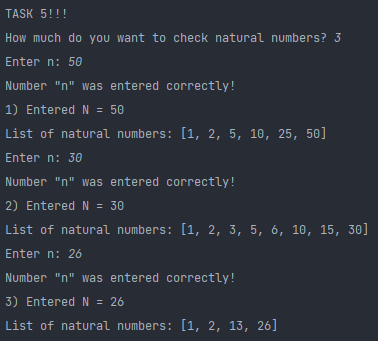
******

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:***

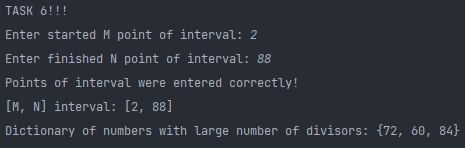
******

Завдання 6. Скласти програму для знаходження чисел з інтервалу [М, 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:***

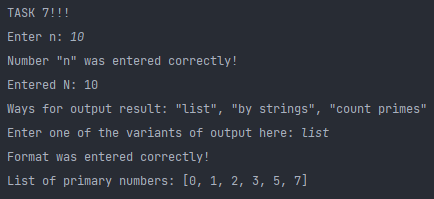
******

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:***

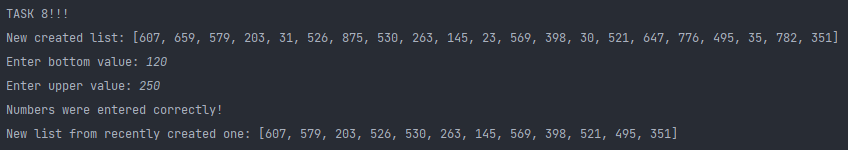
******

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, а 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:***

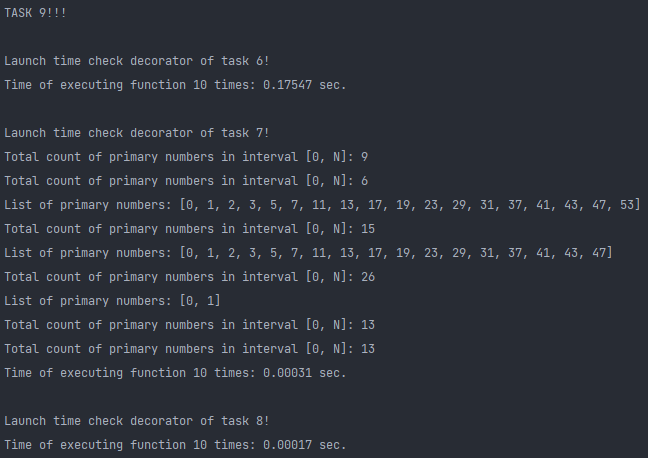
******

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:***

******

***Увесь 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]}')  
  
  
*# 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.