Міністерство освіти і науки України
Державний університет „Житомирська політехніка”

Кафедра ІПЗ
Група: ІПЗ-23-1

Програмування мовою Python
Лабораторна робота №5
«ФУНЦІЇ»

Виконав: Риженко Я.В.
Прийняв: Желізко В. В.

					«Житомирська політехніка».24.121.05.000		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Риженко Я.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 11
Перевір.		Желізко В. В.					
Керівник						ФІКТ, гр. ІПЗ-23-1	
Н. контр.							
Затверд.							

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

Хід роботи

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

Листинг програми:

```
for i in range(3):
    a = float(input(f"Введіть першу сторону прямокутника {i + 1}: "))
    b = float(input(f"Введіть другу сторону прямокутника {i + 1}: "))
    print(f"Площа прямокутника {i + 1}: {round(a*b,3)}")
```

Результат виконання:

```
Введіть першу сторону прямокутника 1: 7
Введіть другу сторону прямокутника 1: 3.6
Площа прямокутника 1: 25.2
Введіть першу сторону прямокутника 2: 2.4
Введіть другу сторону прямокутника 2: 5
Площа прямокутника 2: 12.0
Введіть першу сторону прямокутника 3: 3.55
Введіть другу сторону прямокутника 3: 7.25
Площа прямокутника 3: 25.737
```

Рис. 1. Результат.

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

Листинг програми:

```
import math

def hypotenuse(a, b):
    return math.sqrt(a**2 + b**2)

cathetus1 = float(input("Введіть перший катет для першого трикутника: "))
cathetus2 = float(input("Введіть другий катет для першого трикутника: "))
hyp1 = round(hypotenuse(cathetus1, cathetus2), 3)

cathetus3 = float(input("Введіть перший катет для другого трикутника: "))
cathetus4 = float(input("Введіть другий катет для другого трикутника: "))
hyp2 = round(hypotenuse(cathetus3, cathetus4), 3)

print(f"Гіпотенуза першого трикутника: {hyp1}")
print(f"Гіпотенуза другого трикутника: {hyp2}")
```

```
if hyp1 > hyp2:
    print("Гіпотенуза першого трикутника більша.")
elif hyp1 < hyp2:
    print("Гіпотенуза другого трикутника більша.")
else:
    print("Гіпотенузи однакові.")
```

Результат виконання:

```
Введіть перший катет для першого трикутника: 13
Введіть другий катет для першого трикутника: 7.5
Введіть перший катет для другого трикутника: 12.5
Введіть другий катет для другого трикутника: 13
Гіпотенуза першого трикутника: 15.008
Гіпотенуза другого трикутника: 18.035
Гіпотенуза другого трикутника більша.
```

Рис. 2. Результат.

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.

Листинг програми:

```
def is_inside_circle(x, y, a, b, R):
    return (x - a)**2 + (y - b)**2 < R**2

a = float(input("Введіть координату a центру кола: "))
b = float(input("Введіть координату b центру кола: "))
R = float(input("Введіть радіус кола R: "))

points = [
    ('P', float(input("Введіть p1: ")), float(input("Введіть p2: "))),
    ('F', float(input("Введіть f1: ")), float(input("Введіть f2: "))),
    ('L', float(input("Введіть l1: ")), float(input("Введіть l2: ")))
]

count_inside = 0
for name, x, y in points:
    if is_inside_circle(x, y, a, b, R):
        print(f"Точка {name} знаходиться всередині кола")
        count_inside += 1
    else:
        print(f"Точка {name} не знаходиться всередині кола")

print(f"Кількість точок, що лежать всередині кола: {count_inside}")
```

Результат виконання:

```
Введіть координату a центру кола: 7
Введіть координату b центру кола: 4
Введіть радіус кола R: 6
Введіть p1: 13
Введіть p2: 2
Введіть f1: 4
Введіть f2: 4
Введіть l1: -3
Введіть l2: -5
Точка P не знаходиться всередині кола
Точка F знаходиться всередині кола
Точка L не знаходиться всередині кола
Кількість точок, що лежать всередині кола: 1
```

Рис. 3. Результат.

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

Листинг програми:

```
import math


def calculate_quadrilateral_area(X, Y, Z, T):
    # Площа першого трикутника зі сторонами X і Y (прямокутний трикутник)
    area1 = 0.5 * X * Y
    # Обчислення діагоналі між точками X і Y за теоремою Піфагора
    diagonal = math.sqrt(X ** 2 + Y ** 2)

    # Обчислення площі другого трикутника за формулою Герона
    s = (diagonal + Z + T) / 2  # Півпериметр трикутника
    area2 = math.sqrt(s * (s - diagonal) * (s - Z) * (s - T))
    # Загальна площа чотирикутника
    total_area = area1 + area2
    return total_area

X = float(input("Введіть довжину X: "))
Y = float(input("Введіть довжину Y: "))
Z = float(input("Введіть довжину Z: "))
T = float(input("Введіть довжину T: "))

print(f"Площа чотирикутника: {round(calculate_quadrilateral_area(X, Y, Z, T),
3)}")
```

Результат виконання:

```
Введіть довжину X: 6
Введіть довжину Y: 13
Введіть довжину Z: 5
Введіть довжину T: 12
Площа чотирикутника: 67.618
```

Рис. 4. Результат.

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

Листинг програми:

```
def find_divisible_numbers(n, divisors):
    return [i for i in range(1, n + 1) if all(i % d == 0 for d in divisors)]

n = int(input("Введіть число n: "))
divisors = list(map(int, input("Введіть числа, на які повинні ділитися знайдені
числа, через пробіл: ").split()))

result = find_divisible_numbers(n, divisors)
print(f"Натуральні числа, що не перевищують {n} і діляться на всі задані числа:
{result}")
```

Результат виконання:

```
Введіть число n: 13
Введіть числа, на які повинні ділитися знайдені числа, через пробіл: 2 4 6
Натуральні числа, що не перевищують 13 і діляться на всі задані числа: [12]
```

Рис. 5. Результат.

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

Листинг програми:

```
def count_divisors(num):
    divisors = 0
    for i in range(1, num + 1):
        if num % i == 0:
            divisors += 1
    return divisors

def find_number_with_max_divisors(M, N):
    max_divisors = 0
    numbers = []
    for i in range(M, N + 1):
        divisors = count_divisors(i)
        if divisors > max_divisors:
            max_divisors = divisors
```

```
            numbers = [i]
        elif divisors == max_divisors:
            numbers.append(i)
    return numbers, max_divisors

M = int(input("Введіть початок інтервалу M: "))
N = int(input("Введіть кінець інтервалу N: "))

numbers, max_divisors = find_number_with_max_divisors(M, N)
print("Числа з найбільшою кількістю дільників:", numbers)
print(f"Кількість дільників: {max_divisors}")
```

Результат виконання:

```
Введіть початок інтервалу M: 3
Введіть кінець інтервалу N: 19
Числа з найбільшою кількістю дільників: [12, 18]
Кількість дільників: 6
```

Рис. 6. Результат.

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.

Листинг програми:

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def find_primes(N, format_type='list'):
    primes = [i for i in range(2, N + 1) if is_prime(i)]

    if format_type == 'list':
        print(" ".join(map(str, primes)))
    elif format_type == 'column':
        for prime in primes:
            print(prime)
    elif format_type == 'count':
        print(f"Кількість простих чисел: {len(primes)}")

N = int(input("Пошук всіх простих чисел від 0 до "))
format_type = input("Виберіть формат виведення результату (list/column/count): ")
find_primes(N, format_type)
```

Результат виконання:

```
Пошук всіх простих чисел від 0 до 212
Виберіть формат виведення результату (list/column/count): list
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211
```

Рис. 7. Результат.

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.

Листинг програми:

```
def filter_list(nums, bottom, upper):
    try:
        MIN = min(nums)
        MAX = max(nums)
        result = [num for num in nums if MIN + bottom <= num <= MAX - upper]
        return result
    except ValueError:
        print("Список не може бути порожнім")
        return []

try:
    nums = list(map(int, input("Введіть список чисел (через пробіл): ").split()))
    bottom = int(input("Введіть нижню межу bottom: "))
    upper = int(input("Введіть верхню межу upper: "))

    filtered_nums = filter_list(nums, bottom, upper)
    print("Отриманий відфільтрований список:", filtered_nums)
except ValueError:
    print("Введено неправильні дані")
```

Результат виконання:

```
Введіть список чисел (через пробіл): 13 -3 6 4 55 -17 4
Введіть нижню межу bottom: -3
Введіть верхню межу upper: 25
Отриманий відфільтрований список: [13, -3, 6, 4, -17, 4]
```

Рис. 8. Результат.

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.

Листинг програми:

```
import time
import random
import math

# Декоратор для вимірювання часу виконання
def timer(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Час виконання {func.__name__}: {end_time - start_time} секунд")
        return result
    return wrapper

# Завдання 6: Пошук числа з найбільшою кількістю дільників
def count_divisors(num):
    divisors = 0
    for i in range(1, int(math.sqrt(num)) + 1):
        if num % i == 0:
            divisors += 1
            if i != num // i:
                divisors += 1
    return divisors

@timer
def find_number_with_max_divisors(M, N):
    max_divisors = 0
    numbers = []
    for i in range(M, N + 1):
        divisors = count_divisors(i)
        if divisors > max_divisors:
            max_divisors = divisors
            numbers = [i]
        elif divisors == max_divisors:
            numbers.append(i)
    return numbers, max_divisors

# Завдання 7: Пошук простих чисел
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

@timer
def find_primes(N, format_type='list'):
    primes = [i for i in range(2, N + 1) if is_prime(i)]
```

```
    if format_type == 'list':
        print(" ".join(map(str, primes)))
    elif format_type == 'column':
        for prime in primes:
            print(prime)
    elif format_type == 'count':
        print(f"Кількість простих чисел: {len(primes)}")

# Завдання 8: Фільтрація списку чисел
@timer
def filter_list(nums, bottom, upper):
    try:
        MIN = min(nums)
        MAX = max(nums)
        result = [num for num in nums if MIN + bottom <= num <= MAX - upper]
        return result
    except ValueError:
        print("Список не може бути порожнім")
        return []

# Тестування для різних значень n
for n in range(1, 7):
    size = 10 ** n
    print(f"Тестування для {size} елементів:")

    # Тест для завдання 6
    M = 1
    N = size
    print("Завдання 6:")
    find_number_with_max_divisors(M, N)

    # Тест для завдання 7
    print("Завдання 7:")
    find_primes(size, 'count')

    # Тест для завдання 8
    nums = random.sample(range(1, size + 1), size)
    bottom, upper = random.randint(1, size // 10), random.randint(1, size // 10)
    print("Завдання 8:")
    filter_list(nums, bottom, upper)

    print("=" * 50)
```

Результат виконання:

```
Тестування для 10 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 0.0 секунд
Завдання 7:
Кількість простих чисел: 4
Час виконання find_primes: 0.0 секунд
Завдання 8:
Час виконання filter_list: 0.0 секунд
==================================================
Тестування для 100 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 0.0 секунд
Завдання 7:
Кількість простих чисел: 25
Час виконання find_primes: 0.0 секунд
Завдання 8:
Час виконання filter_list: 0.0 секунд
==================================================
Тестування для 1000 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 0.0019998550415039062 секунд
Завдання 7:
Кількість простих чисел: 168
Час виконання find_primes: 0.0 секунд
Завдання 8:
Час виконання filter_list: 0.0 секунд
==================================================
Тестування для 10000 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 0.027532577514648438 секунд
Завдання 7:
Кількість простих чисел: 1229
Час виконання find_primes: 0.005998849868774414 секунд
Завдання 8:
Час виконання filter_list: 0.001003265380859375 секунд
==================================================
```

```
Тестування для 100000 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 0.7627289295196533 секунд
Завдання 7:
Кількість простих чисел: 9592
Час виконання find_primes: 0.11689281463623047 секунд
Завдання 8:
Час виконання filter_list: 0.009716987609863281 секунд
==================================================
Тестування для 1000000 елементів:
Завдання 6:
Час виконання find_number_with_max_divisors: 27.120696306228638 секунд
Завдання 7:
Кількість простих чисел: 78498
Час виконання find_primes: 2.8946728706359863 секунд
Завдання 8:
Час виконання filter_list: 0.10713028907775879 секунд
==================================================
```

Рис. 9. Результат.

Висновок: У цій лабораторній роботі ми ознайомились з функціями в мові Python.

***Посилання на Git**: : https://github.com/JanRizhenko/Python_Labs*