Міністерство освіти і науки України

Державний університет “Житомирська політехніка”

Кафедра інженерії програмного забезпечення

Група: ВТ-21-1[1]

Програмування мовою Python

Лабораторна робота № 8

«КЛАСИ. Ч. 2»

Виконав: Бабушко А. С.

Прийняв: Морозов Д. С.

					«Житомирська політехніка».22.121.01.000–Лр8		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Бабушко А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 27
Перевір.		Морозов Д.С.					
Керівник						ФІКТ Гр. ВТ-21-1[1]	
Н. контр.							
Затверд.							

Мета роботи: ознайомитися з ООП в мові Python

Хід роботи:

Завдання на лабораторну роботу:

Напишіть програми у середовищі програмування для розв'язування таких завдань:

1. Напишіть клас Bank для опису простих операції з вашим банківським рахунком: покласти на рахунок, зняти з рахунку, переглянути рахунок. При створенні екземпляру класу, екземпляр отримує атрибут __balance з певним значенням. Клас повинен містити методи для додавання коштів на рахунок і знімання з рахунку, за умови, що на рахунку достатньо коштів.

Лістинг програми:

```
class Bank:
    def __init__(self, balance: float):
        self.__balance = balance

    def deposit_money(self, deposit_money):
        self.__balance += deposit_money

    def withdraw_money(self, withdrawal_money):
        if self.__balance - withdrawal_money < 0:
            print('Not enough money!')
        else:
            self.__balance -= withdrawal_money

    def __str__(self):
        return f'You have {self.__balance} UAH!'

    def get_balance(self):
        return self.__balance


print('TASK 1!!!!')
task_1_bank: Bank = Bank(1000)
print(task_1_bank)
print('We are adding to your bank 500 UAH!')
task_1_bank.deposit_money(500)
print(f'Now you have {task_1_bank.get_balance()} UAH!')
print('Try to withdraw 1600 UAH!')
task_1_bank.withdraw_money(1600)
print('Try to withdraw 500 UAH!')
task_1_bank.withdraw_money(500)
print(f'Now you have {task_1_bank.get_balance()} UAH!')
```

Результат програми:

```
TASK 1!!!
You have 1000 UAH!
We are adding to your bank 500 UAH!
Now you have 1500 UAH!
Try to withdraw 1600 UAH!
Not enough money!
Try to withdraw 500 UAH!
Now you have 1000 UAH!
```

2. Напишіть клас Coin, який описує монету, яку можна підкидати. При створенні екземпляру класу, екземпляр отримує атрибут __sideup зі значенням heads або tails. У класі визначте метод toss, який випадково визначає результат підкидання монети - орел чи решка. Створіть екземпляр класу і виведіть на екран n підкидань монети.

Лістинг програми:

```
class Coin:
    def __init__(self):
        self.__side_up = 'Heads'

    def toss(self):
        from random import randint
        rand_side: int = randint(0, 1)
        if rand_side == 1:
            print('Tossed heads!')
            self.__side_up = 'Heads'
        else:
            print('Tossed tails!')
            self.__side_up = 'Tails'


print('\nTASK 2!!!')
task_2_coin: Coin = Coin()
for tossed in range(0, 5):
    print(f'{tossed + 1} toss...')
    task_2_coin.toss()
```

Результат програми:

```
TASK 2!!!
1 toss...
Tossed heads!
2 toss...
Tossed tails!
3 toss...
Tossed tails!
4 toss...
Tossed heads!
5 toss...
Tossed heads!
```

3. Напишіть клас Car, який надає для створених екземплярів такі атрибути даних автомобіля: марку виготовлення автомобіля, модель автомобіля, рік автомобіля, швидкість (початкове значення 0). Клас також повинен мати наступні методи: accelerate (метод повинен щоразу додавати 5 до значення атрибуту даних про швидкість), brake (метод повинен віднімати 5 від значення атрибута даних швидкості кожного разу, коли він викликається), get_speed (метод повинен повернути поточну швидкість). Створіть екземпляр класу Car і викличте метод accelerate п’ять разів. Після кожного виклику методу accelerate отримайте поточну швидкість автомобіля і надрукуйте її значення. Потім викличте метод brake п’ять разів. Після кожного виклику методу brake отримайте поточну швидкість автомобіля та надрукуйте її значення.

Лістинг програми:

```
class Car:
    def __init__(self, mark: str, model: str, year: str, speed: int = 0):
        self.__car_mark: str = mark
        self.__car_model: str = model
        self.__car_year: str = year
        self.__car_speed: int = speed

    def accelerate(self):
        self.__car_speed += 5

    def brake(self):
        if self.__car_speed > 0:
            self.__car_speed -= 5

    def get_speed(self):
        return self.__car_speed


print('\nTASK 3!!!')
task_3_car: Car = Car('Chevrolet', 'Corvette 2020(C8)', '2020')
print('Accelerating car...')
for accelerate in range(0, 5):
    task_3_car.accelerate()
    print(f'Current car speed: {task_3_car.get_speed()}')
print('Braking car...')
for brake in range(0, 5):
    task_3_car.brake()
    print(f'Current car speed: {task_3_car.get_speed()}')
```

Результат програми:

```
TASK 3!!!
Accelerating car...
Current car speed: 5
Current car speed: 10
Current car speed: 15
Current car speed: 20
Current car speed: 25
Braking car...
Current car speed: 20
Current car speed: 15
Current car speed: 10
Current car speed: 5
Current car speed: 0
```

4. Напишіть клас Dog, який має три атрибути класу: mammal (ссавець), nature (характер) і breed (порода), та два атрибути екземпляра: name (кличка) і age (вік). Створіть екземпляри трьох нових собак, кожна з яких різного віку. Визначте у класі Dog метод для виведення значень атрибутів екземпляру - імені та віку конкретної собаки. За потреби, додайте кілька інших методів, які визначають поведінку собаки (подавання голосу тощо). Напишіть кілька класів, які унаслідуються від батьківського класу Dog, що описують конкретні породи собак. Визначте для цих класів атрибути nature і breed відповідно, включіть у класи по одному методу, що визначає поведінку конкретної породи собаки. Створіть батьківський клас Pets, що створює список ваших домашніх улюбленців. У підсумку, надрукуйте інформацію про ваших домашніх тварин, на зразок, як у вихідних даних.

Лістинг програми:

```python
class Dog:
    __mammal: bool
    __nature: str
    __breed: str

    def __init__(self, name: str, age: int):
        self.__name: str = name
        self.__age: int = age

    def __str__(self):
        return f'Dog name: {self.__name}. Dog age: {self.__age}.'

    def voice(self):
        if self.__age < 2:
            return 'Tyaf!'
        elif 2 <= self.__age < 5:
            return 'Wafh!'
        else:
            return 'GAF!'
```

```
class Akita(Dog):
    __mammal = True
    __nature = 'The Akita is generally seen as territorial about its property, and can be
reserved with strangers.'
    __breed = 'Akita'

    def __str__(self):
        return f'Mammal: {self.__mammal}.\nNature: {self.__nature}.\nBreed:
{self.__breed}.'

    @staticmethod
    def temperament():
        return 'Akita is here! It\'s my territory. GAF! I\'m cleaning my face after
eating!'


class Doberman(Dog):
    __mammal = True
    __nature = 'Doberman are considered to be working dogs and often stereotyped as being
ferocious and aggressive.'
    __breed = 'Doberman'

    def __str__(self):
        return f'Mammal: {self.__mammal}.\nNature: {self.__nature}.\nBreed:
{self.__breed}.'

    @staticmethod
    def temperament():
        return 'GAFFFF!!!!! GAF! GAF! GAF!I\'m Doberman and it\'s my owner, bi*ches!'


class Pets:
    def __init__(self):
        self._pets_list: list = []

    def add_pet(self, pet):
        self._pets_list.append(pet)

    def remove_pet(self, pet):
        self._pets_list.remove(pet)

    @property
    def pets_list(self):
        return self._pets_list


print('\nTASK 4!!!')
task_4_first_dog: Akita = Akita('Aki', 4)
task_4_second_dog: Doberman = Doberman('Loli', 2)
task_4_third_dog: Dog = Dog('Poppy', 7)
task_4_pets: Pets = Pets()
task_4_pets.add_pet(task_4_first_dog)
task_4_pets.add_pet(task_4_second_dog)
task_4_pets.add_pet(task_4_third_dog)
counter: int = 1
for pet in task_4_pets.pets_list:
    print(f'{counter} dog:\n{pet}')
    counter += 1
```

Результат програми:

```
TASK 4!!!
1 dog:
Mammal: True.
Nature: The Akita is generally seen as territorial about its property, and can be reserved with strangers..
Breed: Akita.
2 dog:
Mammal: True.
Nature: Doberman are considered to be working dogs and often stereotyped as being ferocious and aggressive..
Breed: Doberman.
3 dog:
Dog name: Poppy. Dog age: 7.
```

5. Дано послідовність цілих чисел. Необхідно її обробити і вивести на екран суму першої п’ятірки чисел із цієї послідовності, потім суму другої п’ятірки, і т. д. Але послідовність не дається відразу загалом. З плином часу до вас надходять її послідовні частини. Наприклад, спочатку перші три елементи, потім наступні шість, потім наступні два і т. д. Реалізуйте клас Buffer, який буде накопичувати в собі елементи послідовності і виводити суму п’ятірок послідовних елементів у міру їх накопичення. Однією з вимог до класу є те, що він не повинен зберігати в собі більше елементів, ніж йому дійсно необхідно, тобто, він не повинен зберігати елементи, які вже увійшли в п’ятірку, для якої була виведена сума. Клас повинен мати наступний вигляд

```
class Buffer:
    def __init__(self):
        # конструктор без аргументів

    def add(self, *a):
        # додати наступну частину послідовності

    def get_current_part(self):
        # повернути збережені в поточний момент часу елементи
послідовності в порядку, в якому вони були додані
```

Зверніть увагу, що під час виконання методу add виводити суму п’ятірок може знадобитися кілька разів до тих пір, поки в буфері не залишиться менше п’яти елементів.

Лістинг програми:

```
# task 5
"""
5. Дано послідовність цілих чисел. Необхідно її обробити і вивести на екран суму першої
п’ятірки чисел із цієї послідовності, потім суму другої п’ятірки, і т. д. Але послідовність
    не дається відразу загалом. З плином часу до вас надходять її послідовні частини.
Наприклад, спочатку перші три елементи, потім наступні шість, потім наступні два і т. д.
    Реалізуйте клас Buffer, який буде накопичувати в собі елементи послідовності і виводити
суму п’ятірок послідовних елементів у міру їх накопичення. Однією з вимог до класу є те,
    що він не повинен зберігати в собі більше елементів, ніж йому дійсно необхідно, тобто,
він не повинен зберігати елементи, які вже увійшли в п’ятірку, для якої була виведена
    сума. Зверніть увагу, що під час виконання методу add виводити суму п’ятірок може
знадобитися кілька разів до тих пір, поки в буфері не залишиться менше п’яти елементів.
```

```
"""


class Buffer:
    def __init__(self, *data: float):
        self.__sequence: list = [*data]

    def __str__(self):
        return f'Sequence: {self.__sequence}'

    def add_elems_to_sequence(self, *data: float):
        for arg in data:
            self.__sequence.append(arg)

    def get_sum_list_5_elems(self):
        sum_list = []

        last_counter = 0
        for counter in range(5, len(self.__sequence), 5):
            float_sum = 0
            for index in range(last_counter, counter):
                float_sum += self.__sequence[index]
            last_counter = counter
            sum_list.append(float_sum)

        return sum_list

    def get_sequence(self):
        return self.__sequence

    def get_sequence_length(self):
        return len(self.__sequence)


print('\nTASK 5!!!')
task_5_sequence: Buffer = Buffer(1, 2, 3, 4, 5, 1, 1, 2)
print(f'Current Buffer sequence: {task_5_sequence.get_sequence()}')
print(f'Sum list of every 5 elems of sequence: {task_5_sequence.get_sum_list_5_elems()}')
task_5_sequence.add_elems_to_sequence(1, 1, 3, 10, -10, -20)
print(f'Current Buffer sequence: {task_5_sequence.get_sequence()}')
print(f'Sum list of every 5 elems of sequence: {task_5_sequence.get_sum_list_5_elems()}')
```

Результат програми:

```
TASK 5!!!
Current Buffer sequence: [1, 2, 3, 4, 5, 1, 1, 2]
Sum list of every 5 elems of sequence: [15]
Current Buffer sequence: [1, 2, 3, 4, 5, 1, 1, 2, 1, 1, 3, 10, -10, -20]
Sum list of every 5 elems of sequence: [15, 6]
```

6. Напишіть клас-виняток, на основі вбудованого в Python класу ValueError(). Клас буде представляти перевірку певного імені на основі його довжини. Якщо довжина введеного імені є меншою 10, то має генеруватися виняток як у вихідних даних. У інших випадках нічого не виводиться.

Лістинг програми:

```
# task 6
"""
    6. Напишіть клас-виняток, на основі вбудованого в Python класу ValueError(). Клас буде
представляти перевірку певного імені на основі його довжини. Якщо довжина введеного
    імені є меншою 10, то має генеруватися виняток як у вихідних даних. У інших випадках
нічого не виводиться.
"""
```

```
class FullNameError(Exception):
    def __init__(self, value: str):
        self.value: str = value


class Name:
    def __init__(self, first_name: str, last_name: str, middle_name: str):
        self.__first_name: str = first_name
        self.__last_name: str = last_name
        self.__middle_name: str = middle_name
        self.__full_name = last_name + ' ' + first_name + ' ' + middle_name

    def __str__(self):
        return f'First name -> {self.__first_name} Last name -> {self.__last_name} Middle
name -> {self.__middle_name}'

    def check_entered_name(self):
        try:
            if len(self.__full_name) < 10:
                raise FullNameError('Name length is less than 10!')
            else:
                pass
                print('Name was entered correctly!')
        except FullNameError as name_error:
            self.__first_name: str = ''
            self.__last_name: str = ''
            self.__middle_name: str = ''
            print(name_error)


print('\nTASK 6!!!')
task_5_name_1: Name = Name('Andrii', 'Babushko', 'Sergiyovich')
print(f'Data of task_5_name_1 instance of a Name class:\n{task_5_name_1}')
task_5_name_1.check_entered_name()
task_5_name_2: Name = Name('A', 'B', 'S')
print(f'\nData of task_5_name_2 instance of a Name class:\n{task_5_name_2}')
task_5_name_2.check_entered_name()
```

Результат програми:

```
TASK 6!!!
Data of task_5_name_1 instance of a Name class:
First name -> Andrii Last name -> Babushko Middle name -> Sergiyovich
Name was entered correctly!


Data of task_5_name_2 instance of a Name class:
First name -> A Last name -> B Middle name -> S
Name length is less than 10!
```

7. Напишіть один клас для перетворення десяткового числа на число в римській системі числення. І ще один клас для перетворення числа з римської системи числення у десяткове число.

Лістинг програми:

```
# task 7
"""
    7. Напишіть один клас для перетворення десяткового числа на число в римській системі
числення. І ще один клас для перетворення числа з римської системи числення у десяткове
    число.
"""
```

```
class DecimalToRoman:
    __roman_number: str

    def __init__(self, decimal: int):
        self.__decimal_number: int = decimal

    def __str__(self):
        return f'Decimal number: {self.__decimal_number}; Roman: {self.__roman_number};'

    def move_decimal_to_roman(self):
        values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
        symbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"]

        i = 0
        number = self.__decimal_number
        roman_number: str = ''

        while number > 0:
            for interator in range(number // values[i]):
                roman_number += symbols[i]
                number -= values[i]
            i += 1

        self.__roman_number = roman_number
        return roman_number

    def set_decimal(self, decimal: int):
        self.__decimal_number: int = decimal

    def get_roman_number(self):
        return self.__roman_number

    def get_decimal_number(self):
        return self.__decimal_number


class RomanToDecimal:
    __decimal_number: int

    def __init__(self, roman: str):
        self.__roman_number: str = roman

    def __str__(self):
        return f'Roman: {self.__roman_number}; Decimal number: {self.__decimal_number};'

    def move_roman_to_decimal(self):
        roman_values = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
        decimal_value = 0

        for i in range(len(self.__roman_number)):
            if i > 0 and roman_values[self.__roman_number[i]] >
roman_values[self.__roman_number[i - 1]]:
                decimal_value += roman_values[self.__roman_number[i]] - 2 *
roman_values[self.__roman_number[i - 1]]
            else:
                decimal_value += roman_values[self.__roman_number[i]]

        return decimal_value

    def set_roman(self, roman: str):
        self.__roman_number: str = roman

    def get_roman_number(self):
        return self.__roman_number

    def get_decimal_number(self):
        return self.__decimal_number
```

```
print('\nTASK 7!!!')
task_7_decimal_to_roman: DecimalToRoman = DecimalToRoman(15)
print(f'Move {task_7_decimal_to_roman.get_decimal_number()} decimal number to roman number:
{task_7_decimal_to_roman.move_decimal_to_roman()}')
task_7_roman_to_decimal: RomanToDecimal =
RomanToDecimal(task_7_decimal_to_roman.get_roman_number())
print(f'Move {task_7_roman_to_decimal.get_roman_number()} roman number to decimal number:
{task_7_roman_to_decimal.move_roman_to_decimal()}')
```

Результат програми:

```
TASK 7!!!
Move 15 decimal number to roman number: XV
Move XV roman number to decimal number: 15
```

8. Онлайн-магазин

   a. Створіть клас з ім'ям Shop(). Клас Shop() повинен містити два атрибути: shop_name і store_type. Створіть метод describe_shop(), який виводить два атрибути, і метод open_shop(), який виводить повідомлення про те, що онлайн-магазин відкритий. Створіть на основі класу екземпляр з ім'ям store. Виведіть два атрибути окремо, потім викличте обидва методи.

   b. Створіть три різних екземпляри класу, викличте для кожного екземпляру метод describe_shop().

   c. Додайте атрибут number_of_units зі значенням за замовчуванням 0; він представляє кількість видів товару у магазині. Створіть екземпляр з ім'ям store. Виведіть значення number_of_units, а потім змініть number_of_units і виведіть знову.

   d. Додайте метод з ім'ям set_number_of_units(), що дозволяє задати кількість видів товару. Викличте метод з новим числом, знову виведіть значення. Додайте метод з ім'ям increment_number_of_units(), який збільшує кількість видів товару на задану величину. Викличте цей метод.

   e. Напишіть клас Discount(), що успадковує від класу Shop(). Додайте атрибут з ім'ям discount_products для зберігання списку товарів, на які встановлена знижка. Напишіть метод get_discounts_ptoducts, який виводить цей список. Створіть екземпляр store_discount і викличте цей метод.

   f. Збережіть код класу Shop() у модулі. Створіть окремий файл, що імпортує клас Shop(). Створіть екземпляр all_store і викличте один з методів Shop(), щоб перевірити, що команда import працює правильно.

Лістинг програми:

1. ***shop.py:***

```python
class Shop:
    def __init__(self, shop_name: str, shop_type: str, shop_number_of_units: int = 0):
        self.name: str = shop_name
        self.type: str = shop_type
        self.number_of_units: int = shop_number_of_units

    def describe_shop(self):
        print(f'Shop name: {self.name}; Shop type: {self.type};')

    @staticmethod
    def open_shop():
        print('Online shop is opened!')

    def set_number_of_units(self, number_of_units: int):
        self.number_of_units: int = number_of_units

    def increment_number_of_units(self, increment: int):
        if increment > 0:
            self.number_of_units += increment
```

2. *lab8.py:*

```python
# task 8
"""
    8. Онлайн-магазин
a. Створіть клас з ім'ям Shop(). Клас Shop() повинен містити два атрибути: shop_name і
store_type. Створіть метод describe_shop(), який виводить два атрибути, і метод
open_shop(),
який виводить повідомлення про те, що онлайн-магазин відкритий. Створіть на основі класу
екземпляр з ім'ям store. Виведіть два атрибути окремо, потім викличте обидва методи.
b. Створіть три різних екземпляри класу, викличте для кожного екземпляру метод
describe_shop().
c. Додайте атрибут number_of_units зі значенням за замовчуванням 0; він представляє
кількість видів товару у магазині. Створіть екземпляр з ім'ям store. Виведіть значення
number_of_units, а потім змініть number_of_units і виведіть знову.
d. Додайте метод з ім'ям set_number_of_units(), що дозволяє задати кількість видів товару.
Викличте метод з новим числом, знову виведіть значення. Додайте метод з ім'ям
increment_number_of_units(), який збільшує кількість видів товару на задану величину.
Викличте цей метод.
e. Напишіть клас Discount(), що успадковує від класу Shop(). Додайте атрибут з ім'ям
discount_products для зберігання списку товарів, на які встановлена знижка. Напишіть метод
get_discounts_ptoducts, який виводить цей список. Створіть екземпляр store_discount і
викличте цей метод.
f. Збережіть код класу Shop() у модулі. Створіть окремий файл, що імпортує клас Shop().
Створіть екземпляр all_store і викличте один з методів Shop(), щоб перевірити, що команда
import працює правильно.
"""

from modules import shop

print('\nTASK 8!!!')
print('a)')
task_8_store: shop.Shop = shop.Shop('Store', 'stuff store')
print(f'Print separately shop name: {task_8_store.name}')
print(f'Print separately shop type: {task_8_store.type}')
print('Shop status:')
task_8_store.open_shop()
print('Describe shop:')
task_8_store.describe_shop()

print('\nb)')
task_8_raccoons_tech_shop: shop.Shop = shop.Shop('Tech Raccoons', 'tech shop')
print('Describe shop:')
task_8_raccoons_tech_shop.describe_shop()
task_8_raccoons_sex_shop: shop.Shop = shop.Shop('Sexy Raccoons', 'sex shop')
print('Describe shop:')
task_8_raccoons_sex_shop.describe_shop()
task_8_raccoons_food_shop: shop.Shop = shop.Shop('Raccoon\'s Food Market', 'food market')
print('Describe shop:')
task_8_raccoons_food_shop.describe_shop()
```

```
print('\nc)')
print(f'Current store\'s number of units: {task_8_store.number_of_units}')
task_8_store.number_of_units = 12
print(f'Changed store\'s number of units: {task_8_store.number_of_units}')

print('\nd)')
task_8_store.set_number_of_units(10)
print(f'Set store\'s number of units by 10: {task_8_store.number_of_units}')
task_8_store.increment_number_of_units(10)
print(f'Incremented store\'s number of units by 10: {task_8_store.number_of_units}')


class Discount(shop.Shop):
    def __init__(self, shop_name: str, shop_type: str, **kwargs):
        super().__init__(shop_name, shop_type)
        self.discount_products: dict = kwargs

    def get_discounts_products(self):
        discount_keys: list = list(self.discount_products.keys())
        discount_values: list = list(self.discount_products.values())

        print(f'Current discounts:')
        for iterator in range(0, len(self.discount_products)):
            print(f'{discount_keys[iterator]}: {discount_values[iterator]}')


print('\ne)')
discounts: dict = {'Car Toy': '10%', 'Doll toy': '20%', 'Mobile phone': '15%'}
task_8_discount_store: Discount = Discount('Store', 'stuff store', Car_Toy='10%',
Doll_toy='20%', Mobile_phone='15%')
task_8_discount_store.get_discounts_products()

print('\nf)')
task_8_all_store: shop.Shop = shop.Shop('All  store', 'store')
print(f'Describe all store:')
task_8_all_store.describe_shop()
task_8_all_store.set_number_of_units(20)
print(f'Set all store\'s number of units by 20: {task_8_all_store.number_of_units}')
task_8_all_store.increment_number_of_units(30)
print(f'Incremented store\'s number of units by 30: {task_8_all_store.number_of_units}')
```

Результат програми:

```
TASK 8!!!
a)
Print separately shop name: Store
Print separately shop type: stuff store
Shop status:
Online shop is opened!
Shop name: Store; Shop type: stuff store;
Describe shop: None

b)
Describe shop:
Shop name: Tech Raccoons; Shop type: tech shop;
Describe shop:
Shop name: Sexy Raccoons; Shop type: sex shop;
Describe shop:
Shop name: Raccoon's Food Market; Shop type: food market;

c)
Current store's number of units: 0
Changed store's number of units: 12

d)
Set store's number of units by 10: 10
Incremented store's number of units by 10: 20

e)
Current discounts:
Car_Toy: 10%
Doll_toy: 20%
Mobile_phone: 15%

f)
Describe all store:
Shop name: All  store; Shop type: store;
Set all store's number of units by 20: 20
Incremented store's number of units by 30: 50
```

9. Облік користувачів на сайті

   a. Створіть клас з ім’ям User. Створіть два атрибути first_name і last_name, а потім ще кілька атрибутів, які зазвичай зберігаються у профілі користувача (поштова адреса, нікнейм, що відображається на сайті, згода на розсилку новин з форуму). Напишіть метод describe_user який виводить повне ім’я користувача. Створіть ще один метод greeting_user() для виведення персонального вітання для користувача. Створіть кілька примірників, які представляють різних користувачів. Викличте обидва методи для кожного користувача.

   b. Додайте атрибут login_attempts у клас User. Напишіть метод increment_login_attempts(), що збільшує значення login_attempts на 1. Напишіть інший метод з ім’ям reset_login_attempts(), обнуляє значення login_attempts. Створіть екземпляр класу User і викличте increment_login_attempts() кілька разів. Виведіть значення login_attempts, щоб переконатися у тому, що значення було змінено правильно, а потім викличте reset_login_attempts(). Знову виведіть login_attempts і переконайтеся у тому, що значення обнулилося

   c. Адміністратор - користувач з повними адміністративними привілеями. Напишіть клас з ім’ям Admin, що успадковує від класу User. Додайте атрибут privileges для зберігання списку рядків виду «Allowed to add message», «Allowed to delete users», «Allowed to ban users» і т. д. Напишіть метод show_privileges() для виведення набору привілеїв адміністратора. Створіть екземпляр Admin і викличте метод.

   d. Напишіть клас Privileges. Клас повинен містити всього один атрибут privileges зі списком, який треба забрати із класу Admin. Водночас, необхідно перемістити метод show_privileges() у клас Privileges із класу Admin. Створіть екземпляр priv як атрибут класу Admin. Створіть новий екземпляр admin і використайте метод для виведення списку привілеїв.

   e. Збережіть клас User в одному модулі, а класи Privileges і Admin у іншому модулі. В окремому файлі створіть екземпляр admin і викличте метод show_privileges(), щоб перевірити, що все працює правильно.

Лістинг програми:

1. ***user.py:***

```
class User:
    def __init__(self, first_name: str, last_name: str, email: str, nickname: str,
mailing_consent: bool, login_attempts: int = 0):
        self.first_name: str = first_name
        self.last_name: str = last_name
        self.full_name: str = last_name + ' ' + first_name
        self.email: str = email
        self.nickname: str = nickname
        self.mailing_consent: bool = mailing_consent
```

```
        self.login_attempts: int = login_attempts

    def describe_user(self):
        print(f'Full user name: {self.full_name}')

    def greeting_user(self):
        print(f'Our greetings, {self.full_name}!')

    def increment_login_attempts(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0
```

2. admin.py:

```
from privileges import Privileges
from user import User


class Admin(User):
    def __init__(self, first_name: str, last_name: str, email: str, nickname: str,
mailing_consent: bool, admin_privileges: list):
        super().__init__(first_name, last_name, email, nickname, mailing_consent)
        self.admin_privileges: Privileges = Privileges(admin_privileges)

    def show_privileges(self):
        print(f'{self.full_name} admin\'s privileges:')
        for privilege in self.admin_privileges.privileges:
            print('* ' + privilege)
```

3. privileges.py:

```
class Privileges:
    def __init__(self, privileges: list):
        self.privileges = privileges

    def show_privileges(self):
        print('Privileges:')
        for privilege in self.privileges:
            print('* ' + privilege)
```

4. lab8.py:

```
# task 9
"""
    9. Облік користувачів на сайті
a. Створіть клас з ім’ям User. Створіть два атрибути first_name і last_name, а потім ще
кілька атрибутів, які зазвичай зберігаються у профілі користувача (поштова адреса, нікнейм,
що відображається на сайті, згода на розсилку новин з форуму). Напишіть метод describe_user
який виводить повне ім’я користувача. Створіть ще один метод greeting_user() для
виведення персонального вітання для користувача. Створіть кілька примірників, які
представляють різних користувачів. Викличте обидва методи для кожного користувача.
b. Додайте атрибут login_attempts у клас User. Напишіть метод increment_login_attempts(),
що збільшує значення login_attempts на 1. Напишіть інший метод з ім’ям
reset_login_attempts(), обнуляє значення login_attempts. Створіть екземпляр класу User і
викличте increment_login_attempts() кілька разів. Виведіть значення login_attempts, щоб
переконатися у тому, що значення було змінено правильно, а потім викличте
reset_login_attempts(). Знову виведіть login_attempts і переконайтеся у тому, що значення
обнулилося
c. Адміністратор - користувач з повними адміністративними привілеями. Напишіть клас з ім’ям
Admin, що успадковує від класу User. Додайте атрибут privileges для зберігання списку
рядків виду «Allowed to add message», «Allowed to delete users», «Allowed to ban users» і
т. д. Напишіть метод show_privileges() для виведення набору привілеїв адміністратора.
Створіть екземпляр Admin і викличте метод.
d. Напишіть клас Privileges. Клас повинен містити всього один атрибут privileges зі
списком, який треба забрати із класу Admin. Водночас, необхідно перемістити метод
show_privileges() у клас Privileges із класу Admin. Створіть екземпляр priv як атрибут
класу Admin. Створіть новий екземпляр admin і використайте метод для виведення
списку привілеїв.
e. Збережіть клас User в одному модулі, а класи Privileges і Admin у іншому модулі. В
```

```
окремому файлі створіть екземпляр admin і викличте метод show_privileges(), щоб перевірити,
що все працює правильно.
"""

from modules import user
from modules import admin

print('\nTASK 8!!!')
print('a)')
task_9_1_user: user.User = user.User('Andrii', 'Babushko', 'andriibabushko@gmail.com',
'AndriiRaccoon', True)
task_9_1_user.greeting_user()
task_9_1_user.describe_user()

print('\nb)')
incrementing: int = 5
print(f'Incrementing login attempts 1_user {incrementing} times...')
for iterator in range(0, incrementing):
    task_9_1_user.increment_login_attempts()
print(f'Current login attempts: {task_9_1_user.login_attempts}')
task_9_1_user.reset_login_attempts()
print(f'Login attempts after resetting: {task_9_1_user.login_attempts}')
incrementing: int = 10
print(f'Repeating incrementing login attempts 1_user {incrementing} times...')
for iterator in range(0, incrementing):
    task_9_1_user.increment_login_attempts()
print(f'Current login attempts: {task_9_1_user.login_attempts}')
task_9_1_user.reset_login_attempts()
print(f'Login attempts after resetting: {task_9_1_user.login_attempts}')

print('\nc)')
task_9_1_admin: admin.Admin = admin.Admin('Andrii', 'Babushko', 'andriibabushko@gmail.com',
'AndriiRaccoon', True,
                                          ['Allowed to block users', 'Allowed to delete
messages', 'Allowed to send voices', 'Allowed to delete users'])
task_9_1_admin.show_privileges()

print('\nd)')
print('Show privileges by using class intense inside Admin class:')
task_9_1_admin.admin_privileges.show_privileges()

print('\ne)')
task_9_2_admin: admin.Admin = admin.Admin('Ihor', 'Juice', 'ihorjuice@gmail.com',
'KindOfHell', False,
                                          ['Allowed to block users', 'Allowed to delete
messages', 'Allowed to delete voices', 'Allowed to delete users', 'Allowed to delete app'])
task_9_2_admin.admin_privileges.show_privileges()
```

Результат програми:

```
a)
Our greetings, Babushko Andrii!
Full user name: Babushko Andrii

b)
Incrementing login attempts 1_user 5 times...
Current login attempts: 5
Login attempts after resetting: 0
Repeating incrementing login attempts 1_user 10 times...
Current login attempts: 10
Login attempts after resetting: 0

c)
Babushko Andrii admin's privileges:
* Allowed to block users
* Allowed to delete messages
* Allowed to send voices
* Allowed to delete users

d)
Show privileges by using class intense inside Admin class:
Privileges:
* Allowed to block users
* Allowed to delete messages
* Allowed to send voices
* Allowed to delete users

e)
Privileges:
* Allowed to block users
* Allowed to delete messages
* Allowed to delete voices
* Allowed to delete users
* Allowed to delete app
```

Увесь лістинг програми:

```
""" Lab 8. Python. Andrii Babushko. Repository: https://github.com/AndriiBabushko/Python
"""
import sys

sys.path.insert(0, r'modules')

# task 1
"""
1. Напишіть клас Bank для опису простих операції з вашим банківським рахунком: покласти на
рахунок, зняти з рахунку, переглянути рахунок. При створенні екземпляру класу, екземпляр
    отримує атрибут __balance з певним значенням. Клас повинен містити методи для додавання
коштів на рахунок і знімання з рахунку, за умови, що на рахунку достатньо коштів.
"""


class Bank:
    def __init__(self, balance: float):
        self.__balance = balance

    def deposit_money(self, deposit_money):
        self.__balance += deposit_money

    def withdraw_money(self, withdrawal_money):
        if self.__balance - withdrawal_money < 0:
            print('Not enough money!')
        else:
            self.__balance -= withdrawal_money

    def __str__(self):
        return f'You have {self.__balance} UAH!'

    def get_balance(self):
        return self.__balance


print('TASK 1!!!!')
task_1_bank: Bank = Bank(1000)
print(task_1_bank)
print('We are adding to your bank 500 UAH!')
task_1_bank.deposit_money(500)
print(f'Now you have {task_1_bank.get_balance()} UAH!')
print('Try to withdraw 1600 UAH!')
task_1_bank.withdraw_money(1600)
print('Try to withdraw 500 UAH!')
task_1_bank.withdraw_money(500)
print(f'Now you have {task_1_bank.get_balance()} UAH!')

# task 2
"""
2. Напишіть клас Coin, який описує монету, яку можна підкидати. При створенні екземпляру
класу, екземпляр отримує атрибут __sideup зі значенням heads або tails. У класі визначте
    метод toss, який випадково визначає результат підкидання монети - орел чи решка.
Створіть екземпляр класу і виведіть на екран n підкидань монети.
"""


class Coin:
    def __init__(self):
        self.__side_up = 'Heads'

    def toss(self):
        from random import randint
        rand_side: int = randint(0, 1)
        if rand_side == 1:
            print('Tossed heads!')
            self.__side_up = 'Heads'
        else:
```

```
            print('Tossed tails!')
            self.__side_up = 'Tails'


print('\nTASK 2!!!')
task_2_coin: Coin = Coin()
for tossed in range(0, 5):
    print(f'{tossed + 1} toss...')
    task_2_coin.toss()

# task 3
"""
3. Напишіть клас Car, який надає для створених екземплярів такі атрибути даних автомобіля:
марку виготовлення автомобіля, модель автомобіля, рік автомобіля, швидкість
    (початкове значення 0). Клас також повинен мати наступні методи: accelerate (метод
повинен щоразу додавати 5 до значення атрибуту даних про швидкість),
    brake (метод повинен віднімати 5 від значення атрибута даних швидкості кожного разу,
коли він викликається), get_speed (метод повинен повернути поточну швидкість).
    Створіть екземпляр класу Car і викличте метод accelerate п’ять разів. Після кожного
виклику методу accelerate отримайте поточну швидкість автомобіля і надрукуйте її значення.
    Потім викличте метод brake п’ять разів. Після кожного виклику методу brake отримайте
поточну швидкість автомобіля та надрукуйте її значення.
"""


class Car:
    def __init__(self, mark: str, model: str, year: str, speed: int = 0):
        self.__car_mark: str = mark
        self.__car_model: str = model
        self.__car_year: str = year
        self.__car_speed: int = speed

    def accelerate(self):
        self.__car_speed += 5

    def brake(self):
        if self.__car_speed > 0:
            self.__car_speed -= 5

    def get_speed(self):
        return self.__car_speed


print('\nTASK 3!!!')
task_3_car: Car = Car('Chevrolet', 'Corvette 2020(C8)', '2020')
print('Accelerating car...')
for accelerate in range(0, 5):
    task_3_car.accelerate()
    print(f'Current car speed: {task_3_car.get_speed()}')
print('Braking car...')
for brake in range(0, 5):
    task_3_car.brake()
    print(f'Current car speed: {task_3_car.get_speed()}')

# task 4
"""
4. Напишіть клас Dog, який має три атрибути класу: mammal (ссавець), nature (характер) і
breed (порода), та два атрибути екземпляра: name (кличка) і age (вік).
    Створіть екземпляри трьох нових собак, кожна з яких різного віку. Визначте у класі Dog
метод для виведення значень атрибутів екземпляру - імені та віку конкретної собаки.
    За потреби, додайте кілька інших методів, які визначають поведінку собаки (подавання
голосу тощо). Напишіть кілька класів, які унаслідуються від батьківського класу Dog, що
    описують конкретні породи собак. Визначте для цих класів атрибути nature і breed
відповідно, включіть у класи по одному методу, що визначає поведінку конкретної породи
собаки.
    Створіть батьківський клас Pets, що створює список ваших домашніх улюбленців. У
підсумку, надрукуйте інформацію про ваших домашніх тварин, на зразок, як у вихідних даних.
"""
```

```python
class Dog:
    __mammal: bool
    __nature: str
    __breed: str

    def __init__(self, name: str, age: int):
        self.__name: str = name
        self.__age: int = age

    def __str__(self):
        return f'Dog name: {self.__name}. Dog age: {self.__age}.'

    def voice(self):
        if self.__age < 2:
            return 'Tyaf!'
        elif 2 <= self.__age < 5:
            return 'Wafh!'
        else:
            return 'GAF!'


class Akita(Dog):
    __mammal = True
    __nature = 'The Akita is generally seen as territorial about its property, and can be
reserved with strangers.'
    __breed = 'Akita'

    def __str__(self):
        return f'Mammal: {self.__mammal}.\nNature: {self.__nature}.\nBreed:
{self.__breed}.'

    @staticmethod
    def temperament():
        return 'Akita is here! It\'s my territory. GAF! I\'m cleaning my face after
eating!'


class Doberman(Dog):
    __mammal = True
    __nature = 'Doberman are considered to be working dogs and often stereotyped as being
ferocious and aggressive.'
    __breed = 'Doberman'

    def __str__(self):
        return f'Mammal: {self.__mammal}.\nNature: {self.__nature}.\nBreed:
{self.__breed}.'

    @staticmethod
    def temperament():
        return 'GAFFFF!!!!! GAF! GAF! GAF!I\'m Doberman and it\'s my owner, bi*ches!'


class Pets:
    def __init__(self):
        self._pets_list: list = []

    def add_pet(self, pet):
        self._pets_list.append(pet)

    def remove_pet(self, pet):
        self._pets_list.remove(pet)

    @property
    def pets_list(self):
        return self._pets_list


print('\nTASK 4!!!')
```

```python
task_4_first_dog: Akita = Akita('Aki', 4)
task_4_second_dog: Doberman = Doberman('Loli', 2)
task_4_third_dog: Dog = Dog('Poppy', 7)
task_4_pets: Pets = Pets()
task_4_pets.add_pet(task_4_first_dog)
task_4_pets.add_pet(task_4_second_dog)
task_4_pets.add_pet(task_4_third_dog)
counter: int = 1
for pet in task_4_pets.pets_list:
    print(f'{counter} dog:\n{pet}')
    counter += 1

# task 5
"""
5. Дано послідовність цілих чисел. Необхідно її обробити і вивести на екран суму першої
п’ятірки чисел із цієї послідовності, потім суму другої п’ятірки, і т. д. Але послідовність
    не дається відразу загалом. З плином часу до вас надходять її послідовні частини.
Наприклад, спочатку перші три елементи, потім наступні шість, потім наступні два і т. д.
    Реалізуйте клас Buffer, який буде накопичувати в собі елементи послідовності і виводити
суму п’ятірок послідовних елементів у міру їх накопичення. Однією з вимог до класу є те,
    що він не повинен зберігати в собі більше елементів, ніж йому дійсно необхідно, тобто,
він не повинен зберігати елементи, які вже увійшли в п’ятірку, для якої була виведена
    сума. Зверніть увагу, що під час виконання методу add виводити суму п’ятірок може
знадобитися кілька разів до тих пір, поки в буфері не залишиться менше п’яти елементів.
"""


class Buffer:
    def __init__(self, *data: float):
        self.__sequence: list = [*data]

    def __str__(self):
        return f'Sequence: {self.__sequence}'

    def add_elems_to_sequence(self, *data: float):
        for arg in data:
            self.__sequence.append(arg)

    def get_sum_list_5_elems(self):
        sum_list = []

        last_counter = 0
        for counter in range(5, len(self.__sequence), 5):
            float_sum = 0
            for index in range(last_counter, counter):
                float_sum += self.__sequence[index]
            last_counter = counter
            sum_list.append(float_sum)

        return sum_list

    def get_sequence(self):
        return self.__sequence

    def get_sequence_length(self):
        return len(self.__sequence)


print('\nTASK 5!!!')
task_5_sequence: Buffer = Buffer(1, 2, 3, 4, 5, 1, 1, 2)
print(f'Current Buffer sequence: {task_5_sequence.get_sequence()}')
print(f'Sum list of every 5 elems of sequence: {task_5_sequence.get_sum_list_5_elems()}')
task_5_sequence.add_elems_to_sequence(1, 1, 3, 10, -10, -20)
print(f'Current Buffer sequence: {task_5_sequence.get_sequence()}')
print(f'Sum list of every 5 elems of sequence: {task_5_sequence.get_sum_list_5_elems()}')

# task 6
"""
    6. Напишіть клас-виняток, на основі вбудованого в Python класу ValueError(). Клас буде
```

```
представляти перевірку певного імені на основі його довжини. Якщо довжина введеного
    імені є меншою 10, то має генеруватися виняток як у вихідних даних. У інших випадках
нічого не виводиться.
"""


class FullNameError(Exception):
    def __init__(self, value: str):
        self.value: str = value


class Name:
    def __init__(self, first_name: str, last_name: str, middle_name: str):
        self.__first_name: str = first_name
        self.__last_name: str = last_name
        self.__middle_name: str = middle_name
        self.__full_name = last_name + ' ' + first_name + ' ' + middle_name

    def __str__(self):
        return f'First name -> {self.__first_name} Last name -> {self.__last_name} Middle
name -> {self.__middle_name}'

    def check_entered_name(self):
        try:
            if len(self.__full_name) < 10:
                raise FullNameError('Name length is less than 10!')
            else:
                pass
                print('Name was entered correctly!')
        except FullNameError as name_error:
            self.__first_name: str = ''
            self.__last_name: str = ''
            self.__middle_name: str = ''
            print(name_error)


print('\nTASK 6!!!')
task_6_name_1: Name = Name('Andrii', 'Babushko', 'Sergiyovich')
print(f'Data of task_6_name_1 instance of a Name class:\n{task_6_name_1}')
task_6_name_1.check_entered_name()
task_6_name_2: Name = Name('A', 'B', 'S')
print(f'\nData of task_6_name_2 instance of a Name class:\n{task_6_name_2}')
task_6_name_2.check_entered_name()

# task 7
"""
    7. Напишіть один клас для перетворення десяткового числа на число в римській системі
числення. І ще один клас для перетворення числа з римської системи числення у десяткове
    число.
"""


class DecimalToRoman:
    __roman_number: str

    def __init__(self, decimal: int):
        self.__decimal_number: int = decimal

    def __str__(self):
        return f'Decimal number: {self.__decimal_number}; Roman: {self.__roman_number};'

    def move_decimal_to_roman(self):
        values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
        symbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"]

        i = 0
        number = self.__decimal_number
        roman_number: str = ''
```

```
        while number > 0:
            for interator in range(number // values[i]):
                roman_number += symbols[i]
                number -= values[i]
            i += 1

        self.__roman_number = roman_number
        return roman_number

    def set_decimal(self, decimal: int):
        self.__decimal_number: int = decimal

    def get_roman_number(self):
        return self.__roman_number

    def get_decimal_number(self):
        return self.__decimal_number


class RomanToDecimal:
    __decimal_number: int

    def __init__(self, roman: str):
        self.__roman_number: str = roman

    def __str__(self):
        return f'Roman: {self.__roman_number}; Decimal number: {self.__decimal_number};'

    def move_roman_to_decimal(self):
        roman_values = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
        decimal_value = 0

        for i in range(len(self.__roman_number)):
            if i > 0 and roman_values[self.__roman_number[i]] >
roman_values[self.__roman_number[i - 1]]:
                decimal_value += roman_values[self.__roman_number[i]] - 2 *
roman_values[self.__roman_number[i - 1]]
            else:
                decimal_value += roman_values[self.__roman_number[i]]

        return decimal_value

    def set_roman(self, roman: str):
        self.__roman_number: str = roman

    def get_roman_number(self):
        return self.__roman_number

    def get_decimal_number(self):
        return self.__decimal_number


print('\nTASK 7!!!')
task_7_decimal_to_roman: DecimalToRoman = DecimalToRoman(15)
print(f'Move {task_7_decimal_to_roman.get_decimal_number()} decimal number to roman number:
{task_7_decimal_to_roman.move_decimal_to_roman()}')
task_7_roman_to_decimal: RomanToDecimal =
RomanToDecimal(task_7_decimal_to_roman.get_roman_number())
print(f'Move {task_7_roman_to_decimal.get_roman_number()} roman number to decimal number:
{task_7_roman_to_decimal.move_roman_to_decimal()}')

# task 8
"""
    8. Онлайн-магазин
a. Створіть клас з ім’ям Shop(). Клас Shop() повинен містити два атрибути: shop_name і
store_type. Створіть метод describe_shop(), який виводить два атрибути, і метод
open_shop(),
який виводить повідомлення про те, що онлайн-магазин відкритий. Створіть на основі класу
екземпляр з ім’ям store. Виведіть два атрибути окремо, потім викличте обидва методи.
```

```
b. Створіть три різних екземпляри класу, викличте для кожного екземпляру метод
describe_shop().
c. Додайте атрибут number_of_units зі значенням за замовчуванням 0; він представляє
кількість видів товару у магазині. Створіть екземпляр з ім'ям store. Виведіть значення
number_of_units, а потім змініть number_of_units і виведіть знову.
d. Додайте метод з ім'ям set_number_of_units(), що дозволяє задати кількість видів товару.
Викличте метод з новим числом, знову виведіть значення. Додайте метод з ім'ям
increment_number_of_units(), який збільшує кількість видів товару на задану величину.
Викличте цей метод.
e. Напишіть клас Discount(), що успадковує від класу Shop(). Додайте атрибут з ім'ям
discount_products для зберігання списку товарів, на які встановлена знижка. Напишіть метод
get_discounts_ptoducts, який виводить цей список. Створіть екземпляр store_discount і
викличте цей метод.
f. Збережіть код класу Shop() у модулі. Створіть окремий файл, що імпортує клас Shop().
Створіть екземпляр all_store і викличте один з методів Shop(), щоб перевірити, що команда
import працює правильно.
"""

from modules import shop

print('\nTASK 8!!!')
print('a)')
task_8_store: shop.Shop = shop.Shop('Store', 'stuff store')
print(f'Print separately shop name: {task_8_store.name}')
print(f'Print separately shop type: {task_8_store.type}')
print('Shop status:')
task_8_store.open_shop()
print('Describe shop:')
task_8_store.describe_shop()

print('\nb)')
task_8_raccoons_tech_shop: shop.Shop = shop.Shop('Tech Raccoons', 'tech shop')
print('Describe shop:')
task_8_raccoons_tech_shop.describe_shop()
task_8_raccoons_sex_shop: shop.Shop = shop.Shop('Sexy Raccoons', 'sex shop')
print('Describe shop:')
task_8_raccoons_sex_shop.describe_shop()
task_8_raccoons_food_shop: shop.Shop = shop.Shop('Raccoon\'s Food Market', 'food market')
print('Describe shop:')
task_8_raccoons_food_shop.describe_shop()

print('\nc)')
print(f'Current store\'s number of units: {task_8_store.number_of_units}')
task_8_store.number_of_units = 12
print(f'Changed store\'s number of units: {task_8_store.number_of_units}')

print('\nd)')
task_8_store.set_number_of_units(10)
print(f'Set store\'s number of units by 10: {task_8_store.number_of_units}')
task_8_store.increment_number_of_units(10)
print(f'Incremented store\'s number of units by 10: {task_8_store.number_of_units}')


class Discount(shop.Shop):
    def __init__(self, shop_name: str, shop_type: str, **kwargs):
        super().__init__(shop_name, shop_type)
        self.discount_products: dict = kwargs

    def get_discounts_products(self):
        discount_keys: list = list(self.discount_products.keys())
        discount_values: list = list(self.discount_products.values())

        print(f'Current discounts:')
        for iterator in range(0, len(self.discount_products)):
            print(f'{discount_keys[iterator]}: {discount_values[iterator]}')


print('\ne)')
discounts: dict = {'Car Toy': '10%', 'Doll toy': '20%', 'Mobile phone': '15%'}
```

```
task_8_discount_store: Discount = Discount('Store', 'stuff store', Car_Toy='10%',
Doll_toy='20%', Mobile_phone='15%')
task_8_discount_store.get_discounts_products()

print('\nf)')
task_8_all_store: shop.Shop = shop.Shop('All  store', 'store')
print(f'Describe all store:')
task_8_all_store.describe_shop()
task_8_all_store.set_number_of_units(20)
print(f'Set all store\'s number of units by 20: {task_8_all_store.number_of_units}')
task_8_all_store.increment_number_of_units(30)
print(f'Incremented store\'s number of units by 30: {task_8_all_store.number_of_units}')

# task 9
"""
    9. Облік користувачів на сайті
a. Створіть клас з ім’ям User. Створіть два атрибути first_name і last_name, а потім ще
кілька атрибутів, які зазвичай зберігаються у профілі користувача (поштова адреса, нікнейм,
що відображається на сайті, згода на розсилку новин з форуму). Напишіть метод describe_user
який виводить повне ім’я користувача. Створіть ще один метод greeting_user() для
виведення персонального вітання для користувача. Створіть кілька примірників, які
представляють різних користувачів. Викличте обидва методи для кожного користувача.
b. Додайте атрибут login_attempts у клас User. Напишіть метод increment_login_attempts(),
що збільшує значення login_attempts на 1. Напишіть інший метод з ім’ям
reset_login_attempts(), обнуляє значення login_attempts. Створіть екземпляр класу User і
викличте increment_login_attempts() кілька разів. Виведіть значення login_attempts, щоб
переконатися у тому, що значення було змінено правильно, а потім викличте
reset_login_attempts(). Знову виведіть login_attempts і переконайтеся у тому, що значення
обнулилося
c. Адміністратор - користувач з повними адміністративними привілеями. Напишіть клас з ім’ям
Admin, що успадковує від класу User. Додайте атрибут privileges для зберігання списку
рядків виду «Allowed to add message», «Allowed to delete users», «Allowed to ban users» і
т. д. Напишіть метод show_privileges() для виведення набору привілеїв адміністратора.
Створіть екземпляр Admin і викличте метод.
d. Напишіть клас Privileges. Клас повинен містити всього один атрибут privileges зі
списком, який треба забрати із класу Admin. Водночас, необхідно перемістити метод
show_privileges() у клас Privileges із класу Admin. Створіть екземпляр priv як атрибут
класу Admin. Створіть новий екземпляр admin і використайте метод для виведення
списку привілеїв.
e. Збережіть клас User в одному модулі, а класи Privileges і Admin у іншому модулі. В
окремому файлі створіть екземпляр admin і викличте метод show_privileges(), щоб перевірити,
що все працює правильно.
"""

from modules import user
from modules import admin

print('\nTASK 8!!!')
print('a)')
task_9_1_user: user.User = user.User('Andrii', 'Babushko', 'andriibabushko@gmail.com',
'AndriiRaccoon', True)
task_9_1_user.greeting_user()
task_9_1_user.describe_user()

print('\nb)')
incrementing: int = 5
print(f'Incrementing login attempts 1_user {incrementing} times...')
for iterator in range(0, incrementing):
    task_9_1_user.increment_login_attempts()
print(f'Current login attempts: {task_9_1_user.login_attempts}')
task_9_1_user.reset_login_attempts()
print(f'Login attempts after resetting: {task_9_1_user.login_attempts}')
incrementing: int = 10
print(f'Repeating incrementing login attempts 1_user {incrementing} times...')
for iterator in range(0, incrementing):
    task_9_1_user.increment_login_attempts()
print(f'Current login attempts: {task_9_1_user.login_attempts}')
task_9_1_user.reset_login_attempts()
print(f'Login attempts after resetting: {task_9_1_user.login_attempts}')
```

```
print('\nc)')
task_9_1_admin: admin.Admin = admin.Admin('Andrii', 'Babushko', 'andriibabushko@gmail.com',
'AndriiRaccoon', True,
                                          ['Allowed to block users', 'Allowed to delete
messages', 'Allowed to send voices', 'Allowed to delete users'])
task_9_1_admin.show_privileges()

print('\nd)')
print('Show privileges by using class intense inside Admin class:')
task_9_1_admin.admin_privileges.show_privileges()

print('\ne)')
task_9_2_admin: admin.Admin = admin.Admin('Ihor', 'Juice', 'ihorjuice@gmail.com',
'KindOfHell', False,
                                          ['Allowed to block users', 'Allowed to delete
messages', 'Allowed to delete voices', 'Allowed to delete users', 'Allowed to delete app'])
task_9_2_admin.admin_privileges.show_privileges()
```

Висновок: під час виконання лабораторної роботи було отримано навички написання власних класів на мові Python, створення та підключення власних модулів до файлів .py.