**ЛАБОРАТОРНА РОБОТА № 8**

**КЛАСИ. Ч. 2**

***Мета***: ознайомитися з ООП в мові Python

**Хід роботи**

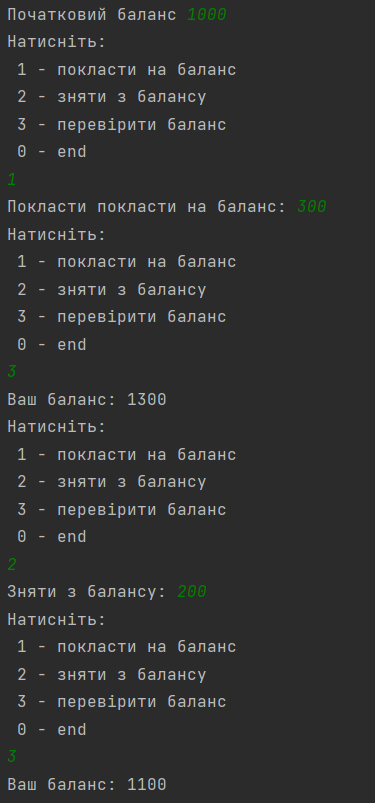
Напишіть програми у середовищі програмування для розв’язування таких завдань:

**Завдання 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):  
 self.balance = balance  
  
 def \_\_sub\_\_(self, other):  
 self.balance += other  
 return self.balance  
  
 def \_\_neg\_\_(self, other):  
 if self.balance - other >= 0:  
 self.balance -= other  
 else:  
 print("Недостатньо коштів")  
  
 def check(self):  
 print(f"Ваш баланс: {self.balance}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 x = int(input("Початковий баланс "))  
 user = Bank(x)  
 while 1:  
 print(  
 "Натисніть:\n 1 - покласти на баланс\n 2 - зняти з балансу \n 3 - перевірити баланс\n 0 - end")  
 n = int(input())  
 if n == 1:  
 sub = int(input("Покласти покласти на баланс: "))  
 user.\_\_sub\_\_(sub)  
 elif n == 2:  
 neg = int(input("Зняти з балансу: "))  
 user.\_\_neg\_\_(neg)  
 elif n == 3:  
 user.check()  
 elif n == 0:  
 break  
 else:  
 print("Спробуйте ще раз!!!")

Результат виконання програми:

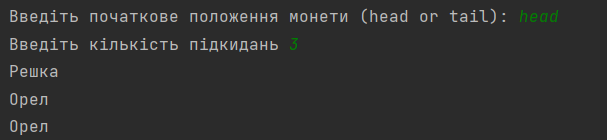


**Завдання 2:** Напишіть клас Coin, який описує монету, яку можна підкидати. При створенні екземпляру класу, екземпляр отримує атрибут \_\_sideup зі значенням heads або tails. У класі визначте метод toss, який випадково визначає результат підкидання монети - орел чи решка. Створіть екземпляр класу і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

Лістинг програми

import random  
  
  
class Coin:  
 def \_\_init\_\_(self, \_\_sideup):  
 self.sideup = \_\_sideup  
  
 def toss(self):  
 tmp = random.randint(1, 2)  
 if self.sideup == "head" and tmp == 1:  
 return "Орел"  
 elif self.sideup == "head" and tmp == 2:  
 return "Решка"  
 elif self.sideup == "tail" and tmp == 1:  
 return "Решка"  
 elif self.sideup == "tail" and tmp == 2:  
 return "Орел"  
 else:  
 print("Помилка стартового положення монети")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 x = str(input("Введіть початкове положення монети (head or tail): "))  
 coin = Coin(x)  
 tmp = int(input("Введіть кількість підкидань "))  
 for i in range(tmp):  
 print(coin.toss())

Результат виконання програми:

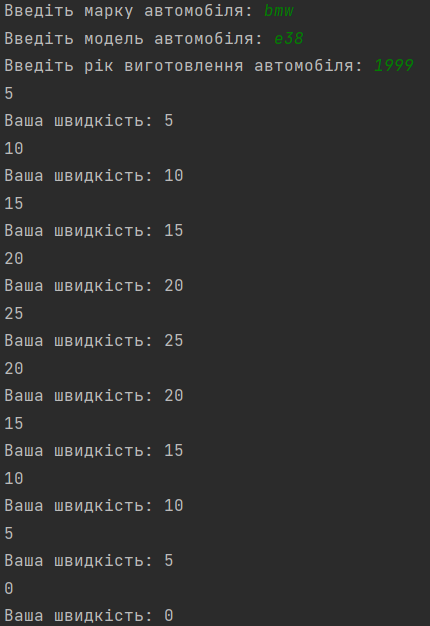


**Завдання 3:** Напишіть клас Car, який надає для створених екземплярів такі атрибути даних автомобіля: марку виготовлення автомобіля, модель автомобіля, рік автомобіля, швидкість (початкове значення 0). Клас також повинен мати наступні методи: accelerate (метод повинен щоразу додавати 5 до значення 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, model, year, speed):  
 self.mark = mark  
 self.model = model  
 self.year = year  
 self.speed = speed  
  
 def accelerate(self):  
 self.speed += 5  
 return self.speed  
  
 def brake(self):  
 self.speed -= 5  
 return self.speed  
  
 def get\_speed(self):  
 return self.speed  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 mark = str(input("Введіть марку автомобіля "))  
 model = str(input("Введіть модель автомобіля "))  
 year = int(input("Введіть рік виготовлення автомобіля "))  
 car = Car(mark, model, year, 0)  
 for i in range(5):  
 print(car.accelerate())  
 print(f"Ваша швидкість {car.get\_speed()}")  
 for i in range(5):  
 print(car.brake())  
 print(f"Ваша швидкість {car.get\_speed()}")

Результат виконання програми:

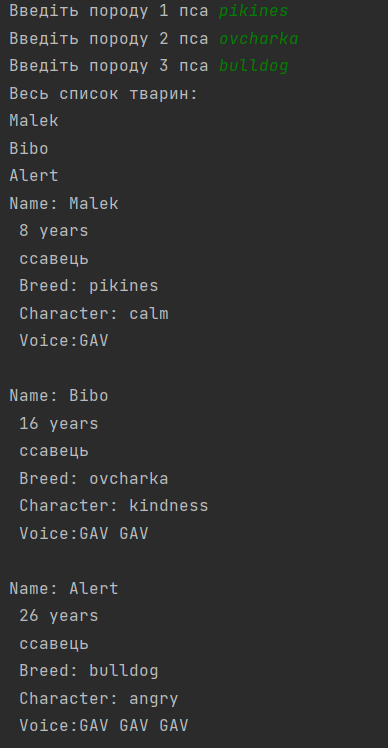


**Завдання 4:** Напишіть клас Dog, який має три атрибути класу: mammal (ссавець), nature (характер) і breed (порода), та два атрибути ексземпляра: name (кличка) і age (вік). Створіть екземпляри трьох нових собак, кожна з яких різного віку. Визначте у класі Dog метод для виведення значень атрибутів екземпляру - 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

class Dog:  
 count = 0  
  
 def \_\_init\_\_(self, name, age):  
 Dog.count += 1  
 self.name = name  
 self.age = age  
 self.mammal = "ссавець"  
 self.breed = input(f"Введіть породу {Dog.count} пса ")  
 self.nature = Behavior.natures(self)  
  
 def print\_name\_age(self):  
 print(f"{self.name} має вік {self.age}")  
  
 def voice(self):  
 if (self.age >= 0) and (self.age < 10):  
 return "GAV"  
 elif (self.age >= 10) and (self.age < 20):  
 return "GAV GAV"  
 elif (self.age >= 20) and (self.age < 30):  
 return "GAV GAV GAV"  
 else:  
 return "AUU"  
  
  
class Behavior(Dog):  
 def \_\_init\_\_(self, nature, breed):  
 Dog.\_\_init\_\_(self, nature, breed)  
  
 def natures(self):  
 if self.breed == "pikines":  
 self.nature = "calm"  
 return self.nature  
 elif self.breed == "ovcharka":  
 self.nature = "kindness"  
 return self.nature  
 elif self.breed == "bulldog":  
 self.nature = "angry"  
 return self.nature  
 else:  
 self.nature = ""  
 return self.nature  
  
  
class Pets:  
 def \_\_init\_\_(self, list\_):  
 self.list = list\_  
  
 def print\_all(self):  
 for i in self.list:  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 dog1 = Dog("Malek", 8)  
 dog2 = Dog("Bibo", 16)  
 dog3 = Dog("Alert", 26)  
 list1 = [dog1.name, dog2.name, dog3.name]  
 list2 = Pets(list1)  
 print("Весь список тварин: ")  
 list2.print\_all()  
 print(  
 f"Name: {dog1.name}\n {dog1.age} years\n {dog1.mammal}\n Breed: {dog1.breed}\n Character: {dog1.nature}\n Voice:{dog1.voice()}\n")  
 print(  
 f"Name: {dog2.name}\n {dog2.age} years\n {dog2.mammal}\n Breed: {dog2.breed}\n Character: {dog2.nature}\n Voice:{dog2.voice()}\n")  
 print(  
 f"Name: {dog3.name}\n {dog3.age} years\n {dog3.mammal}\n Breed: {dog3.breed}\n Character: {dog3.nature}\n Voice:{dog3.voice()}\n")

Результат виконання програми:

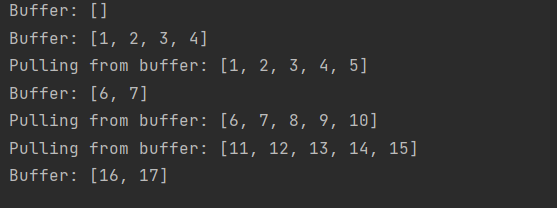


**Завдання 5:** Дано послідовність цілих чисел. Необхідно її обробити і вивести на екран суму першої п’ятірки чисел із цієї послідовності, потім суму другої п’ятірки, і т. д. Але послідовність не дається відразу загалом. З плином часу до вас надходять її послідовні частини. Наприклад, спочатку перші 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. 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

class Buffer:  
 def \_\_init\_\_(self):  
 self.\_\_buffer = []  
  
 def get\_current\_part(self):  
 return [self.\_\_buffer.pop(0) for \_ in self.\_\_buffer[:5]] if len(self.\_\_buffer) >= 5 else None  
  
 def add(self, \*amount):  
 self.\_\_buffer += amount  
 while part := self.get\_current\_part():  
 print(f'Pulling from buffer: {part}')  
  
 @property  
 def buffer(self):  
 return self.\_\_buffer  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 b = Buffer()  
 print(f'Buffer: {b.buffer}')  
 b.add(1, 2, 3, 4)  
 print(f'Buffer: {b.buffer}')  
 b.add(5, 6, 7)  
 print(f'Buffer: {b.buffer}')  
 b.add(8, 9, 10, 11, 12, 13, 14, 15, 16, 17)  
 print(f'Buffer: {b.buffer}')

Результат виконання програми:

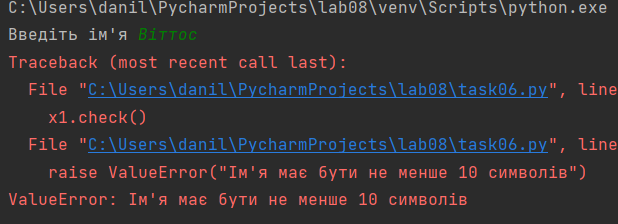


**Завдання 6:** Напишіть клас-виняток, на основі вбудованого в Python класу ValueError(). Клас буде представляти перевірку певного імені на основі його довжини. Якщо довжина введеного імені є меншою 10, то має генеруватися виняток як у вихідних даних. У інших випадках нічого не виводиться

Лістинг програми

class Error:  
 def \_\_init\_\_(self, len):  
 self.len = len  
  
 def check(self):  
 if self.len < 10:  
 raise ValueError("Ім'я має бути не менше 10 символів")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 x = input("Введіть ім'я ")  
 x1 = Error(len(x))  
 x1.check()

Результат виконання програми:



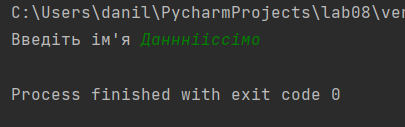


Рис. 6. Результат виконання програми

**Завдання 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

import roman  
  
  
class Transform1:  
 def \_\_init\_\_(self, num):  
 self.num = num  
  
 def function1(self):  
 result = roman.toRoman(self.num)  
 return result  
  
  
class Transform2:  
 def \_\_init\_\_(self, num):  
 self.num = num  
  
 def function2(self):  
 result = roman.fromRoman(self.num)  
 return result  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 num1 = int(input("Введіть десятокве число: "))  
 x1 = Transform1(num1)  
 print(x1.function1())  
 num2 = input("Введіть римське число: ")  
 x2 = Transform2(num2)  
 print(x2.function2())

Результат виконання програми:

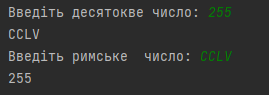


Рис. 7. Результат виконання програми

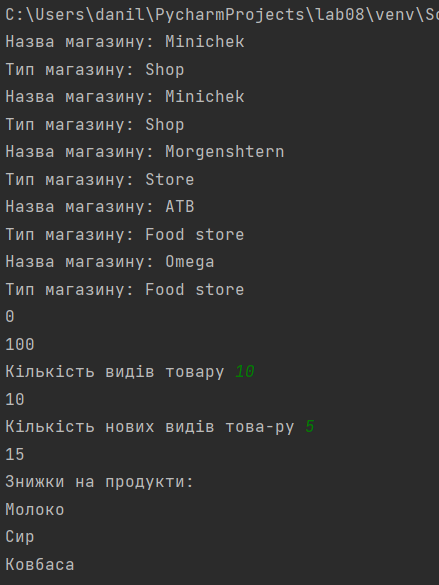
**Завдання 8:** Онлайн-магазин:

1. 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.
2. 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.
3. 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.
4. 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.
5. 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(task\_8.py)

class Shop:  
 def \_\_init\_\_(self, shop\_name, shop\_type, numbers\_of\_units=0):  
 self.shop\_name = shop\_name  
 self.shop\_type = shop\_type  
 self.number\_of\_units = numbers\_of\_units  
  
 def set\_number\_of\_units(self):  
 numbers\_of\_units = int(input("Кількість видів товару "))  
 self.number\_of\_units = numbers\_of\_units  
 return self.number\_of\_units  
  
 def increment\_number\_of\_units(self):  
 increment\_number\_of\_units = int(input("Кількість нових видів това-ру "))  
 self.number\_of\_units += increment\_number\_of\_units  
 return self.number\_of\_units  
  
 def describe\_shop(self):  
 print(f"Назва магазину: {self.shop\_name}\nТип магазину: {self.shop\_type}")  
  
 @staticmethod  
 def open\_shop():  
 print("Магазин відкрито")  
  
  
class Discount(Shop):  
 def \_\_init\_\_(self, discount\_products, shop\_name, shop\_type):  
 super().\_\_init\_\_(shop\_name, shop\_type)  
 self.discount\_products = discount\_products  
  
 def get\_discounts\_products(self):  
 print("Знижки на продукти:")  
 for i in self.discount\_products:  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 store = Shop("Minichek", "Shop")  
 print(f"Назва магазину: {store.shop\_name}\nТип магазину: {store.shop\_type}")  
 store.describe\_shop()  
 store1 = Shop("Morgenshtern", "Store")  
 store2 = Shop("ATB", "Food store")  
 store3 = Shop("Omega", "Food store")  
 store1.describe\_shop()  
 store2.describe\_shop()  
 store3.describe\_shop()  
 store\_C = Shop("Nikotino", "Tabako store")  
 print(store\_C.number\_of\_units)  
 store\_C.number\_of\_units = 100  
 print(store\_C.number\_of\_units)  
 print(store\_C.set\_number\_of\_units())  
 print(store\_C.increment\_number\_of\_units())  
 list1 = ["Молоко", "Сир", "Ковбаса"]  
 store\_discount = Discount(list1, "ATB", "Food store")  
 store\_discount.get\_discounts\_products()

Результат виконання програми:



Лістинг програми(task\_8.1.py)

from task08 import Shop  
  
all\_store = Shop("Bittos", "store")  
all\_store.describe\_shop()

Результат виконання програми:

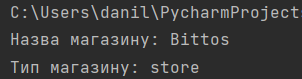


Рис. 8. Результат виконання програми

**Завдання 9:** Облік користувачів на сайті

1. 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.
2. 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.
3. 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.
4. 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.
5. 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(task\_9.py)

from task09\_adm import Admin  
from task09\_user import User  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 user1 = User("Богданенко", "Карпо", "Клосовського 14", "chill")  
 user2 = User("Савченко", "Данило", "Вільський 14", "top")  
 user3 = User("Вітренко", "Петро", "Корольова 12", "lol")  
 user1.describe\_user()  
 user1.greeting\_user()  
 user2.describe\_user()  
 user2.greeting\_user()  
 user3.describe\_user()  
 user3.greeting\_user()  
 for i in range(5):  
 user1.increment\_login\_attempts()  
 print(user1.login\_attempts)  
 user1.reset\_login\_attempts()  
 print(user1.login\_attempts)  
 list1 = ["Allowed to add message", "Allowed to delete users", "Allowed to ban users"]  
 user4 = Admin(list1, "Савченко", "Данило", "Вільський шлях 14", "dan1ssimo\_")  
 user4.priv.show\_privileges()

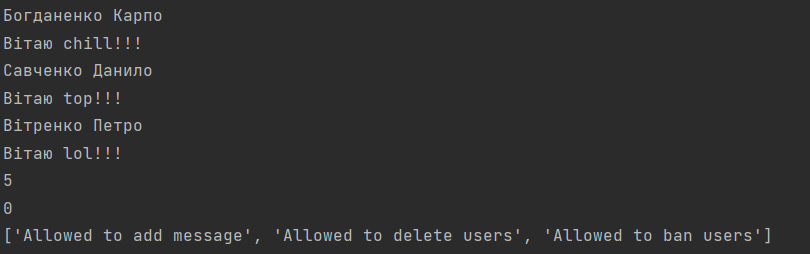
Лістинг програми(task\_9\_priv\_admin.py)

from task09\_user import User  
  
  
class Admin(User):  
 def \_\_init\_\_(self, privileges, first\_name, last\_name, adress, nickname):  
 super().\_\_init\_\_(first\_name, last\_name, adress, nickname)  
 self.privileges = privileges  
 self.priv = Privileges()  
  
  
class Privileges:  
 def \_\_init\_\_(self):  
 self.privileges = ['Allowed to add message', 'Allowed to delete users', 'Allowed to ban users']  
  
 def show\_privileges(self):  
 print(self.privileges)

Лістинг програми(task\_9\_user.py)

class User:  
 def \_\_init\_\_(self, first\_name, last\_name, adress, nickname, login\_attempts=0):  
 self.first\_name = first\_name  
 self.last\_name = last\_name  
 self.adress = adress  
 self.nickname = nickname  
 self.login\_attempts = login\_attempts  
  
 def describe\_user(self):  
 print(self.first\_name, self.last\_name)  
  
 def greeting\_user(self):  
 print(f"Вітаю {self.nickname}!!!")  
  
 def increment\_login\_attempts(self):  
 self.login\_attempts += 1  
 return self.login\_attempts  
  
 def reset\_login\_attempts(self):  
 self.login\_attempts = 0  
 return self.login\_attempts

Результат виконання програми:



***Висновки:*** ознайомилися з ООП в мові Python