Міністерство освіти і науки України

Державний університет ,,Житомирська політехніка”

Кафедра КІ та КБ

Група: КБ-21-1

Програмування мовою Python

Лабораторна робота №9

«КЛАСИ. Ч. 3»

Виконала: Поліщук К. Р.

Прийняв: Морозов Д. С.

**Мета роботи:** : ознайомитися з ООП, множинним наслідуванням, міксинами в мові Python

**Хід роботи**

Завдання 1: Створіть клас Alphabet. Його метод \_\_init \_\_ (), буде мати визначені два параметри: lang - мова і letters - список букв. Значення змінних lang і letters будуть визначенні за замовчуванням і міститимуться у вигляді статичних атрибутів для української мови.

Клас матиме метод метод print\_alphabet(), який виведе в консоль літери українського алфавіту. Метод letters\_num(), повертатиме кількість букв в алфавіті. Метод is\_ua\_lang() прийматиме довільний текст і визначатиме чи відноситься він до української мови (незалежно від регістру).

Створіть клас EngAlphabet шляхом успадкування від класу Alphabet. Для його методу \_\_init \_\_(), всередині якого буде викликатися батьківський метод \_\_init \_\_(), в якості параметрів будуть передаватися позначення мови (наприклад, 'En') і рядок, що складається з усіх букв алфавіту. Додайте приватний статичний атрибут \_\_en\_letters\_num, який буде зберігати кількість букв в алфавіті. Створіть метод is\_en\_letter(), який буде приймати строку в якості параметра і визначати, чи відноситься ця строка до англійського алфавіту. Перевизначити метод letters\_num() - нехай в поточному класі класі він буде повертати значення властивості \_\_en\_letters\_num. 6. Створіть статичний метод example(), який буде повертати приклад тексту англійською мовою.

Тести до модуля:

- Створіть об'єкт класу EngAlphabet

- Надрукуйте літери алфавіту для цього об'єкту

- Виведіть кількість букв в алфавіті

- Перевірте, чи відноситься буква J до англійського алфавіту.

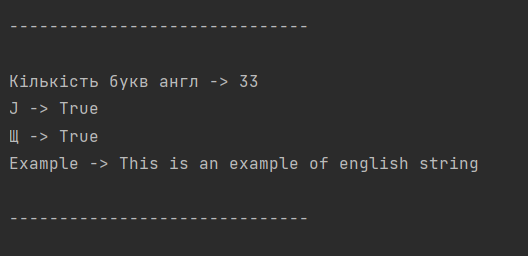
- Перевірте, чи відноситься буква Щ до українського алфавіту

- Виведіть приклад тексту англійською мовою

Лістинг програми:

def task1():  
 class Alphabet:  
 lang = 'українська'  
 letters = (  
 'а', 'б', 'в', 'г', 'ґ', 'д', 'е', 'є', 'ж', 'з', 'и', 'і', 'ї',  
 'й', 'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с', 'т',  
 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ь', 'ю', 'я'  
 )  
  
 def \_\_init\_\_(self, lang=lang, letters=letters):  
 self.lang = lang  
 self.letters = letters  
  
 def print\_alphabet(self):  
 print(f'{self.lang} => {self.letters}')  
  
 def letters\_num(self):  
 return len(self.letters)  
  
 def is\_ua\_lang(self, text):  
 for letter in text.lower():  
 if letter not in self.\_\_class\_\_.letters:  
 return False  
  
 return True  
  
 class EngAlphabet(Alphabet):  
 lang = 'українська'  
 letters = (  
 'а', 'б', 'в', 'г', 'ґ', 'д', 'е', 'є', 'ж',  
 'з', 'и', 'і', 'ї', 'й', 'к', 'л', 'м', 'н', 'о',  
 'п', 'р', 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ь', 'ю', 'я'  
 )  
  
 \_\_en\_letters\_num = len(letters)  
  
 def \_\_innit\_\_(self, lang=lang, letters=letters):  
 super().\_\_init\_\_(lang, letters if type(letters) is list else [\*letters])  
 self.\_\_en\_letters\_num = len(self.letters)  
  
 def is\_en\_lang(self, text):  
 return bool(sum(map(text.lower().count, self.letters)))  
  
 def letters\_num(self):  
 return self.\_\_en\_letters\_num  
  
 @staticmethod  
 def example():  
 return 'This is an example of english string'  
  
 lettersEng = 'abcdefghijklmnopqrstuvwxyz'  
 testEng = EngAlphabet('english', lettersEng)  
 print(f'Кількість букв англ -> {testEng.letters\_num()}')  
 print(f'J -> {testEng.is\_en\_lang("J")}')  
 print(f'Щ -> {testEng.is\_ua\_lang("Щ")}')  
 print(f'Example -> {EngAlphabet.example()}')

Результат виконання програми:



Завдання 2: Створіть клас Human. Визначте для нього два статичних атрибути: default\_name і default\_age. Його метод \_\_init \_\_(), який крім self приймає ще два публічних параметри(name і age) і два приватних (money і house). Параметр money визначатиме кількість грошей, а house – посилання на об’єкт класу House. Метод info(), має виводити поля name, age, house і money. Реалізуйте довідковий статичний метод default\_info(), який буде виводити статичні поля default\_name і default\_age. Реалізуйте приватний метод make\_deal(), який буде відповідати за технічну реалізацію покупки будинку: зменшувати кількість грошей на рахунку і привласнювати посилання на тільки що куплений будинок. В якості аргументів даний метод приймає об'єкт будинку та його ціну. Реалізуйте метод earn\_money(), що збільшує значення поля money. Реалізуйте метод buy\_house(), який буде перевіряти, що у людини достатньо грошей для покупки, і здійснювати операцію. Якщо грошей занадто мало – потрібно вивести попередження в консоль. Параметри методу: посилання на будинок і розмір знижки (за замовчуванням 10%).

Створіть клас House. Його метод \_\_init \_\_() містить два динамічних параметри: \_area і \_price, що мають значення за замовчуваннями.

Створіть метод final\_price(), який приймає як параметр розмір знижки і повертає ціну з урахуванням даної знижки. Створіть клас SmallHouse, успадкувавши його функціонал від класу House. Всередині класу SmallHouse перевизначите метод \_\_init \_\_() так, щоб він створював об'єкт з площею 40м2

Тести до модуля:

- Викличте довідковий метод default\_info() для класу Human

- Створіть об'єкт класу Human

- Виведіть довідкову інформацію про створений об'єкт (викличте метод info ()).

- Створіть об'єкт класу SmallHouse

- Спробуйте купити створений будинок, переконайтеся в отриманні

попередження.

- Виправте фінансове становище об'єкта - викличте метод earn\_money()

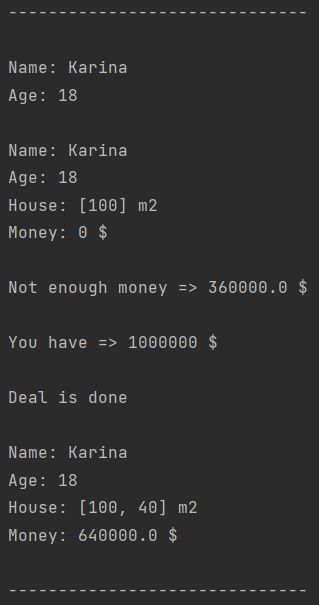
- Знову спробуйте купити будинок

- Подивіться, як змінився стан об'єкта класу Human.

Лістинг програми:

def task2():  
 class House:  
 def \_\_init\_\_(self, area=100, price=1000000):  
 self.\_area = area  
 self.\_price = price  
  
 def final\_price(self, discount=10):  
 return self.\_price \* (1 - (discount / 100))  
  
 class SmallHouse(House):  
 def \_\_init\_\_(self, area=40, price=400000):  
 super().\_\_init\_\_(area, price)  
  
 class Human:  
 default\_name = "Karina"  
 default\_age = 18  
  
 def \_\_init\_\_(self, name=default\_name, age=default\_age, money=0, house=None):  
 if house is None:  
 house = [House()]  
  
 self.name = name  
 self.age = age  
 self.\_\_money = money  
 self.\_\_house = house  
  
 def info(self):  
 print(f'Name: {self.name}')  
  
 print(f'Age: {self.age}')  
 print(f'House: {[i.\_area for i in self.\_\_house]} m2')  
 print(f'Money: {self.\_\_money} $')  
  
 @staticmethod  
 def default\_info():  
 print(f'Name: {Human.default\_name}')  
  
 print(f'Age: {Human.default\_age}')  
  
 def \_\_make\_deal(self, house, price):  
 self.\_\_money -= price  
  
 self.\_\_house.append(house)  
  
 def earn\_money(self, money):  
 self.\_\_money += money  
  
 print(f'You have => {self.\_\_money} $')  
  
 def buy\_house(self, house, discount):  
 if self.\_\_money >= house.final\_price(discount):  
 self.\_\_make\_deal(house, house.final\_price(discount))  
  
 print('Deal is done')  
  
 else:  
 print(f'Not enough money => {house.final\_price(discount) - self.\_\_money} $')  
  
 Human.default\_info()  
 human = Human()  
 print()  
 human.info()  
 small\_house = SmallHouse()  
 print()  
 human.buy\_house(small\_house, 10)  
 print()  
 human.earn\_money(1000000)  
 print()  
 human.buy\_house(small\_house, 10)  
 print()  
 human.info()

Результат виконання програми:



Завдання 3: Створіть клас Apple. Його статичний атрибут states, яке буде містити всі стадії дозрівання яблука («Відсутнє», «Цвітіння», «Зелене», «Червоне»). Метод \_\_init \_\_(), всередині якого будуть визначені два динамічних protected атрибути: \_index (номер яблука) і \_state (приймає перше значення зі словника states). Створіть метод grow(), який буде переводити яблуко на наступну стадію дозрівання Створіть метод is\_ripe(), який буде перевіряти, що яблуко дозріло (досягло останньої стадії дозрівання).

Створіть клас AppleTree. Визначте метод \_\_init \_\_(), який буде приймати як параметр кількість яблук і на його основі буде створювати список об'єктів класу Apple. Даний список буде зберігатися всередині динамічного атрибуту apples.

Створіть метод grow\_all(), який буде переводити всі об'єкти зі списку яблук на наступний етап дозрівання.

Створіть метод all\_are\_ripe(), який буде повертати True, якщо все яблука зі списку стали стиглими. Створіть метод give\_away\_all(), який буде чистити список яблук після збору врожаю

Створіть клас Gardener. Його метод \_\_init \_\_(), міститиме два динамічних атрибути: name (ім’я садівника, публічний атрибут) і \_tree (приймає об’єкт класу AppleTree). Створіть метод work(), який змушує садівника працювати, що дозволяє яблукам ставати більш стиглими. Створіть метод harvest(), який перевіряє, чи всі плоди дозріли. Якщо всі - садівник збирає урожай. Якщо і - метод друкує попередження. Створіть статичний метод apple\_base(), який виведе в консоль довідку з кількості яблук і ступені їх стиглості.

Тести до модуля:

- Створіть декілька об’єктів класу Apple.

- Викличте довідку по всім наявним яблукам

- Створіть об'єкти класів AppleTree і Gardener

- Використовуючи об'єкт класу Gardener, попрацювати над яблучним деревом.

- Спробуйте зібрати урожай

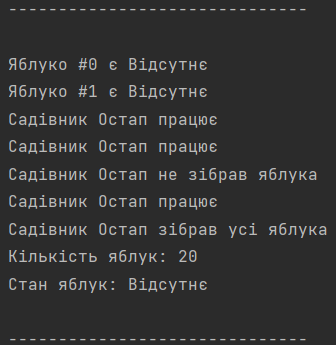
- Якщо яблука ще не дозріли, продовжуйте доглядати за деревом

- Зберіть урожай.

Лістинг програми:

def task3():  
 class Apple:  
 states = {0: 'Відсутнє', 1: 'Цвітіння', 2: 'Зелене', 3: 'Червоне'}  
  
 def \_\_init\_\_(self, \_index, \_state=0):  
 self.\_index = \_index  
 self.\_state = \_state  
  
 def grow(self):  
 if not self.is\_ripe():  
 self.\_state += 1  
  
 def is\_ripe(self):  
 return True if self.\_state == 3 else False  
  
 def give\_away(self):  
 self.\_state = 0  
  
 def get\_state(self):  
 return Apple.states[self.\_state]  
  
 def \_\_str\_\_(self):  
 return f'Яблуко #{self.\_index} є {self.get\_state()}'  
  
 class AppleTree:  
 def \_\_init\_\_(self, count=10):  
 self.apples = [Apple(i) for i in range(count)]  
  
 def grow\_all(self):  
 for apple in self.apples:  
 apple.grow()  
  
 def all\_are\_ripe(self):  
 return all([apple.is\_ripe() for apple in self.apples])  
  
 def get\_states(self):  
 return self.apples[0].get\_state()  
  
 def give\_away\_all(self):  
 self.apples = [Apple(i) for i in range(len(self.apples))]  
  
 class Gardener:  
 def \_\_init\_\_(self, name, \_tree=AppleTree()):  
 self.name = name  
 self.\_tree = \_tree  
  
 def work(self):  
 print(f'Садівник {self.name} працює')  
  
 self.\_tree.grow\_all()  
  
 def harvest(self):  
 if self.\_tree.all\_are\_ripe():  
 self.\_tree.give\_away\_all()  
  
 print(f'Садівник {self.name} зібрав усі яблука')  
 else:  
 print(f'Садівник {self.name} не зібрав яблука')  
  
 def get\_count\_apples(self):  
 return len(self.\_tree.apples)  
  
 def get\_states(self):  
 return self.\_tree.get\_states()  
  
 @staticmethod  
 def apple\_base(gardener):  
 print(f'Кількість яблук: {gardener.get\_count\_apples()}')  
 print(f'Стан яблук: {gardener.get\_states()}')  
  
 apple1 = Apple(0)  
 apple2 = Apple(1)  
 print(apple1)  
 print(apple2)  
 apple\_tree = AppleTree(20)  
 gardener = Gardener('Остап', apple\_tree)  
 gardener.work()  
 gardener.work()  
 gardener.harvest()  
 gardener.work()  
 gardener.harvest()  
 Gardener.apple\_base(gardener)

**Результат виконання програми:**



Завдання 4: Створіть клас KmrCsv, який має два атрибути класу за замовчуванням: ref (посилання на CSV файл з оцінками) і num (номер КМР), та методи для встановлення і, відповідно, визначення посилання на файл з оцінками, встановлення номеру КМР, читання файлу з оцінками та виведення інформації про файл (номер КМР і кількість студентів, що її виконали).

Створіть клас Statistic, що містить наступні методи:

- avg\_stat() визначає відсотки правильних відповідей на кожне питання серед усіх студентів і повертає результат у вигляді кортежу чисел;

- метод marks\_stat() визначає яку оцінку набрала відповідна кількість студентів і повертає результат у викляді словника формату {оцінка: кількість студентів};

- метод marks\_per\_time() визначає який середній бал за хвилину набирав студент за під час виконання КМР і повертає результат у вигляді словника формату {id студента (це перша колонка csv файлу): середній бал за хвилину} ;

- метод best\_marks\_per\_time(), який приймає два аргументи bottom\_margin і top\_margin (нижня і верхня межа вибірки підсумкових балів за КМР), та формує для цієї вибірки п’ять найкращих результатів середніх балів за хвилину у вигляді кортежу формату (id студента, підсумкова оцінка, середній бал за хвилину).

Створіть клас Plots, що містить наступні методи:

- set\_cat() – встановлює каталог в який зберігатимуться отримані графіки;

- avg\_plot() – приймає кортеж з відсотками правильних відповідей на кожне окреме питання, формує гістограму на його основі і зберігає отриманий графік;

- marks\_plot() – приймає словник з оцінками і кількістю студентів, що їх набрали, формує на його основі гістрограму і зберігає її

- best\_marks\_plot() – формує для п’яти найкращих результатів середніх балів за хвилину гістрограму і зберігає її.

Створіть клас KmrWork, що успадковує класи CsvKmr, Statistic і Plots. В якості аргументів екземпляр класу приймає посилання на csv файл та номер КМР.

Клас KmrWork містить наступні статичні атрибути

- kmrs - в ньому зберігається словник формату {номер КМР: адреса відповідного csv файла}

- cat – каталог для збереження результатів роботи

Крім успадкованих, клас KmrWork містить наступні методи:

- compare\_csv() – виводить на екран і зберігає в txt файл результат порівняння статистики двох КМР (кількість виконаних КМР, середній бал за КМР, середній час виконання КМР);

- compare\_avg\_plots() – виводить на екран і зберігає дві гістограми з відсотками правильних відповідей на кожні окремі питання.

Тести до модуля:

- Створіть об’єкти kmr1 і kmr2 класу KmrWork.

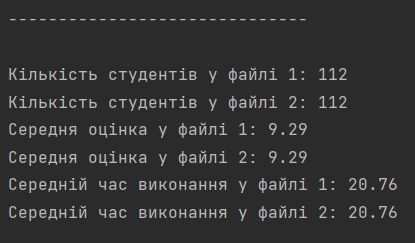
- Використайте для об’єкту kmr2 методи avg\_plot() і marks\_plot()

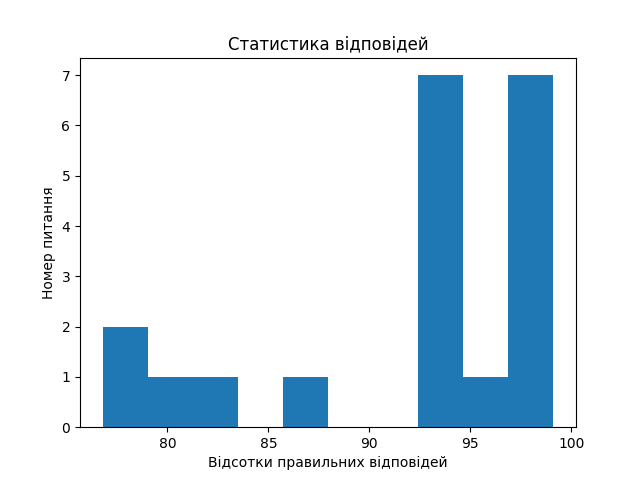
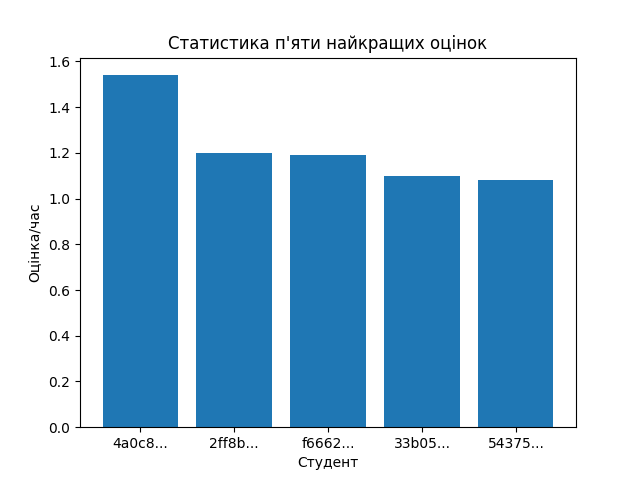
- Для класу KmrWork використайте методи compare\_csv() і compare\_avg\_plots().

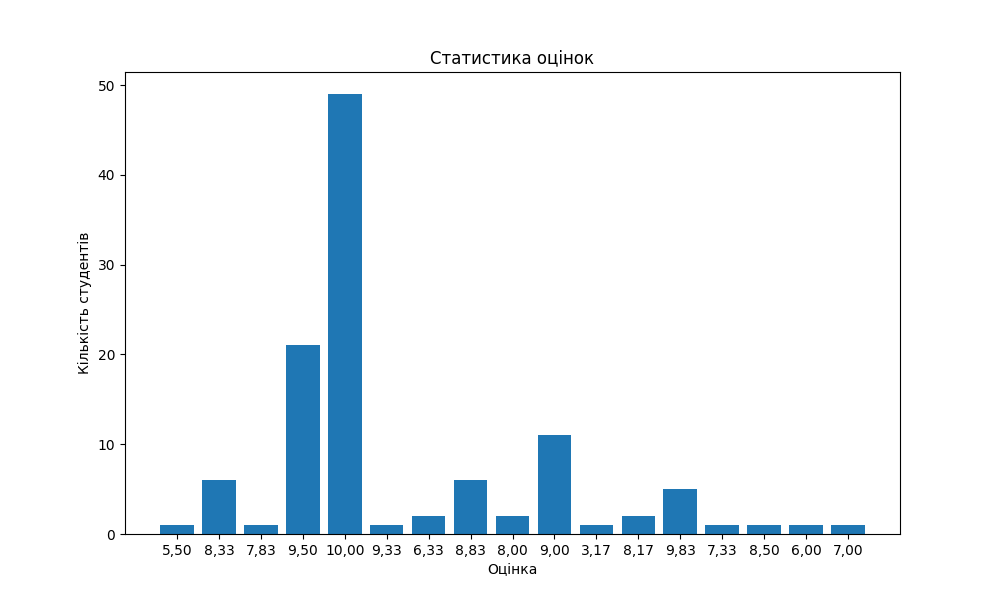
Лістинг програми:

def task4():  
 import csv  
 import os  
 import matplotlib.pyplot as plt  
 import codecs  
  
 class KmrCsv:  
 def \_\_init\_\_(self, num=1, ref='files\\marks2.lab11.csv'):  
 self.ref = ref  
 self.num = num  
  
 def set\_ref(self, ref):  
 self.ref = ref  
  
 def set\_num(self, num):  
 self.num = num  
  
 def \_\_str\_\_(self):  
 return f'Файл: {self.ref}\nНомер: {self.num}'  
  
 class Statistic:  
 def \_\_init\_\_(self, KmrCsv):  
 self.KmrCsv = KmrCsv  
  
 def avg\_stat(self):  
 correct = ()  
  
 with open(self.KmrCsv.ref, 'r') as file:  
 allStudents = len(list(csv.reader(file)))  
 for i in range(20):  
 true = 0  
 file.seek(0)  
 for row in csv.reader(file):  
 try:  
 if float(row[5 + i].replace(',', '.')) == 0.5:  
 true += 1  
 except ValueError:  
 true += 0  
 file.seek(0)  
 percent = round((true \* 100) / allStudents, 2)  
 correct += (percent,)  
 return correct  
  
 def marks\_stat(self):  
 rating = {}  
  
 with open(self.KmrCsv.ref, 'r') as file:  
 for row in csv.reader(file):  
 rating[row[4]] = 0  
 file.seek(0)  
 for row in csv.reader(file):  
 rating[row[4]] += 1  
 return rating  
  
 def marks\_per\_time(self):  
 with open(self.KmrCsv.ref, 'r') as file:  
 file.seek(0)  
 bestRating = {}  
 for row in csv.reader(file):  
 try:  
 time = int(row[3][:2]) + int(row[3][-9:-7]) / 60  
 except:  
 time = int(row[3][:2])  
 rating = float(row[4].replace(',', '.'))  
 bestRating[row[0]] = round(rating / time, 2)  
 return bestRating  
  
 def best\_marks\_per\_time(self, bottom\_margin=0, top\_margin=20):  
 with open(self.KmrCsv.ref, 'r') as file:  
 file.seek(0)  
 bestRating = ()  
 for row in csv.reader(file):  
 try:  
 time = int(row[3][:2]) + int(row[3][-9:-7]) / 60  
 except:  
 time = int(row[3][:2])  
 rating = float(row[4].replace(',', '.'))  
 if bottom\_margin <= rating <= top\_margin:  
 bestRating += ((row[0], rating, round(rating / time, 2)),)  
  
 bestRating = sorted(bestRating, key=lambda x: x[2], reverse=True)  
 return bestRating[:5]  
  
 class Plots:  
 def \_\_init\_\_(self, Statistic, path="files\\graphs"):  
 self.Statistic = Statistic  
 self.set\_cat(path)  
 self.path = path  
  
 def set\_cat(self, path):  
 if not os.path.exists(path):  
 os.mkdir(path)  
 KmrWork.cat = path  
  
 def avg\_plot(self):  
 correctPercent = self.Statistic.avg\_stat()  
 plt.xlabel('Відсотки правильних відповідей')  
 plt.title('Статистика відповідей')  
 plt.ylabel('Номер питання')  
 plt.hist(correctPercent)  
 plt.savefig(f'{self.path}\\[{self.Statistic.KmrCsv.num}] Відсотки правильних відповідей.png')  
 plt.show()  
  
 def marks\_plot(self):  
 rating = self.Statistic.marks\_stat()  
 plt.figure(figsize=(10, 6))  
 plt.title('Статистика оцінок')  
 plt.xlabel('Оцінка')  
 plt.ylabel('Кількість студентів')  
 plt.bar(rating.keys(), rating.values())  
 plt.savefig(f'{self.path}\\[{self.Statistic.KmrCsv.num}] Розподіл оцінок.png')  
 plt.show()  
  
 def best\_marks\_plot(self):  
 bestRating = self.Statistic.best\_marks\_per\_time()  
 plt.title('Статистика п\'яти найкращих оцінок')  
 plt.xlabel('Студент')  
 plt.ylabel('Оцінка/час')  
 plt.bar([i[0][:5] + '...' for i in bestRating], [i[-1] for i in bestRating])  
 plt.savefig(f'{self.path}\\[{self.Statistic.KmrCsv.num}] Кращі оцінки.png')  
 plt.show()  
  
 class KmrWork(KmrCsv, Statistic, Plots):  
 kmrs = {1: 'files\\marks2.lab11.csv'}  
 cat = 'files\\graphs'  
  
 def \_\_init\_\_(self, num=1, ref='files\\marks2.lab11.csv'):  
 KmrCsv.\_\_init\_\_(self, num, ref)  
 Statistic.\_\_init\_\_(self, self)  
 Plots.\_\_init\_\_(self, self)  
 KmrWork.kmrs[num] = ref  
  
 @staticmethod  
 def compare\_csv(kmr1, kmr2):  
 with open(kmr1.ref, 'r') as file1:  
 with open(kmr2.ref, 'r') as file2:  
 with codecs.open(f'files\\Статистика.txt', 'w', "utf-8") as file3:  
 allStudents\_1 = len(list(csv.reader(file1)))  
 allStudents\_2 = len(list(csv.reader(file2)))  
 print(f'Кількість студентів у файлі 1: {allStudents\_1}')  
 print(f'Кількість студентів у файлі 2: {allStudents\_2}')  
 file3.write(f'Кількість студентів у файлі 1: {allStudents\_1}\n')  
 file3.write(f'Кількість студентів у файлі 2: {allStudents\_2}\n')  
 file1.seek(0)  
 file2.seek(0)  
 average\_rating\_1 = 0  
 average\_rating\_2 = 0  
 average\_time\_1 = 0  
 average\_time\_2 = 0  
 for row1 in csv.reader(file1):  
 average\_rating\_1 += float(row1[4].replace(',', '.'))  
 try:  
 average\_time\_1 += int(row1[3][:2]) + int(row1[3][-9:-7]) / 60  
 except:  
 average\_time\_1 += int(row1[3][:2])  
 average\_rating\_1 = round(average\_rating\_1 / allStudents\_1, 2)  
 average\_time\_1 = round(average\_time\_1 / allStudents\_1, 2)  
 for row2 in csv.reader(file2):  
 average\_rating\_2 += float(row2[4].replace(',', '.'))  
 try:  
 average\_time\_2 += int(row2[3][:2]) + int(row2[3][-9:-7]) / 60  
 except:  
 average\_time\_2 += int(row2[3][:2])  
 average\_rating\_2 = round(average\_rating\_2 / allStudents\_2, 2)  
 average\_time\_2 = round(average\_time\_2 / allStudents\_2, 2)  
 print(f'Середня оцінка у файлі 1: {average\_rating\_1}')  
 print(f'Середня оцінка у файлі 2: {average\_rating\_2}')  
 file3.write(f'Середня оцінка у файлі 1: {average\_rating\_1}\n')  
 file3.write(f'Середня оцінка у файлі 2: {average\_rating\_2}\n')  
 print(f'Середній час виконання у файлі 1: {average\_time\_1}')  
 print(f'Середній час виконання у файлі 2: {average\_time\_2}')  
 file3.write(f'Середній час виконання у файлі 1: {average\_time\_1}\n')  
 file3.write(f'Середній час виконання у файлі 2:{average\_time\_2}\n') \  
  
 @staticmethod  
 def compare\_avg\_plots(kmr1, kmr2):  
 correctPercent = kmr1.Statistic.avg\_stat()  
 plt.figure('Відсотки правильних відповідей 1')  
 plt.hist(correctPercent)  
 plt.savefig(f'{KmrWork.cat}\\[{kmr1.Statistic.KmrCsv.num}] Відсотки правильних відповідей 1.png')  
 correctPercent = kmr2.Statistic.avg\_stat()  
 plt.figure('Відсотки правильних відповідей 2')  
 plt.hist(correctPercent)  
 plt.savefig(f'{KmrWork.cat}\\[{kmr2.Statistic.KmrCsv.num}] Відсотки правильних відповідей 2.png')  
 plt.show()  
 kmr1 = KmrWork(1)  
 kmr2 = KmrWork(2)  
  
 kmr1.set\_cat('files\\graphs')  
  
 kmr2.avg\_plot()  
 kmr2.marks\_plot()  
  
 KmrWork.compare\_csv(kmr1, kmr2)  
 KmrWork.compare\_avg\_plots(kmr1, kmr2)  
  
 kmr1.best\_marks\_plot()

Результат виконання програми:







**Висновок:** у ході виконання лабораторної роботи ми ознайомилися з ООП, множинним наслідуванням, міксинами в мові Python.