Міністерство освіти і науки України
Державний університет „Житомирська політехніка”

Кафедра ФІКТ
Група:ВТ-21-1

Програмування мовою Python
Лабораторна робота №
«КЛАСИ. Ч. 3»

Виконав: Вигнич О. С.
Прийняв: Морозов Д. С.

					ІРТР.420001.123-ЗЛ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Вигнич О.С..			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		Морозов Д. С.					
Керівник						ФІКТ, гр. ВТ-21-1	
Н. контр.							
Затверд.							

Мета роботи: ознайомитися з алгоритмами послідовної (лінійної) структури, з процедурами запуску програм, які реалізують ці алгоритми на мові Python; знайомство з інтегрованим середовищем розробки – integrated development environment (IDLE).

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 украïхнського 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

```
class Alphabet:
    Lang = "UA"
    Letters =
["б","г","ґ","д","ж","з","к","л","м","н","п","р","с","т","ф","х","ц","ч","ш","щ","
а","е","є","и","і","ї","о","у","ю","я"]
    def __init__(self, lang = Lang, letters = Letters):
        self.lang = lang
        self.letters = letters
    def  print_alphabet(self):
```

```
        print(self.letters)
    def  letters_num(self):
        return len(self.letters)
    def is_ua_lang(self, str: str):
        Letters = ["б", "г", "ґ", "д", "ж", "з", "л", "н", "ф", "ц", "ч", "ш",
"щ",
                   "а", "е", "є", "и", "і", "ю", "я"]
        str = str.lower()
        for i in Letters:
            for j in str:
                if i == j:
                    return "Це українська мова"
                    break
        return "Це не українська мова"
class  EngAlphabet(Alphabet):
    def __init__(self, lang = "En", letters = ["q", "w", "e", "r", "t", "y", "u",
"i", "o", "p", "a", "s", "d", "f", "g", "h", "j", "k", "l", "z", "x", "c", "v",
"b", "n", "m"]):
        super().__init__(lang, letters)
        self.__en_letters_num = len(letters)
    def  is_en_letter(self, str:str):
        letters = ["q", "w", "r", "t", "u", "s", "d", "f", "g", "h", "j", "k",
"l", "z", "v", "b", "n", "m"]
        str = str.lower()
        if str in letters:
            return "Це англійська літера"
        return "Це не англійська літера"

    def letters_num(self):
        return self.__en_letters_num

    @staticmethod
    def example():
        print("Some text")

test = EngAlphabet()
test.print_alphabet()
print(test.letters_num())
print(test.is_en_letter("J"))
print(test.is_ua_lang("Щ"))
print(EngAlphabet.example())
```

```
['q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', 'a', 's', 'd', 'f', 'g', 'h',
26
Це англійська літера
Це українська мова
Some text
```

2. Створіть клас Human. Визначте для нього два статичних атрибути:

default_name і default_age. Його метод __init __(), який крім self приймає 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.

```
class House:
    def __init__(self, area=100, price=100):
        self.area = area
        self.price = price
    def final_price(self, discount):
        return self.price * (1-(discount/100))
    def __str__(self):
        return (f"area=>{self.area} price=>{self.price}")
class SmallHouse(House):
    def __init__(self, price = 100):
        self.area = 40
```

```
        self.price = price
class Human:
    default_name = "Name"
    default_age = 101
    def __init__(self, money, house, name=default_name, age = default_age):
        self.name = name
        self.age = age
        self.__money = money
        self.__house = house
    def info(self):
        print(f"money=>{self.__money} house=>{self.__house} name=>{self.name}
age=>{self.age}")

    @staticmethod
    def defaulte_info():
        print(f"name=>{Human.default_name} age=>{Human.default_age}")


    def __make_deal(self, price):
        self.__money -= price

    def earn_money(self, money):
        self.__money += money

    def buy_house(self, house, discount =10):
        if self.__money < house.price:
            print("Грошей не достатньо")
        else:
            price = house.final_price(discount)
            self.__make_deal(price)
            print("Куплено")



Human.defaulte_info()
house = House(200, 200_000)
testHuman = Human(50_000, house)
testHuman.info()
sh = SmallHouse(100_000)
testHuman.buy_house(sh)
testHuman.earn_money(500_000)
testHuman.buy_house(sh)
testHuman.info()
```

```
name=>Name age=>101
money=>50000 house=>area=>200 price=>200000 name=>Name age=>101
Грошей не достатньо
Куплено
money=>460000.0 house=>area=>200 price=>200000 name=>Name age=>101
```

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 атриути: 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.

```
class Apple:
    states = ["Відсутнє", "Цвітіння", "Зелене", "Червоне"]
    def __init__(self, index, state = states[0]):
        self.index = index
        self.state = state

    def grow(self):
        if self.state != "Червоне":
            self.state = self.states[self.states.index(self.state) +1]

    def  is_ripe(self):
        if self.state == "Червоне":
```

```
            return "Дозріло"
        return "Не дозріло"
    def __str__(self):
        return f"state=>{self.state} id=>{self.index}"

class AppleTree:
    def __init__(self, a):
        self.Apples = []
        self.Apples.extend(a)

    def grow_all(self):
        states = ["Відсутнє", "Цвітіння", "Зелене", "Червоне"]
        for i in self.Apples:
            i.state = states[states.index(i.state) + 1]

    def all_are_ripe(self):
        for i in self.Apples:
            if i.state != "Червоне":
                return "Не дозріло"

        return "дозріло"

    def give_away_all(self):
        self.Apples.clear()

class Gardener:
    def __init__(self, name, tree: AppleTree):
        self.name = name
        self._tree = tree
    def work(self):
        if self._tree.all_are_ripe() != 'дозріло':
            print("Ще не дозріло")
            for i in self._tree.Apples:
                i.grow()
        else:
            print("Збір")
            self._tree.give_away_all()

    def harvest(self):
        self._tree.all_are_ripe()

    def apple_base(self):
        print(len(self._tree.Apples))
        for i in self._tree.Apples:
            print(i)



a1 = Apple(1)
print(a1)
a2 = Apple(2)
print(a2)
a3 = Apple(3)
print(a3)
tr = AppleTree([a1,a2,a3])
gd = Gardener("Alex", tr)
gd.work()
gd.work()
gd.work()
gd.work()
```

```
state=>Відсутнє id=>1
state=>Відсутнє id=>2
state=>Відсутнє id=>3
Ще не дозріло
Ще не дозріло
Ще не дозріло
Збір
```

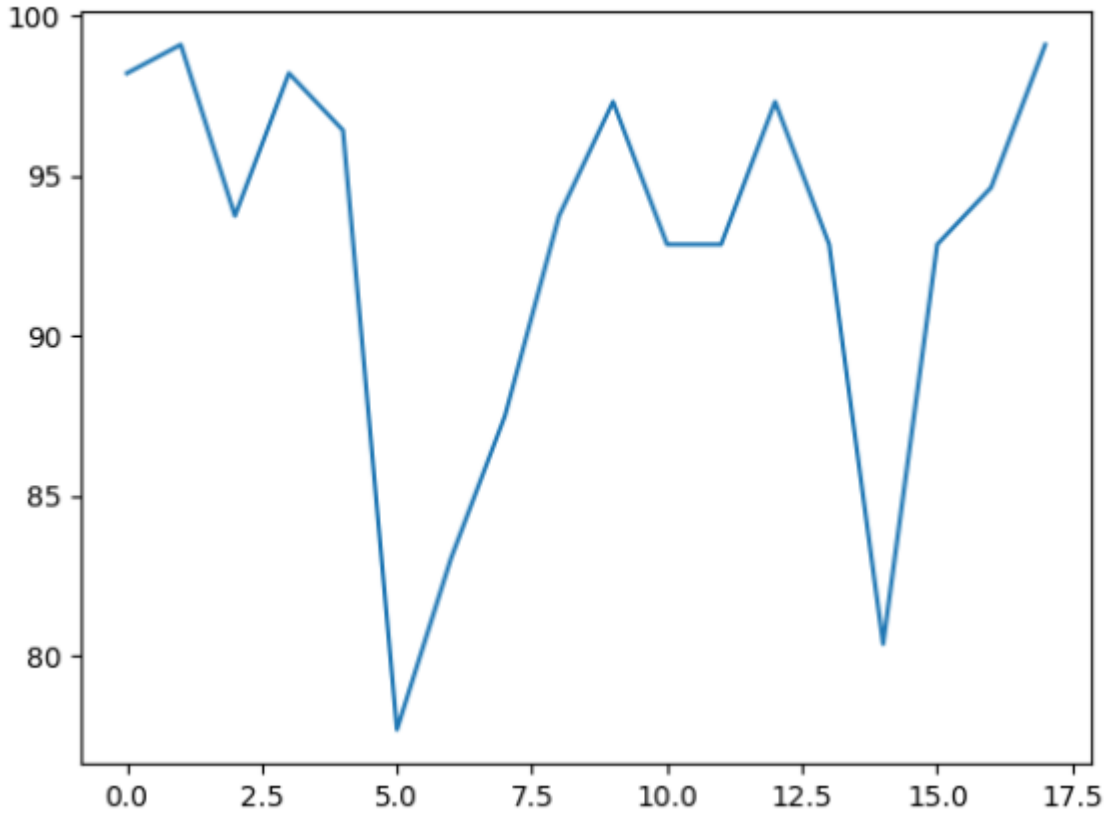
4. Створіть клас KmrCsv, який має два атрибути класу за замовчуванням: ref (посилання на CSV файл з оцінками) і num (номер КМР), та методи для встановлення і, відповідно, визначення посилання на файл з оцінками, встановлення номеру КМР, читання файлу з оцінками та виведення інформації про файл (номер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

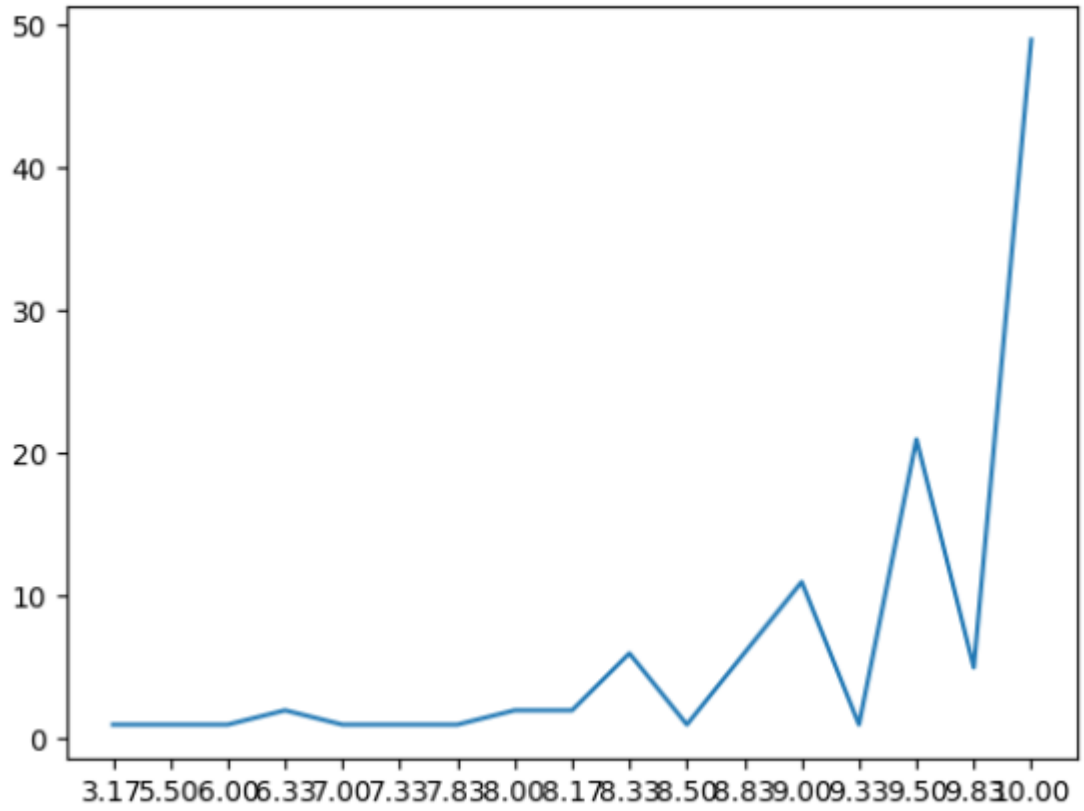
балів за хвилину гістрограму і зберігає її.
Створіть клас KmrWork, що успадковує класи CsvKmr, Statistic і Plots. В якості аргументів екземпляр класу приймає посилання на csv файл та номер КМР.
Клас KmrWork містить наступні статичні атрибути
- kmrs - в ньому зберігається словник формату {номер КМР: адреса 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().

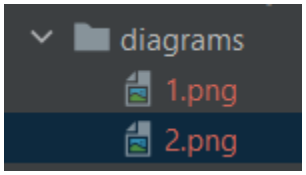
```
import csv
import matplotlib.pyplot as plt
import os

class KmrCsv:
    def __init__(self, ref = "marks2.lab11.csv", num = 2):
        self.ref = ref
        self.num = num
class Statistic:
    def __init__(self, ref ="marks2.lab11.csv" ):
        with open("marks2.lab11.csv", 'rt', encoding="utf8") as freading:
            creading = csv.reader(freading)
            students = [row for row in creading]
            self.students = students
    def avg_stat(self):
        with open("marks2.lab11.csv", 'rt', encoding="utf8") as freading:
            creading = csv.reader(freading)
            students = [row for row in creading]
            self.students = students
        arrBoolAnswer = [i[7:] for i in self.students]
        arr =[]
        for i in range(len(arrBoolAnswer[0])):
            counter = 0
            for j in arrBoolAnswer:
                if float(j[i].replace(",", ".").replace("-", "0")) > 0: counter +=
1
            arr.append([i, (100 / (len(self.students) / counter))])
        return arr

    def marks_stat(self):
        with open("marks2.lab11.csv", 'rt', encoding="utf8") as freading:
            creading = csv.reader(freading)
            students = [row for row in creading]
            self.students = students
```

```
        arr =[]
        arrMark = sorted([*set([i[4].replace(',', '.') for i in self.students])],
key=float)
        for i in arrMark:
            counter = 0
            for j in self.students:
                if (i.replace(".", ",") == j[4]): counter += 1
            arr.append([i, counter])
        return arr

    def marks_per_time(self):
        def ConvertMinutes(time, mark):
            time = time.replace(" хв", "").replace(" сек", "")
            arr = time.split(" ")
            return (mark / int(arr[0]))

        arrTimeMark = [[i[0], ConvertMinutes(i[3], float(i[4].replace(",", ".")))]
for i in self.students]
        return arrTimeMark

    def best_marks_per_time(self, bottom_margin, top_margin):
        arr = []
        def ConvertMinutes(time, mark):
            time = time.replace(" хв", "").replace(" сек", "")
            arr = time.split(" ")
            return (mark / int(arr[0]))

        arrTimeMark = [[i[0], ConvertMinutes(i[3], float(i[4].replace(",", "."))),
float(i[4].replace(",", "."))] for i in self.students]
        arrTimeMark.sort(key=lambda x: x[1], reverse=True)
        print("Top 5 students mark/time\n")
        counter = 0
        for i in arrTimeMark:
            if i[2] > bottom_margin and i[2] < top_margin and counter < 5:
                arr.append(i)
                counter += 1
        return arr
class Plots:
    def set_cat(self, name):
        os.mkdir(name)
    def avg_plot(self, arr):
        arr1 = [i[0] for i in arr]
        arr2 = [i[1] for i in arr]
        plt.plot(arr1, arr2)
        plt.show()
        plt.savefig(f'diagrams\\foo.png')
    def marks_plot(self, arr):
        arr1 = [i[0] for i in arr]
        arr2 = [i[1] for i in arr]
        plt.plot(arr1, arr2)
        plt.show()
        plt.savefig(f'diagrams\\2.png')
class KmrWork(KmrCsv, Statistic, Plots):
    kmrs = "marks2.lab11.csv"
    cat = "diagrams"
    def __init__(self):
        self.path = "marks2.lab11.csv"


s = Statistic()
arr = s.avg_stat()
```

```
print(arr)


mk = KmrWork()
mk.avg_plot(mk.avg_stat())
mk.marks_plot(mk.marks_stat())
```


100
95
90
85
80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


50
40
30
20
10
0
3.17
5.50
6.00
6.33
7.00
7.33
7.83
8.00
8.17
8.33
8.50
8.83
9.00
9.33
9.50
9.83
10.00


diagrams
1.png
2.png