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.

**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

україхнського 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

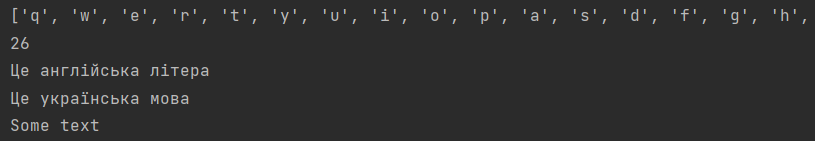
- Виведіть кількість букв в алфавіті

- 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", "ї", "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())



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

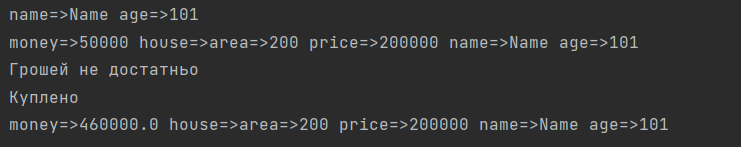
попередження.

- 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()



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

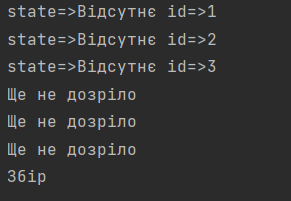
деревом.

- 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()



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())

