**ЛАБОРАТОРНА РОБОТА №9**

**КЛАСИ. Ч. 3.**

**Мета:** 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 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

Лістинг програми:

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'.split(  
 ', ')  
  
 def \_\_init\_\_(self, lang: str = lang, letters: list[str] = letters) -> None:  
 Alphabet.lang = lang  
 Alphabet.letters = letters  
  
 @classmethod # вивід алфавіту  
 def print\_alphabet(cls) -> None:  
 print(f'Алфавіт: {", ".join(cls.letters)}')  
  
 @classmethod # клкст. усіх літер  
 def letters\_num(cls) -> int:  
 return len(cls.letters)  
  
 @classmethod # перевірити чи мова українська  
 def is\_ua\_lang(cls, text: str) -> bool:  
 return all([sign.lower() in ''.join(  
 cls.letters).lower() # перетворюєм великі літери у малі, повертаємо тру якщо літера укр  
 if sign.lower().isalpha() else True  
 for sign in text])  
  
  
class EngAlphabet(Alphabet):  
 lang = 'English'  
 letters = 'a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z'.split(', ')  
 \_\_en\_letters\_num = len(letters)  
  
 def \_\_init\_\_(self, lang: str = lang, letters: str = ''.join(letters)) -> None:  
 EngAlphabet.lang = lang  
 EngAlphabet.letters = letters  
 EngAlphabet.\_\_en\_letters\_num = len(letters)  
  
 @classmethod # перевірка чи мова англ  
 def is\_en\_letter(cls, text: str) -> bool:  
 return all(sign.lower() in ''.join(  
 cls.letters).lower() # перетворюєм великі літери у малі, повертаємо тру якщо літера англ  
 if sign.lower().isalpha() else True for sign in text)  
  
 @classmethod # кількість літер  
 def letters\_num(cls) -> int:  
 return cls.\_\_en\_letters\_num  
  
 @staticmethod # текст англ мовою  
 def example() -> str:  
 return 'Python the best!!!'  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 ENG = EngAlphabet()  
 EngAlphabet.print\_alphabet()  
 print(f'Кількість букв: {ENG.letters\_num()}')  
 print(f'Чи відноситься буква J до англійського алфавіту: {ENG.is\_en\_letter("J")}')  
 print(f'Чи відноситься буква Щ до українського алфавіту: {Alphabet.is\_ua\_lang("Щ")}')  
 print(ENG.example())

Результат виконання програми:

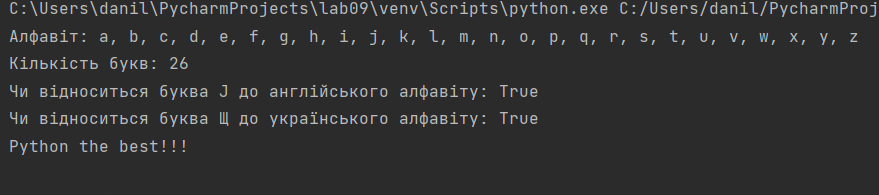


Рис. 1. Результат виконання програми

**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:

class House:  
 def \_\_init\_\_(self, area: float = 100, price: float = 1000000) -> None:  
 self.\_area = area  
 self.\_price = price  
  
 def \_\_str\_\_(self) -> str:  
 return f'Будинок(площа: {self.\_area}, ціна: {self.\_price})'  
  
 def final\_price(self, discount: float) -> float: # ціна зі знижкою  
 return self.\_price - self.\_price \* (discount / 100)  
  
  
class SmallHouse(House):  
 def \_\_init\_\_(self) -> None:  
 super().\_\_init\_\_(area=40, price=100000) # супер клас для доступа до батьківського класу  
  
  
class Human:  
 default\_name = 'Undefined'  
 default\_age = -1  
  
 def \_\_init\_\_(self, name: str, age: int, money: float, house: House) -> None:  
 self.name = name  
 self.age = age  
 self.\_\_money = money  
 self.\_\_house = house  
  
 @classmethod # вивід станд. інформації  
 def default\_info(cls):  
 print(f'Стандартне ім\'я: {cls.default\_name}')  
 print(f'Стандартний вік: {cls.default\_age}')  
  
 def info(self): # вивід інформації про власника і будинок  
 print(f'Ім\'я: {self.name}')  
 print(f'Вік: {self.age}')  
 print(f'Гроші: {self.\_\_money}')  
 print(f'Будинок: {self.\_\_house}')  
  
 def earn\_money(self, value: float): # заробити гроші  
 self.\_\_money += value  
  
 def buy\_house(self, house: House, discount: float = 10): # купити будинок  
 if self.\_\_money < house.\_price: # якщо невистачає грошей  
 print('Недостатньо грошей!')  
 return None  
 self.\_\_money -= house.final\_price(discount)  
 self.\_\_house = house  
 print(f'Успішно куплено: {house}') # будинок куплено!  
  
 def \_\_make\_deal(self, house: House): # заключити угоду  
 self.buy\_house(house)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 Human.default\_info()  
 human = Human('Danilo\_Savchenko', 19, 10000, None)  
 human.info()  
 shs = SmallHouse()  
 human.buy\_house(shs)  
 human.earn\_money(110000)  
 human.buy\_house(shs)  
 human.info()

Результат виконання програми:

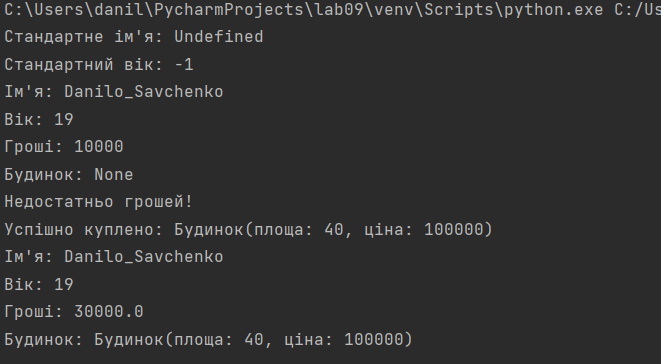


Рис. 2. Результат виконання програми.

**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 будеперевіряти, 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.

Лістинг програми:

class Apple: # apple  
 states = ("Відсутнє", "Цвітіння", "Зелене", "Червоне")  
 all\_apples = [] # список всіх яблук  
  
 def \_\_init\_\_(self) -> None:  
 self.\_index = len(Apple.all\_apples) + 1  
 self.\_state = Apple.states[0]  
 Apple.all\_apples.append(self)  
  
 def \_\_str\_\_(self) -> str:  
 return f'Яблуко {self.\_index}'  
  
 def grow(self) -> None: # підвищення стиглості яблука  
 i = Apple.states.index(self.\_state)  
 if i + 1 < len(Apple.states):  
 print(f'{self.\_index} росте...')  
 self.\_state = Apple.states[i + 1]  
  
 def is\_ripe(self) -> bool: # перевірка чи яблуко стигле  
 return Apple.states.index(self.\_state) + 1 == len(Apple.states)  
  
 @classmethod # вивід статистики усіх яблук  
 def apple\_base(cls):  
 print(f'Кількість яблук - {len(cls.all\_apples)}')  
 for apple in cls.all\_apples:  
 print(f'{apple} - {apple.\_state}')  
  
  
class AppleTree: # tree  
 def \_\_init\_\_(self, amount: int) -> None: # список усіх існ яблук на дереві  
 self.apples = [Apple() for \_ in range(amount)]  
  
 def \_\_str\_\_(self) -> str:  
 return f'Яблукове дерево({", ".join([str(apple) for apple in self.apples])})' # вивід існ ябл на дереві  
  
 def grow\_all(self) -> None: # підвищити стиглості усіх яблук  
 for apple in self.apples:  
 apple.grow()  
  
 def all\_are\_ripe(self) -> bool: # перевірка чи усі яблука стиглі  
 return all(apple.is\_ripe() for apple in self.apples)  
  
 # очистити список від усіх стигли яблук  
 def give\_away\_all(self):  
 self.apples = list(filter(lambda apple: not apple.is\_ripe(), self.apples))  
  
  
class Gardener: # садівник  
 def \_\_init\_\_(self, name: str, tree: AppleTree) -> None:  
 self.name = name  
 self.\_tree = tree  
  
 def \_\_str\_\_(self) -> str:  
 return f'Садівник {self.name}'  
  
 def work(self): # статус роботи працівника  
 print(f'{self} працює...')  
 self.\_tree.grow\_all()  
  
 def harvest(self): # зібрати яблука  
 if self.\_tree.all\_are\_ripe():  
 self.\_tree.give\_away\_all()  
 else:  
 print('Ще не всі плоди дозріли')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 apple1 = Apple()  
 apple2 = Apple()  
 apple3 = Apple()  
 Apple.apple\_base()  
 tree = AppleTree(4)  
 worker = Gardener('Danilo', tree)  
 print(worker.\_tree)  
 worker.work()  
 worker.harvest()  
 worker.work()  
 worker.work()  
 worker.work()  
 worker.harvest()  
 print(worker.\_tree)

Результат виконання програми:

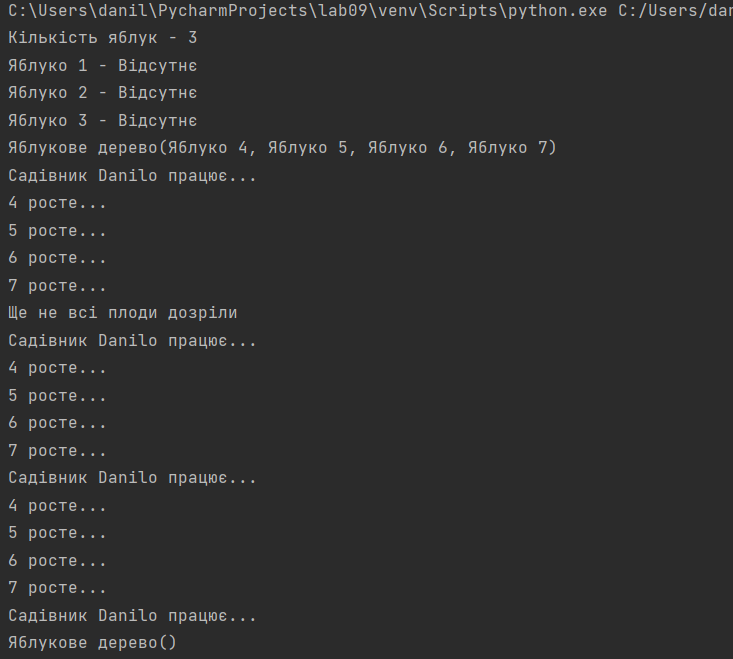


Рис. 3. Результат виконання програми.

**Висновок:** виконуючи лабораторну роботу, ми ознайомилися з ООП, множинним наслідуванням, міксинами в мові Python.