I. ООР

1. **(6б).** Створіть клас **Animal,** додайте три атрибути**,** один з яких має значення за замовчуванням та два методи на свій розсуд**.**

In [31]:

```
class Animal:
    def __init__(self, name, age, species="Animal"):
        self.name = name
        self.age = age
        self.species = species

    def print(self):
        print(f"Name: {self.name}, Age: {self.age}, Species: {self.species}")

    def print_short(self):
        print(f"{self.name} is a {self.age} year old animal")
```

1. **(2б).** Створіть два об'єкти цього класу**,** де один з об'єктів створюється із дефолтним значенням атрибуту**.** На одному об'єкті отримайте значення його атрибуту**,** а на іншому викличте один з його методів**.**

In [32]:

```
barsik = Animal("Barsik", 3, "Cat")
print(barsik.name)
animal = Animal("Something", 13)
animal.print_short()
```

```
Barsik
Something is a 13 year old animal
```

1. **(5б).** Створіть клас**,** де атрибути мають різні рівні доступу**.** Спробуйте отримати їхні значення та опишіть результати**.**

In [33]:

```
class BankCard:
    _cardNumber = 0
    __cvv = 0

    def __init__(self, cardHolderName, cardNumber, cvv):
        self.cardHolderName = cardHolderName
        self._cardNumber = cardNumber
        self.__cvv = cvv

    def print(self):
        print(f"CardHolder: {self.cardHolderName}, CardNumber: {self._cardNumber}, CVV:
***")

    def _print_full(self):
        print(f"CardHolder: {self.cardHolderName}, CardNumber: {self._cardNumber}, CVV:
{self.__cvv}")

card = BankCard("Ivan Ivanov", 1234567890, 123)
card.print()
card._print_full()
print(card._cardNumber)
print(card.__cvv)
```

```
CardHolder: Ivan Ivanov, CardNumber: 1234567890, CVV: ***
CardHolder: Ivan Ivanov, CardNumber: 1234567890, CVV: 123
1234567890
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[33], line 20
     18 card._print_full()
     19 print(card._cardNumber)
---> 20 print(card.__cvv)

AttributeError: 'BankCard' object has no attribute '__cvv'
```

1. **(7б).** Як ви розумієте термін **self?** Для чого використовується метод **init ?**

Self - це ключове слово, яке використовують для позначення об'єкта у контексті класу - самого себе, але не класу, а конкретного екземпляру класу. Метод **__init__** використовується для створення об'єкта - екземляру класу з певним набором значень атрибутів.

1. **(9б).** Створіть клас Фігура без атрибутів, з методом **get_area** для отримання площі фігури, що повертає **0** та **add ,** який приймає **self** та **other** в якості аргументів, а повертає суму площин фігур **self** та **other.**

In [ ]:

```
class Shape:
    def get_area(self):
        return 0

    def __add__(self, other):
        return self.get_area() + other.get_area()
```

1. **(11б).** Створіть **2** дочірніх класи від Фігури: Трикутник та Коло, які мають атрибути, необхідні для розрахунку площин. Визначте метод **get_area** в кожному з них так, щоби вони розраховували площу в залежності від формули для кожного типу фігури. Створіть об'єкт класу Трикутник та об'єкт класу Коло. Виконайте операцію суми за допомогою оператора **+** між ними.

In [ ]:

```
class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def get_area(self):
        return self.base * self.height / 2

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def get_area(self):
        return 3.14 * self.radius ** 2

triangle = Triangle(3, 4)
circle = Circle(5)
print(triangle.get_area())
print(circle.get_area())
print(triangle + circle)
```

```
6.0
78.5
84.5
```

1. **(3б).** Продемонструйте різницю між **isinstance** та **issubclass.**

In [35]:

```
print(isinstance(triangle, Triangle))
```

```
print(isinstance(triangle, Shape))
print(isinstance(triangle, Circle))
print("-----")
print(issubclass(Triangle, Triangle))
print(issubclass(Triangle, Shape))
print(issubclass(Triangle, Circle))
```

```
True
True
False
-----
True
True
False
```

1. **(13б).** Створіть клас **BankAccount** з приватними атрибутами **balance** та **account_number.** Реалізуйте методи поповнення та зняття коштів**,** забезпечивши належну інкапсуляцію**.** Підказка**:** використовуйте декоратори **property** та **setter.**

In [ ]:

```
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.__account_number = account_number
        self.__balance = balance

    @property
    def balance(self):
        return self.__balance

    @balance.setter
    def balance(self, amount):
        if amount < 0:
            print("Balance must be positive")
            return
        self.__balance = amount

    def deposit(self, amount):
        if amount <= 0:
            print("Deposit amount must be positive")
            return
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= 0 or amount > self.__balance:
            print("Withdraw amount must be positive and less than balance")
            return
        self.__balance -= amount

account = BankAccount(1234567890, 100)
print(account.balance)
account.deposit(50)
print(account.balance)
account.withdraw(20)
print(account.balance)
#account.__balance += 100
```

```
100
150
130
```

1. **(11б).** Створіть клас **Library,** який містить список об'єктів типу **Book.** Реалізуйте методи для додавання книги**,** видалення книги та відображення списку книг**.**

In [ ]:

```
class Book:
    def __init__(self, title, author):
        self.title = title
```

```
        self.author = author

    def __str__(self):
        return f"'{self.title}' by {self.author}"


class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        if isinstance(book, Book):
            self.books.append(book)
        else:
            print("The object is not a book")

    def remove_book(self, book):
        if book in self.books:
            self.books.remove(book)
        else:
            print("The book is not in the library")

    def display_books(self):
        for book in self.books:
            print(book)

lib = Library()
book1 = Book("Harry Potter", "J.K. Rowling")
book2 = Book("HGTTG", "Douglas Adams")
lib.add_book(book1)
lib.display_books()
print()
lib.add_book(book2)
lib.display_books()
lib.remove_book(book1)
print()
lib.display_books()
```

```
'Harry Potter' by J.K. Rowling

'Harry Potter' by J.K. Rowling
'HGTTG' by Douglas Adams

'HGTTG' by Douglas Adams
```

1. **(13б).** Створіть клас **Person** з атрибутами **name** та **age.** Створіть ще один клас **Employee** з такими атрибутами**,** як **department** та **salary.** Створіть клас **Manager,** який успадковує обидва класи **Person** та **Employee.** Продемонструйте використання множинної спадковості**,** створивши об'єкт класу **Manager** та отримавши **mro** для цього класу**.**

In [ ]:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


class Employee:
    def __init__(self, department, salary):
        self.department = department
        self.salary = salary


class Manager(Person, Employee):
    def __init__(self, name, age, department, salary):
        Person.__init__(self, name, age)
        Employee.__init__(self, department, salary)

manager = Manager('John Doe', 45, 'Sales', 80000)
```

```
print(f"Name: {manager.name}, Age: {manager.age}, Department: {manager.department}, Salar
y: {manager.salary}")

print(Manager.mro())
```

```
Name: John Doe, Age: 45, Department: Sales, Salary: 80000
[<class '__main__.Manager'>, <class '__main__.Person'>, <class '__main__.Employee'>, <cla
ss 'object'>]
```

II. Iterator

1. **(4б).** Визначте рядок**(str)** з **4**ма різними за значенням символами**.** Створіть ітератор на основі цього рядка**.** Викличте **5** разів функцію **next** на ітераторі**, 4**ри перших з них огорніть у ф-цію **print().**

In [ ]:

```
s = "str("

iterator = iter(s)

for _ in range(4):
    print(next(iterator))

next(iterator)
```

```
s
t
r
(

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Cell In[24], line 8
      5 for _ in range(4):
      6     print(next(iterator))
----> 8 next(iterator)

StopIteration:
```

А що якщо викликати цю ж функцію на рядку**?**

In [ ]:

```
print(next("str"))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 print(next("str"))

TypeError: 'str' object is not an iterator
```

1. **(5б).** Опишіть своїми словами в одному реченні**,** як ви розумієте різницю між ітерабельними об'єктами та ітераторами **(**можна на прикладі**).**

Ітерабельний об'єкт **-** це коллекція**,** яку можна перебирати**;** ітератор **-** це об'єкт**,** який перебирає цю колекцію і слідкує за станом перебору **(**який поточний елемент**,** який наступний**).**

1. **(11б).** Створіть клас**,** що має визначений **Iterator Protocol** та при кожному виклику **next** повертає літери англійської абетки**,** поки вони не вичерпаються**.**

In [ ]:

```
class AlphabetIterator:
```

```
class AlphabetIterator:
    def __init__(self):
        self.letters = 'abcdefghijklmnopqrstuvwxyz'
        self.i = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.i < len(self.letters):
            result = self.letters[self.i]
            self.i += 1
            return result
        else:
            raise StopIteration

alphabet = AlphabetIterator()

print(next(alphabet))
for letter in alphabet:
    print(letter)
```

```
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
```

Вітаю! Ви велика(ий) молодець, що впоралась(вся). Похваліть себе та побалуйте чимось приємним. Я Вами пишаюся.