```python
class Animal:
    def __init__(self, name, food_requirement):
        self.name = name
        self.food_requirement = food_requirement

    def move(self):
        pass

class Млекопитающее(Animal):
    def __init__(self, name, food_requirement, wool):
        super().__init__(name, food_requirement)
        self.wool = wool

    def feed_milk(self):
        return f"{self.name} вскармливает детёнышей молоком"

class Птица(Animal):
    def __init__(self, name, food_requirement, feather_color):
        super().__init__(name, food_requirement)
        self.feather_color = feather_color

    def fly(self):
        return "Может летать" if self.flies else "не умеет летать"

class Лев(Млекопитающее):
    def __init__(self, name, pride_size):
        super().__init__(name, "мясо", "густая")
        self.pride_size = pride_size
        self.__energy = 60  # это приватный атрибут а можем понять что он приватный по двум нижним подчеркиваниям

    def hunting(self):
        return f"{self.name} охотится в прайде из {self.pride_size} львов"

    def move(self):
        return f"{self.name} бегает по полю"

    def perform_action(self, food, action):
        if action == "поесть":
            if food == self.food_requirement:
                self.__energy += 20  # Увеличиваем энергию
                return f"{self.name} поел и теперь у него уровень энергии {self.__energy}"
            else:
                return f"{self.name} не ест {food}."

        if action == "рычать":
            if food == self.food_requirement:
                self.__energy -= 20  # Теряет энергию
                return f"{self.name} рычит! Теперь у него уровень энергии {self.__energy}"
            else:
                return f"{self.name} не рычит! {food}"

        return f"не теряет энергию"

    def get_energy(self):
        return self.__energy  #показывает сколько энергий

class Слон(Млекопитающее):
    def __init__(self, name, herd_size):
        super().__init__(name, "Травка", "короткая")
        self.herd_size = herd_size

    def remembers(self):
        return f"{self.name} запоминает информацию"

    def move(self):
        return f"{self.name} медленно шагает"

class Попугай(Птица):
    def __init__(self, name):
        super().__init__(name, "семена", "яркие")
        self.flies = True
        self.pair = None
```

```
    def imitate_sound(self, sound):
        return f"{self.name} имитирует звук: '{sound}'"

    def move(self):
        return f"{self.name} перелетает с ветки на ветку"

class Пингвин(Птица):
    def __init__(self, name):
        super().__init__(name, "рыбой", "чёрно-белые")
        self.flies = False

    def swims(self):
        return f"{self.name} плавает в воде"

    def move(self):
        return f"{self.name} топает топ топ"

if __name__ == "__main__":
    лев = Лев("Лев", 9)
    слон = Слон("Слон", 6)
    попугай = Попугай("Попугай")
    пингвин = Пингвин("Пингвин")

    print(лев.hunting())
    print(слон.remembers())
    print(попугай.imitate_sound("попугая"))
    print(пингвин.swims())
    print(лев.move())
    print(слон.move())
    print(попугай.move())
    print(пингвин.move())

    # задание 2. пример инкапсуляций
    #лев рычит терает энергию и восполняет как в играх. Но думал я долго такое я бы не стал делать на зачёте 👍
    print(лев.perform_action("мясо", "рычать"))
    print(лев.get_energy())
    print(лев.perform_action("мясо", "поесть"))
    print(лев.get_energy())
    print(лев.perform_action("травку", "поесть"))

    # Задание 3. Укажите, какую концепцию объектно-ориентированного программирования иллюстрирует приведённый код, и
объясните суть этой концепции на данном примере
class Animal:
    def make_sound(self):
        raise NotImplementedError("Subclasses should implement this!")

class Dog(Animal):
    def make_sound(self):
        return "Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow!"

def animal_sound(animal: Animal):
    print(animal.make_sound())

    dog = Dog()
    cat = Cat()
    animal_sound(dog)
    animal_sound(cat)
```
в коде полиморфизм. Он позволяет объектам разных под классов Dog и Cat Использовать один и тот же метод make_sound, обеспечивая разные реализации что делает код гибким.

```
    # Задание 4. На какие категории делятся структуры данных по признаку изменяемости? Приведите примеры для каждой
категории
```
структуры данных делятся по признаку изменяемости и не изменяемости в изменяемые входят списки, ?словари? которые можно изменять они перезаписываються в ту же ячейку памяти.
В не изменяемые входят кортежи, числовые, строки, и тд... которые нельзя перезаписать прямо в ячейку памяти, она зписывает в отдельную ячейку памяти.
словари не помню про списки точно помню

```
    # Задание 5. Что выведет этот код?
def process_numbers(numbers):
```

```
    total = 0
    for number in numbers:
        if number % 2 == 0:
            total += number
        elif number % 3 == 0:
            total -= number
        return total


def main():
    data = [5, 12, 7, 9, 18, 4, 11]
    result = process_numbers(data)
    print("Result is:", result)


main()
```

этот код выведет число 25 потому что из списка data проверяються чётные деляться ли они на 2 без остатка
потом берет все чётные 12+18+4 что даёт 34 и потом проверяеться числа которые деляться на 3 без остатка это
9 и вычитает из нашего числа 34-9=25

#коротко

Код суммирует четные числа из data (12, 18, 4) до 34 и вычитает 9 (число, делящееся на 3), в итоге получая 25.