📚 Краткий конспект лекции: Наследование и полиморфизм в Python

🔷 1. Наследование

Наследование — это механизм, позволяющий создавать новый класс на основе существующего. Новый класс (дочерний) **получает все атрибуты и методы** родительского класса и может **расширять или изменять** их.

Синтаксис:

```
class ChildClass(ParentClass):
    # тело класса
```

Пример:

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

class Bird(Animal):
    def __init__(self, name, species, can_fly):
        super().__init__(name, species)  # вызов конструктора родителя
        self.can_fly = can_fly
```

✅ `super()` — позволяет вызвать метод родительского класса (чаще всего — `__init__`).

🔷 2. Переопределение методов

Дочерний класс может **переопределять** методы родителя — задавать своё поведение.

```
class Animal:
    def make_sound(self):
        print("Животное издаёт звук.")

class Bird(Animal):
    def make_sound(self):  # переопределение
        print(f"{self.name} чирикает!")
```

🔷 3. Полиморфизм

Полиморфизм — возможность использовать объекты разных классов **единообразно**, если у них есть **общий интерфейс** (например, одинаковые методы).

Пример:

```
animals = [Bird("Кеша", "Попугай", True), Mammal("Лёва", "Лев")]

for animal in animals:
    animal.make_sound()  # работает по-разному для каждого класса
```

> ✅ Один вызов — разное поведение. Это и есть полиморфизм.

🔷 4. Проверка типа объекта

Иногда нужно выполнить **специфическое действие** только для определённого типа. Используйте:

- `isinstance(obj, Class)` — возвращает `True`, если объект принадлежит классу (или его наследнику).

```
if isinstance(animal, Bird):
    animal.fly()
```

🔷 5. Зачем это нужно?

- **Избегаем дублирования кода** (общее — в родителе).
- **Расширяем функциональность** (специфическое — в потомках).
- **Пишем гибкий код**, который работает с разными типами через единый интерфейс.

🧠 Ключевые слова

Термин	Значение
Родительский класс	Базовый класс, от которого наследуются другие
Дочерний класс	Класс, наследующий от родителя
`super()`	Вызов метода родительского класса
Переопределение	Изменение поведения метода в дочернем классе
Полиморфизм	Единый интерфейс — разное поведение

✅ **Запомните**:

> «Наследование — для повторного использования.
> Полиморфизм — для гибкости.»

🧪 Практическая работа: Зоопарк — Учёт животных

> **Цель**: Создать иерархию классов животных с разным поведением, используя наследование и полиморфизм. Все данные — публичные атрибуты. Управление — через методы.

⏱️ **Время выполнения**: 60 минут
🎯 **Темы**: классы, объекты, методы, наследование, полиморфизм, списки, строки

📚 Сценарий

Вы — администратор зоопарка. Вам нужно создать систему учёта животных. В зоопарке живут **птицы**, **млекопитающие** и **рептилии**. Все они — животные, но издают разные звуки, едят разную еду и ведут себя по-разному.

🎯 Часть 1: Базовый класс `Animal` (15 минут)

Создайте класс `Animal`.

Требования:

- **Атрибуты** (все публичные, устанавливаются в `__init__`):
  - `name` — имя животного (строка)
  - `species` — вид (строка, например, "лев", "орёл")
  - `age` — возраст (целое число)
- **Методы**:
  - `make_sound()` — должен **выводить** сообщение: `"Животное издаёт звук."`
    (Этот метод будет переопределяться в дочерних классах.)
  - `eat(food)` — выводит: `"{name} ест {food}."`
  - `__str__()` — возвращает строку: `"{name} ({species}), возраст: {age}"`

📌 Пример:

```
animal = Animal("Зверь", "Неизвестно", 5)
print(animal)  # Зверь (Неизвестно), возраст: 5
animal.make_sound()  # Животное издаёт звук.
```

🎯 Часть 2: Дочерние классы (25 минут)

Создайте три дочерних класса, наследующих от `Animal`.

1. Класс `Bird(Animal)`

- В `__init__` принимает `name`, `species`, `age`, **дополнительно** — `can_fly` (булево: может ли летать).
- Атрибуты: `name`, `species`, `age`, `can_fly` (все публичные).
- Переопределяет:
  - `make_sound()` → выводит: `"{name} чирикает!"`
  - Добавляет метод `fly()`:
    - Если `can_fly == True` → выводит: `"{name} летает в небе."`
    - Иначе → выводит: `"{name} не может летать."`

2. Класс `Mammal(Animal)`

- В `__init__` принимает `name`, `species`, `age`.
- Переопределяет:
  - `make_sound()` → выводит: `"{name} издаёт звериный звук."`
  - Добавляет метод `run()` → выводит: `"{name} бежит по земле."`

3. Класс `Reptile(Animal)`

- В `__init__` принимает `name`, `species`, `age`.
- Переопределяет:
  - `make_sound()` → выводит: `"{name} шипит."`
  - Добавляет метод `sunbathe()` → выводит: `"{name} греется на солнце."`

> 💡 Во всех дочерних классах **вызывайте** `super().__init__()` для инициализации базовых атрибутов.

🎯 Часть 3: Полиморфизм и управление зоопарком (20 минут)

Создайте функцию **`simulate_zoo()`**, которая:

1. Создаёт 4 животных:
   - `parrot = Bird("Кеша", "Попугай", 3, can_fly=True)`
   - `penguin = Bird("Пинг", "Пингвин", 2, can_fly=False)`
   - `lion = Mammal("Лёва", "Лев", 5)`
   - `snake = Reptile("Сэм", "Удав", 4)`
2. Помещает их в список: `animals = [parrot, penguin, lion, snake]`
3. Для каждого животного в списке:
   - Выводит его через `print()`
   - Вызывает `make_sound()`
   - Вызывает `eat("корм")`
4. Затем — **специфическое поведение** (используйте `type()` или `isinstance()`):

- Если это `Bird` → вызывает `fly()`
- Если это `Mammal` → вызывает `run()`
- Если это `Reptile` → вызывает `sunbathe()`

📌 Пример части вывода:

```
Кеша (Попугай), возраст: 3
Кеша чирикает!
Кеша ест корм.
Кеша летает в небе.

Пинг (Пингвин), возраст: 2
Пинг чирикает!
Пинг ест корм.
Пинг не может летать.

Лёва (Лев), возраст: 5
Лёва издаёт звериный звук!
Лёва ест корм.
Лёва бежит по земле.
```

📥 Требования к сдаче

- Один файл `.py` с классами `Animal`, `Bird`, `Mammal`, `Reptile` и функцией `simulate_zoo()`.
- В конце файла — вызов `simulate_zoo()` для демонстрации работы.