**Лекция 8**

**ООП (Объектно-Ориентированное Программирование)**

*(Как моделировать реальный мир в коде)*

**1. Основные понятия**

* **Класс** — шаблон для создания объектов (например, класс Собака).
* **Объект** — экземпляр класса (например, конкретная собака «Шарик»).
* **Атрибуты** — переменные, описывающие объект (вес, возраст).
* **Методы** — функции, которые может выполнять объект (лаять, бежать).

**2. Создание класса**

class Cat:

# Конструктор (вызывается при создании объекта)

def \_\_init\_\_(self, name, age):

self.name = name # Атрибут объекта

self.age = age

# Метод

def meow(self):

print(f"{self.name} говорит: Мяу!")

# Создание объекта

my\_cat = Cat("Барсик", 3)

my\_cat.meow() # Барсик говорит: Мяу!

**Пояснение:**

* self — ссылка на сам объект (как я в русском языке).
* \_\_init\_\_ — метод-конструктор, инициализирует атрибуты.

**3. Наследование**

Создаем класс-потомок, который наследует атрибуты и методы родителя:

class Robot:

def \_\_init\_\_(self, name):

self.name = name

def say\_hello(self):

print(f"Я {self.name}. Приветствую!")

class BattleRobot(Robot): # Наследование

def attack(self):

print(f"{self.name} атакует лазером!")

# Использование

terminator = BattleRobot("T-800")

terminator.say\_hello() # Я T-800. Приветствую!

terminator.attack() # T-800 атакует лазером!

**4. Инкапсуляция и полиморфизм**

**Инкапсуляция** — сокрытие внутренней реализации:

class BankAccount:

def \_\_init\_\_(self):

self.\_\_balance = 0 # Приватный атрибут (два подчеркивания)

def deposit(self, amount):

if amount > 0:

self.\_\_balance += amount

def get\_balance(self):

return self.\_\_balance

account = BankAccount()

account.deposit(100)

print(account.get\_balance()) # 100

# account.\_\_balance = 200 # Ошибка! Нельзя изменить напрямую.

**Полиморфизм** — одинаковые методы работают по-разному для разных классов:

class Dog:

def sound(self):

print("Гав!")

class Duck:

def sound(self):

print("Кря!")

def make\_sound(animal):

animal.sound()

make\_sound(Dog()) # Гав!

make\_sound(Duck()) # Кря!

**5. Зачем нужно ООП?**

* **Упрощение сложных систем** — объекты соответствуют сущностям реального мира.
* **Переиспользование кода** — наследование и композиция.
* **Сокрытие деталей** — пользователь класса не видит внутреннюю логику.

**6. Практический пример: игра**

class Character:

def \_\_init\_\_(self, name, health):

self.name = name

self.health = health

def take\_damage(self, damage):

self.health -= damage

print(f"{self.name} получил урон {damage}. Осталось здоровья: {self.health}")

class Mage(Character):

def cast\_spell(self):

print(f"{self.name} бросает огненный шар!")

# Создаем персонажей

warrior = Character("Воин", 100)

mage = Mage("Маг", 80)

warrior.take\_damage(20) # Воин получил урон 20. Осталось здоровья: 80

mage.cast\_spell() # Маг бросает огненный шар!