Классы и объекты

Что такое классы

Класс - это шаблон или чертёж, по которому создаются объекты. Он описывает атрибуты (переменные) и методы (функции), которые будут присутствовать у объектов, созданных на основе этого класса. Классы позволяют организовать данные и логику в единую сущность.

class название_класса:
  атрибуты_класса
  методы_класса

Объект = Класс (Значения атрибутов)

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

    def introduce(self):
            return f"Привет, меня зовут {self.name} и мне {self.age} лет."

    def celebrate_birthday(self):
        self.age += 1
            return f"С днём рождения! Теперь мне {self.age} лет."

# Создание объекта (экземпляра) класса
person1 = Person("Иван", 30)

# Вызов методов
print(person1.introduce())   # Выведет: "Привет, меня зовут Иван и мне 30 лет."

# Празднование дня рождения
print(person1.celebrate_birthday())  # Выведет: "С днём рождения! Теперь
```

Методы классов

Методы - это функции, определенные внутри класса. Они предоставляют способ организации логики и поведения объектов. Методы могут работать с атрибутами этого класса.

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

    def introduce(self):
            return f"Привет, меня зовут {self.name} и мне {self.age} лет."

person1 = Person("Иван", 30)

# Вызов методов
print(person1.introduce())
```

ключевое слово self

```
self.атрибут    # обращение к атрибуту
self.метод      # обращение к методу
```

```
class Person:

    def say(self, message):
        print(message)

    def say_hello(self):
            self.say("Hello work")   # обращаемся к выше определенному методу say


tom = Person()
tom.say_hello()     # Hello work
```

Конструкторы и атрибуты объекта

Для создания объекта класса используется конструктор.

```
tom = Person()
```

определить в классах конструктор (действия выполняемые при создании объекта) с помощью специального метода, который называется **__init__()**

```
class Person:
    # конструктор
    def __init__(self):
        print("Создание объекта Person")

    def say_hello(self):
        print("Hello")


tom = Person()      # Создание объекта Person
tom.say_hello()     # Hello
```

Атрибуты объекта - хранят состояние объекта (переменные).

```
class Person:

    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 1        # возраст человека
tom = Person("Tom")
print(tom.name)     # Tom
print(tom.age) # 1
```

Можно определить атрибут вне класса

```
class Person:
    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 1        # возраст человека

 tom = Person("Tom")
tom.company = "Microsoft"
print(tom.company)  # Microsoft
```

Работа с несколькими объектами

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

    def introduce(self):
            return f"Привет, меня зовут {self.name} и мне {self.age} лет."

# Создание нескольких экземпляров класса
person1 = Person("Иван", 30)
person2 = Person("Мария", 25)
person3 = Person("Алексей", 35)

# Использование методов
print(person1.introduce())
print(person2.introduce())
print(person3.introduce())
```

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

    def info(self):
        return f"Книга '{self.title}' автора {self.author}."

# Создание нескольких экземпляров класса
book1 = Book("Война и мир", "Лев Толстой")
book2 = Book("Преступление и наказание", "Федор Достоевский")
book3 = Book("1984", "Джордж Оруэлл")

# Использование методов
print(book1.info())
print(book2.info())
print(book3.info())
```

Примеры работы с классами

Пример 1. Управление банковским счетом

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance
    def deposit(self, amount):
        self.balance += amount
        return f"{amount} рублей успешно внесены. Новый баланс: {self.balance} рублей."
    def withdraw(self, amount):
        if amount > self.balance:
            return "Недостаточно средств на счете."
        self.balance -= amount
        return f"{amount} рублей успешно сняты. Новый баланс: {self.balance} рублей."
    def check_balance(self):
        return f"Текущий баланс: {self.balance} рублей."

# Пример использования
account1 = BankAccount("Иванов И.И.", 1000)
print(account1.check_balance())
print(account1.deposit(500))
print(account1.withdraw(2000))
```

Пример 2. Моделирование библиотеки

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

    def check_out(self):
        if self.checked_out:
            return f"Книга '{self.title}' уже взята."
        self.checked_out = True
        return f"Книга '{self.title}' успешно взята."

    def return_book(self):
        if not self.checked_out:
            return f"Книга '{self.title}' уже на месте."
        self.checked_out = False
        return f"Книга '{self.title}' успешно возвращена."

# Пример использования
book1 = Book("Война и мир", "Лев Толстой")
print(book1.check_out())
print(book1.check_out())
print(book1.return_book())
print(book1.return_book())
```

Самостоятельная работа

Задача 1: Работа с банкоматом. Создайте класс ATM, который имеет атрибут balance (баланс банкомата). Реализуйте методы check_balance (проверка баланса) и withdraw (снятие денег с банкомата).

Задача 2: Учет сотрудников. Создайте класс Employee для представления сотрудника с атрибутами name (имя), position (должность) и salary (заработная плата). Реализуйте методы для изменения должности и увеличения зарплаты.

Задача 3: Работа с заказами. Создайте класс Order для представления заказа с атрибутами order_id (идентификатор заказа), items (список товаров) и total (общая сумма заказа). Реализуйте метод для добавления товаров к заказу.

Задача 4: Учет посещенных мест. Создайте класс VisitedPlaces для отслеживания мест, которые посетил пользователь. Реализуйте методы add_place (добавление посещенного места) и list_places (вывод списка посещенных мест).

Задача 5: Работа с автомобилем. Создайте класс Car для представления автомобиля с атрибутами make (марка), model (модель) и year (год выпуска). Реализуйте метод для вывода информации о машине.

Задача 6: Управление почтовым ящиком. Создайте класс Mailbox для представления почтового ящика с атрибутами owner (владелец) и messages (список сообщений). Реализуйте методы для отправки и получения сообщений.