

Практика 10.1 - Конструкторы, переопределение методов

В объектно-ориентированном программировании (ООП), особенно в Python, концепции конструкторов и переопределения методов играют ключевую роль. Давайте рассмотрим каждую из этих концепций подробнее:

Что такое конструктор?

Конструктор в ООП — это специальный метод, который автоматически вызывается при создании объекта класса. В Python, этот метод обычно называется `__init__`.

Основная задача конструктора — инициализировать (то есть присваивать начальные значения) поля объекта или выполнять другие необходимые действия при создании объекта.

В Python, `__init__` это специальный метод, который называется конструктором. Конструктор `__init__` используется для инициализации состояния объекта сразу после его создания. Он имеет ключевую роль в определении классов.

Вот как он обычно используется:

Определение класса: Когда вы создаете новый класс в Python, вы можете использовать метод `__init__`, чтобы задать начальное состояние объекта.

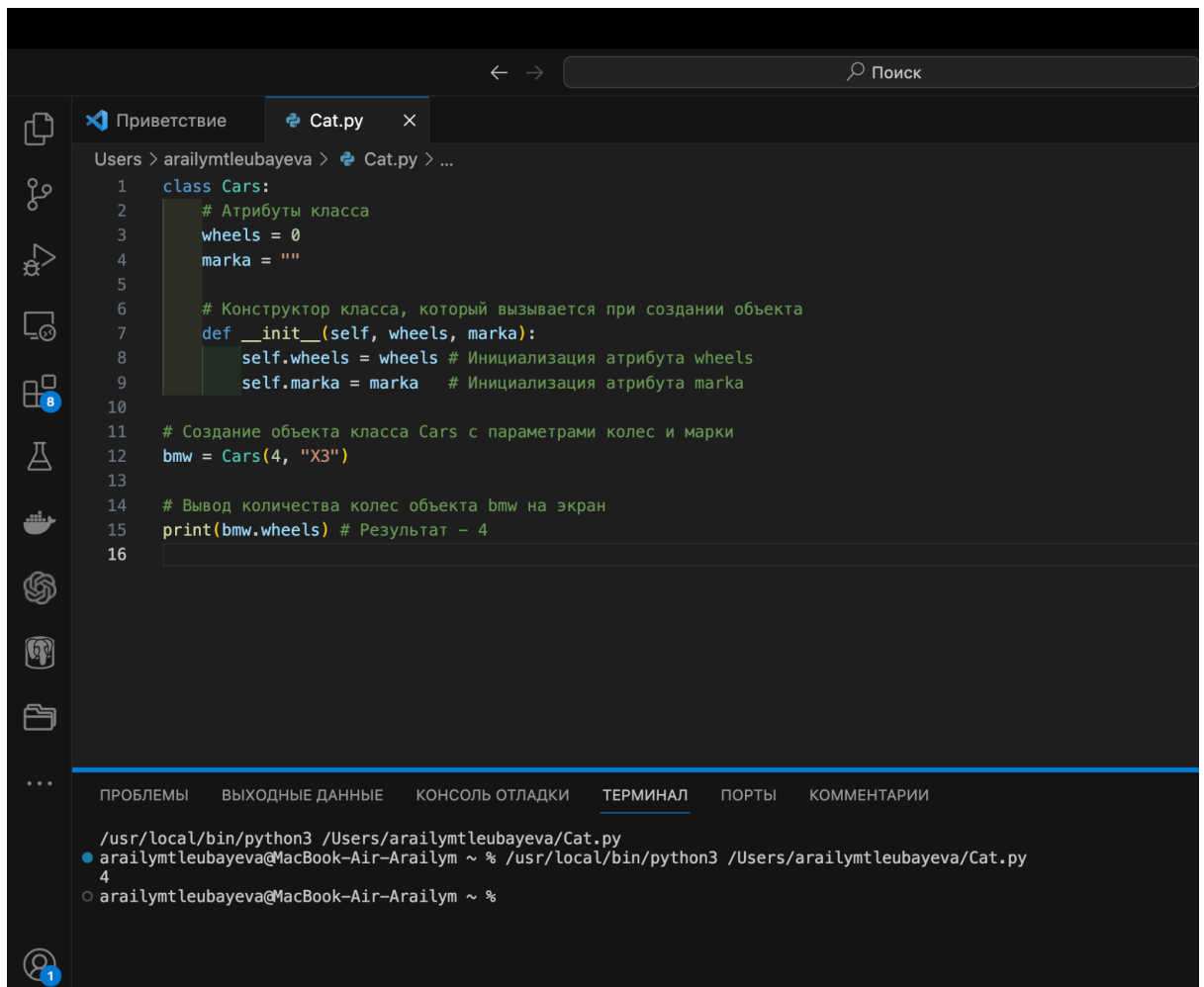
Автоматический вызов: Метод `__init__` автоматически вызывается, когда создается новый экземпляр класса.

Передача параметров: Вы можете передать параметры в `__init__`, чтобы инициализировать атрибуты объекта при его создании.

Конструкторы позволяют задать некие характеристики для объекта сразу же при его создании. К примеру, у вас есть несколько переменных, которые точно должен иметь объект. Вы можете создать конструктор и указать несколько параметров, которые будут переданы при создании объекта.

В одном классе может быть неограниченное количество конструкторов и сам интерпретатор будет понимать к какому конструктору вы обращаетесь. Чтобы создать конструктор необходимо использовать ключевое слово `__init__`.

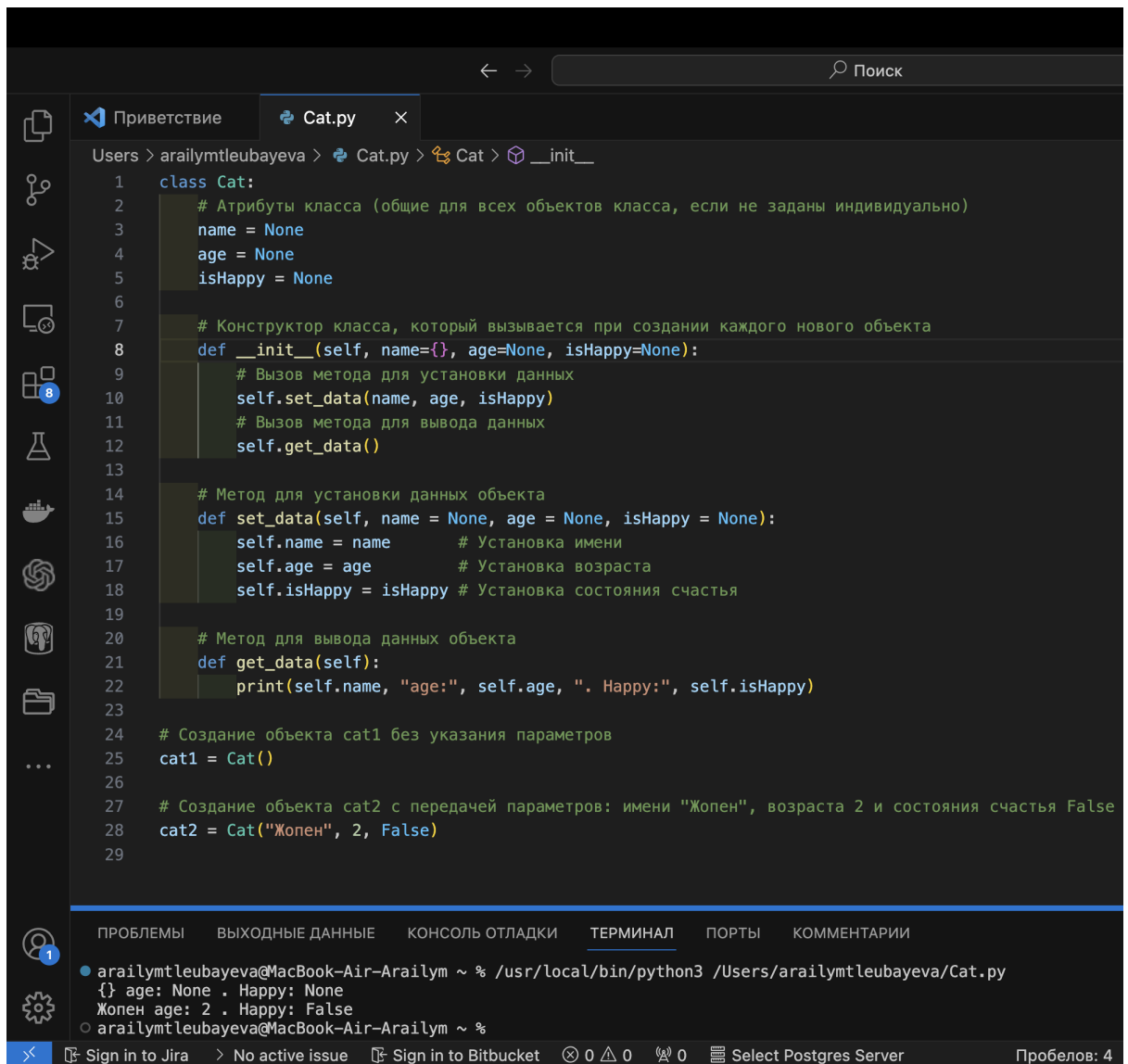
Пример класса с конструктором:



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

```
# Создание объекта класса Person
person = Person("Анна", 30)
```

Пример,Конструкторы



```
1 class Cat:
2     # Атрибуты класса (общие для всех объектов класса, если не заданы индивидуально)
3     name = None
4     age = None
5     isHappy = None
6
7     # Конструктор класса, который вызывается при создании каждого нового объекта
8     def __init__(self, name={}, age=None, isHappy=None):
9         # Вызов метода для установки данных
10        self.set_data(name, age, isHappy)
11        # Вызов метода для вывода данных
12        self.get_data()
13
14    # Метод для установки данных объекта
15    def set_data(self, name = None, age = None, isHappy = None):
16        self.name = name        # Установка имени
17        self.age = age           # Установка возраста
18        self.isHappy = isHappy # Установка состояния счастья
19
20    # Метод для вывода данных объекта
21    def get_data(self):
22        print(self.name, "age:", self.age, ". Happy:", self.isHappy)
23
24    # Создание объекта cat1 без указания параметров
25    cat1 = Cat()
26
27    # Создание объекта cat2 с передачей параметров: имени "Жопен", возраста 2 и состояния счастья False
28    cat2 = Cat("Жопен", 2, False)
29
```

arailymtleubayeva@MacBook-Air-Arilym ~ % /usr/local/bin/python3 /Users/arailymtleubayeva/Cat.py
{ } age: None . Happy: None
Жопен age: 2 . Happy: False
arailymtleubayeva@MacBook-Air-Arilym ~ %

Переопределение методов

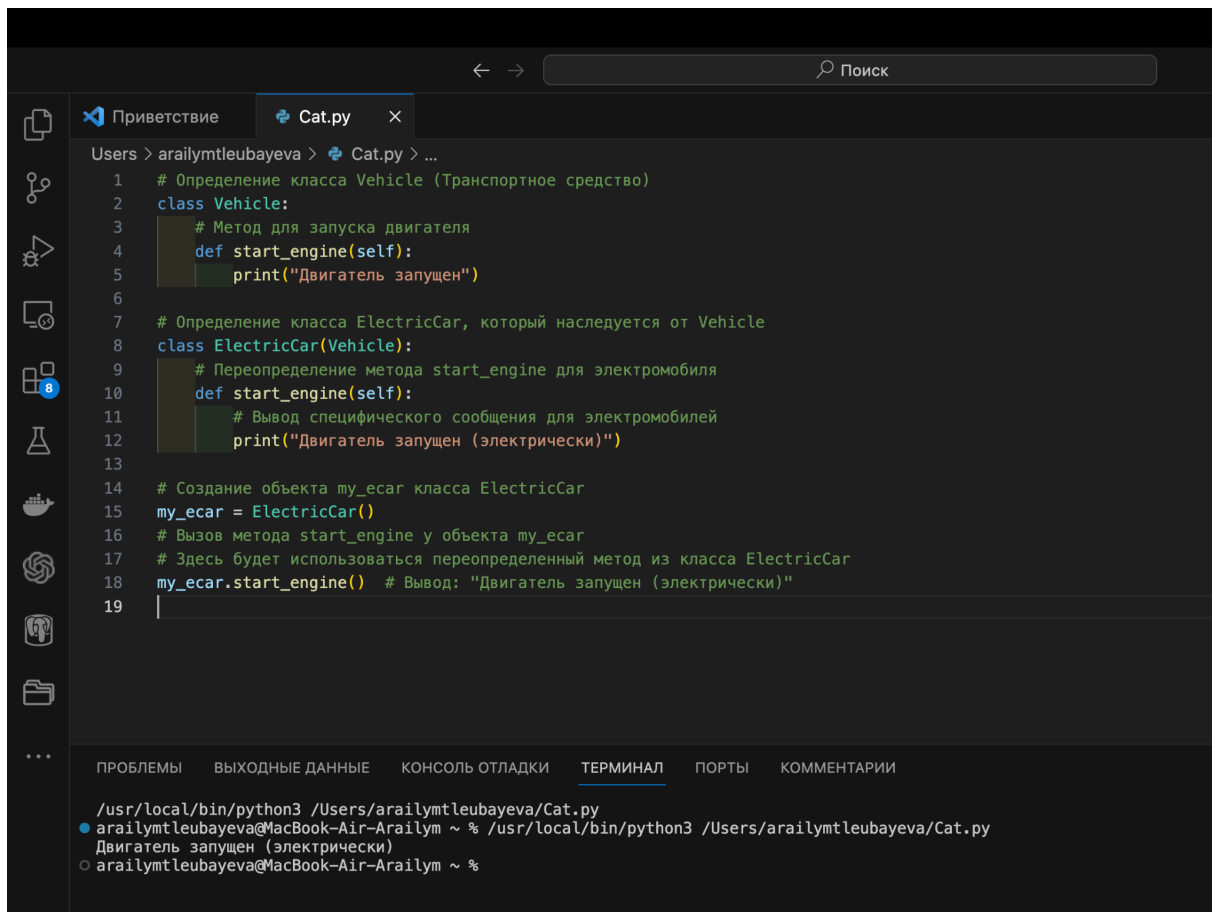
Что такое переопределение методов?

Переопределение методов — это возможность в подклассе изменить поведение метода, унаследованного от родительского класса.

Это позволяет подклассам иметь свою уникальную реализацию метода, который уже определен в их родительском классе.

Пример переопределения метода:

В этом примере, класс `ElectricCar` наследует от `Vehicle` и переопределяет метод `start_engine`. Когда вызывается `start_engine` для объекта `Electric Car`, используется переопределенная версия метода.



```
Users > arailymtleubayeva > Cat.py > ...
1  # Определение класса Vehicle (Транспортное средство)
2  class Vehicle:
3      # Метод для запуска двигателя
4      def start_engine(self):
5          print("Двигатель запущен")
6
7  # Определение класса ElectricCar, который наследуется от Vehicle
8  class ElectricCar(Vehicle):
9      # Переопределение метода start_engine для электромобиля
10     def start_engine(self):
11         # Вывод специфического сообщения для электромобилей
12         print("Двигатель запущен (электрически)")
13
14 # Создание объекта my_ecar класса ElectricCar
15 my_ecar = ElectricCar()
16 # Вызов метода start_engine у объекта my_ecar
17 # Здесь будет использоваться переопределенный метод из класса ElectricCar
18 my_ecar.start_engine() # Вывод: "Двигатель запущен (электрически)"
19 |
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ **ТЕРМИНАЛ** ПОРТЫ КОММЕНТАРИИ

```
/usr/local/bin/python3 /Users/arailymtleubayeva/Cat.py
arailymtleubayeva@MacBook-Air-Arilym ~ % /usr/local/bin/python3 /Users/arailymtleubayeva/Cat.py
Двигатель запущен (электрически)
arailymtleubayeva@MacBook-Air-Arilym ~ %
```

Использование конструкторов и переопределение методов являются основными аспектами ООП, позволяющими создавать гибкие и мощные программные структуры.

Задание 1. Простой конструктор

Есть класс Car:

```
class Car:
    wheels = 4
    model = "Some"
    speed = 123.5

    def get_all(self):
        print("Транспорт ", self.model, " может ехать со скоростью ", self.speed, " на всех ", self.wheels, " колесах!")
```

Создайте конструктор для класса, который будет устанавливать все три значения при создании объекта.

Также на основе класса создайте два объекта.

Самостоятельная работа

Задание 1: Простой конструктор

Цель: Понять основы работы с конструкторами в Python.

- Дополните класс Car конструктором `__init__`, который устанавливает значения для `wheels`, `model` и `speed`.
- Создайте два объекта этого класса с различными характеристиками.

Задание 2: Класс с методами

Цель: Изучить создание методов в классе.

- Создайте класс Book с атрибутами `title` и `author`.
- Добавьте метод `display_info`, который выводит информацию о книге.
- Создайте несколько объектов класса Book и вызовите их методы.

Задание 3: Наследование и базовый функционал

Цель: Изучить концепцию наследования в Python.

- Создайте класс Animal с методом `make_sound`.
- Создайте два подкласса Dog и Cat, наследующие от Animal.
- Переопределите метод `make_sound` в каждом подклассе.

Задание 4: Конструкторы в иерархии наследования

Цель: Понять, как работают конструкторы в иерархии наследования.

- Создайте класс Person с атрибутами `name` и `age` и соответствующим конструктором.
- Создайте класс Employee, наследующий от Person, с дополнительным атрибутом `position`.
- Создайте объекты обоих классов и исследуйте их поведение.

Задание 5: Полиморфизм и абстрактные классы

Цель: Изучить полиморфизм и абстрактные классы в Python.

- Создайте абстрактный класс Shape с абстрактным методом `area`.
- Реализуйте подклассы Rectangle и Circle, переопределяя метод `area`.
- Создайте объекты этих классов и вызовите их методы.

Задание 6: Сложный класс с наследованием и инкапсуляцией

Цель: Понять сложные концепции ООП, такие как наследование и инкапсуляция.

- Создайте класс BankAccount с приватными атрибутами для баланса и номера счета.
- Реализуйте методы для депозита, снятия наличных и проверки баланса.
- Создайте подкласс SavingsAccount, который добавляет функциональность начисления процентов.