ООП

01 Парадигмы программирования

Парадигма программирования

Совокупность принципов, методов и практик, которые определяют стиль организации и написания кода, а также подход программиста к решению задачи

Она определяет, каким образом мы думаем о программировании и как мы структурируем наш код. Каждая парадигма программирования предлагает свои способы описания данных, структур данных, алгоритмов и взаимодействия между компонентами программы

Существуют различные парадигмы программирования, которые определяют разные способы организации кода, работы с данными и решения задач. Рассмотрим некоторые из них

Декларативное программирование

это парадигма программирования, в которой код описывает, что должно быть сделано, а не, как это должно быть сделано

В декларативном программировании разработчик задает желаемый результат, и система сама оптимально реализует его. Это обычно достигается путем описания ограничений, правил и зависимостей

Императивное программирование

это парадигма программирования, в которой код описывает последовательность команд, указывающих компьютеру, как выполнять конкретную задачу

Основной принцип императивного программирования - изменение состояния программы через исполнение команд, которые производят побочные эффекты

Структурное программирование

это парадигма программирования, в которой программа организуется в виде последовательности блоков кода, называемых структурами, которые могут включать условия, циклы и подпрограммы

Главной целью структурного программирования является упрощение кода и повышение его понятности

Процедурное программирование

это парадигма программирования, в которой код организуется вокруг процедур (или функций), которые выполняют определенные действия

Процедуры могут принимать аргументы и возвращать значения. Одним из основных принципов процедурного программирования является декомпозиция программы на более мелкие и понятные части

ООП | объектно-ориентированное программирование

это парадигма программирования, в которой программы организуются вокруг объектов, которые являются экземплярами классов. Классы определяют состояние и поведение объекта

02 Парадигмы программирования

В Python **класс**

это основной механизм объектно-ориентированного программирования

Класс определяет набор атрибутов (переменных) и методов (функций), которые могут быть использованы для создания объектов. Для создания класса в Python используется ключевое слово class, за которым следует имя класса. Имя класса обычно начинается с заглавной буквы, и для именования используется стиль "CamelCase"

Пример

```
class MyClass:
    pass
```

В этом примере мы создали класс с именем MyClass. Ключевое слово pass используется, чтобы указать, что внутри класса нет никаких дополнительных определений для атрибутов и методов. Атрибуты (переменные) класса могут быть объявлены и инициализированы внутри метода __init__, который является конструктором класса

Пример

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

В этом примере мы создали класс Person со свойствами name и age. При создании объекта класса Person, атрибуты name и age могут быть указаны и инициализированы через конструктор

Методы класса

это функции, которые определены внутри класса. Они имеют доступ к атрибутам класса и позволяют работать с ними. Все методы класса должны иметь первый параметр self, который представляет ссылку на объект, через который метод вызывается

Пример

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def calculate_area(self):
        return self.width * self.height
```

В этом примере мы создали класс Rectangle с атрибутами width и height. Метод calculate_area вычисляет площадь прямоугольника на основе значений атрибутов

Для создания объекта класса используется вызов имени класса, за которым следуют аргументы, необходимые для конструктора класса

Пример

```
person = Person("John", 25)
```

В этом примере мы создаем объект класса Person с именем "John" и возрастом 25

Таким образом, классы в Python предоставляют инструменты для описания сущностей, их атрибутов и методов, а объекты класса представляют конкретные экземпляры сущностей, которые можно использовать в программе

03 Принципы ООП

Принципы ООП определяют основные концепции и подходы, которые следует соблюдать при разработке программного кода с использованием ООП. Вот некоторые из основных принципов ООП:

01 **Инкапсуляция** | Encapsulation

Принцип инкапсуляции говорит о том, что данные и методы, работающие с этими данными, должны быть объединены внутри класса. Внешний мир может взаимодействовать с объектом только через публичный интерфейс класса. Это позволяет скрыть детали реализации и обеспечить безопасность данных

Пример

```
class BankAccount:
    def __init__(self):
        self.balance = 0

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds.")

    def get_balance(self):
        return self.balance
```

В этом примере класс BankAccount инкапсулирует данные balance и методы deposit, withdraw и get_balance. Внешний мир может получать доступ только к публичным методам класса

02 **Наследование** | Inheritance

Наследование позволяет создавать иерархию классов, где дочерние классы наследуют свойства и методы от родительских классов. Это позволяет повторно использовать код, расширять функциональность и создавать более специализированные классы

Пример

```
class Shape:
    def __init__(self, color):
        self.color = color

    def get_color(self):
        return self.color

class Circle(Shape):
    def __init__(self, color, radius):
        super().__init__(color)
        self.radius = radius

    def get_area(self):
        return 3.14 * self.radius**2
```

В этом примере класс Circle наследуется от класса Shape. Класс Circle наследует свойство color и метод get_color, определенные в родительском классе Shape

03 **Полиморфизм** | Polymorphism

Принцип полиморфизма позволяет обрабатывать объекты разных классов одним и тем же способом. Это означает, что одно имя метода может иметь разные реализации в разных классах. Полиморфизм способствует гибкости и переиспользованию кода

Пример

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

    def make_sound(self):
        pass

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

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

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

dog = Dog("Buddy")
cat = Cat("Whiskers")

animal_sound(dog)  # Output: Woof!
animal_sound(cat)  # Output: Meow!
```

В этом примере функция animal_sound может принимать объекты разных классов в качестве параметра и вызывать их метод make_sound в полиморфной манере

Одной из основных целей ООП является создание модульного, гибкого и легко поддерживаемого кода. Принципы ООП помогают в достижении этой цели, обеспечивая четкую организацию, управление сложностью и повторное использование кода

Итог

Парадигмы программирования представляют разные подходы к организации и написанию кода. Каждая парадигма предлагает свои принципы и методы, которые определяют стиль работы программиста и организацию программы

Классы в ООП являются основным строительным блоком и позволяют структурировать и организовать код. Они определяют свойства (атрибуты) и поведение (методы) объектов. Классы позволяют создавать объекты, которые являются экземплярами класса, и использовать их для моделирования реальных объектов и взаимодействия между ними

Принципы ООП являются основополагающими концепциями в объектно-ориентированном программировании. Они помогают создавать модульный, гибкий и легко поддерживаемый код