Лекция 2. Понятие наследования. Виды наследования

**Оглавление**

[**Введение** 3](#_Toc188432041)

[**1.** **Понятие наследования** 4](#_Toc188432042)

[**2.** **Виды наследования** 6](#_Toc188432043)

[**Заключение** 10](#_Toc188432044)

[**Список литературы** 11](#_Toc188432045)

**Понятие наследования. Виды наследования**

**Цель:** формирование знаний по одному из базовых элементов объектно-ориентированного программирования – наследованию, его назначению и способам его применения.

**План лекции:**Введение.1. Понятие наследования.  
2. Виды наследования.Заключение.Список литературы.

# **Введение**

Наследование является одним из ключевых принципов объектно-ориентированного программирования (ООП), который значительно упрощает процесс разработки программного обеспечения. Оно позволяет создавать новые классы на основе существующих, повторно используя уже написанный код и расширяя его функциональность. Этот механизм способствует созданию более структурированного, понятного и масштабируемого кода.

Основная идея наследования заключается в том, что мы можем определить базовый (родительский) класс, содержащий общие атрибуты и методы, а затем создавать производные (дочерние) классы, которые наследуют свойства и поведение базового класса. Такой подход не только сокращает дублирование кода, но и делает его более логичным и легко поддерживаемым. Например, если вы разрабатываете программу для управления транспортными средствами, можно создать базовый класс "Транспортное средство" с общими характеристиками, такими как цвет и скорость, а затем унаследовать от него классы "Автомобиль" и "Самолет" с уникальными особенностями.

Наследование также позволяет реализовывать такие концепции, как переопределение методов, полиморфизм и использование абстрактных классов. Благодаря этому можно легко модифицировать или адаптировать поведение дочерних классов без необходимости изменять базовый класс. Кроме того, наследование играет важную роль в соблюдении принципов SOLID, таких как принцип подстановки Барбары Лисков (Liskov Substitution Principle), который обеспечивает корректность взаимодействия объектов в иерархии наследования.

В рамках этой лекции мы рассмотрим, что такое наследование, как оно реализуется в Python, и как правильно использовать его возможности. Мы также разберём преимущества и недостатки наследования. Понимание наследования является важным этапом на пути к освоению объектно-ориентированного программирования и разработке сложных программных систем.

# **Понятие наследования**

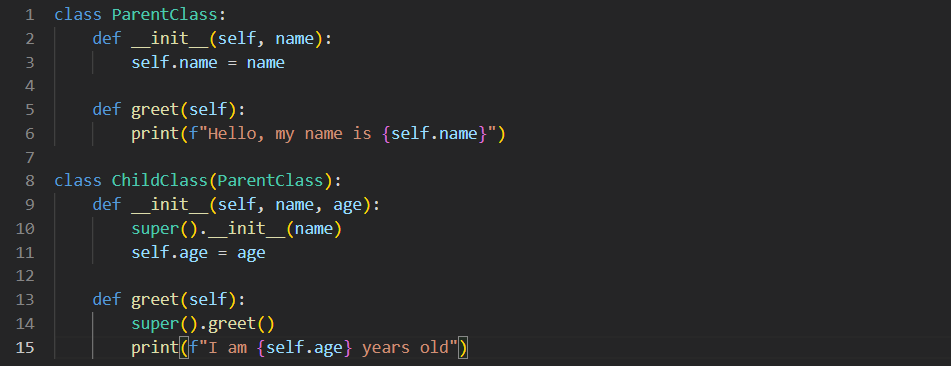
Чтобы разговаривать на одном языке, введем некоторую новую терминологию, а также вспомним материал прошлой лекции.

## **Основные понятия и термины**

* **Класс (Class)**: шаблон для создания объектов (экземпляров), который определяет их свойства и методы. Классы позволяют организовать код в логические блоки и повторно использовать его.
* **Объект (Object)**: экземпляр класса, обладающий определенными свойствами и методами. Объекты позволяют взаимодействовать с данными и функциями, определенными в классе.
* **Наследование (Inheritance)**: механизм, позволяющий одному классу (подклассу) унаследовать свойства и методы другого класса (родительского класса). Наследование способствует повторному использованию кода и упрощает его поддержку.
* **Родительский класс (Parent Class)**: класс, от которого наследуется другой класс. Родительский класс может содержать общие свойства и методы, которые будут унаследованы подклассами.
* **Подкласс (Subclass)**: класс, который наследует свойства и методы родительского класса. Подклассы могут добавлять новые свойства и методы, а также переопределять унаследованные.
* **Переопределение (Overriding)**: процесс замены метода родительского класса в подклассе. Переопределение позволяет изменять поведение унаследованных методов в подклассе.

## **Синтаксис наследования в Python**

В Python наследование реализуется с помощью ключевых слов class и super(). В Python принято именовать классы с использованием PascalCase, однако ошибкой не будет использование других стилей. Вас всего лишь будут осуждать. Давайте рассмотрим базовый синтаксис:



Класс является дочерним, если в скобках указывается его родительский класс

**Разберем, что здесь написано:**

1. **Родительский класс (*встречается терминология «суперкласс» или «базовый»*)**: ParentClass содержит метод \_\_init\_\_ (*конструктор класса, рассмотренный в предыдущей лекции*) для инициализации объекта и метод greet, который выводит приветствие. Метод \_\_init\_\_ используется для задания начальных значений свойств объекта.
2. **Подкласс (*он же «дочерний класс» или «класс-потомок»*)**: ChildClass наследует ParentClass. В его методе \_\_init\_\_ используется super(), чтобы вызвать метод \_\_init\_\_ родительского класса. Это позволяет инициализировать свойства родительского класса в подклассе. Метод greet переопределен, но также вызывает метод greet родительского класса с помощью super(). Это позволяет расширить функциональность метода greet в подклассе.

Работа данного кода будет выглядеть следующим образом:



# **Виды наследования**

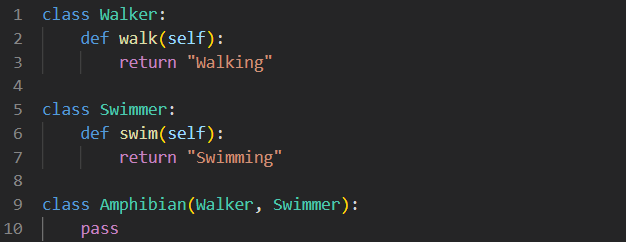
В языке Python существует несколько видов наследования:

* простое или одноуровневое наследование;
* множественное наследование;
* многоуровневое наследование;
* иерархическое наследование;
* гибридное наследование.

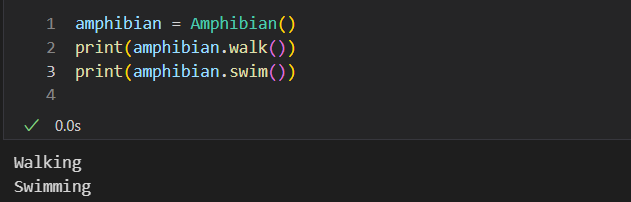
В первом вопросе данной лекции мы рассмотрели базовый случай наследования – простое наследование. Теперь перейдем к более сложным и объемным видам наследования.

## **Множественное наследование**

Множественное наследование позволяет классу наследовать свойства и методы сразу нескольких родительских классов. Это может быть полезно для комбинирования функциональности из разных источников, но требует осторожности, так как **может приводить к сложности и неоднозначности**.

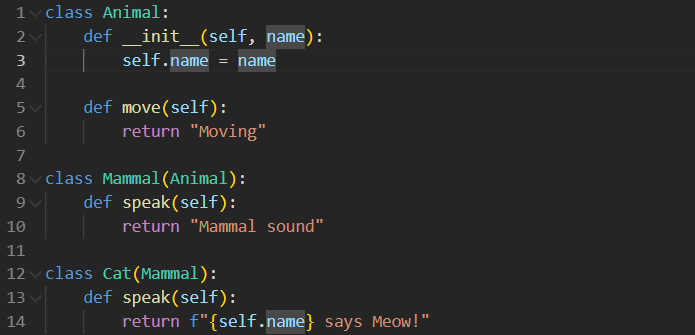


В данном случае мы создали два родительских класса: Walker и Swimmer. Они вполне способны существовать в отдельности друг от друга. А потом мы создали класс Amphibian – класс, наследующий от обоих классов. То есть, в новом классе сразу становятся доступны оба метода из разных родительских классов:

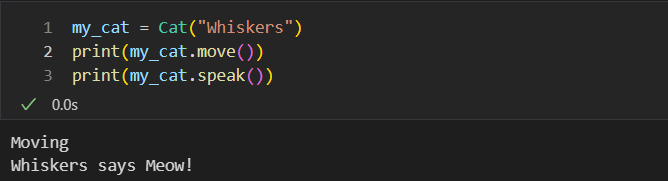


## **Многоуровневое наследование**

Многоуровневое наследование происходит тогда, когда один класс наследует другой класс, который в свою очередь наследует третий класс, и так далее. Таким образом, создается цепочка наследования. По сути, мнемоническим примером такого наследования может быть матрешка.

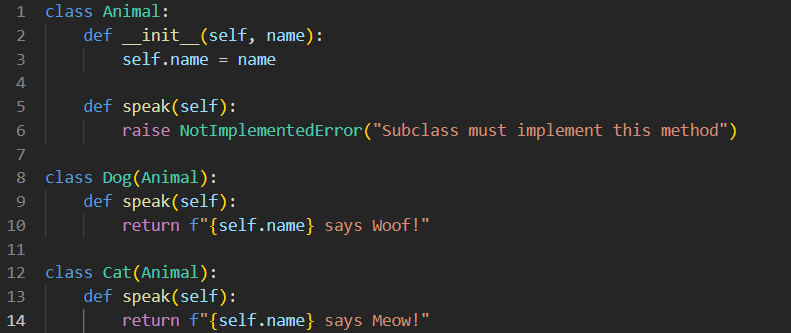


В данном случае мы видим последовательное использование одноуровневого наследования несколько раз – так и получается многоуровневое наследование:

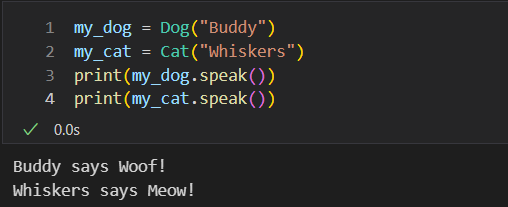


## **Иерархическое наследование**

Иерархическое наследование происходит тогда, когда несколько классов наследуют один и тот же родительский класс. Это позволяет создавать несколько вариантов поведения на основе одного общего базового класса. Грубо говоря, это обратный множественному вид наследования: там было *от многих к одному*, а в иерархическом *от одного ко многим*.

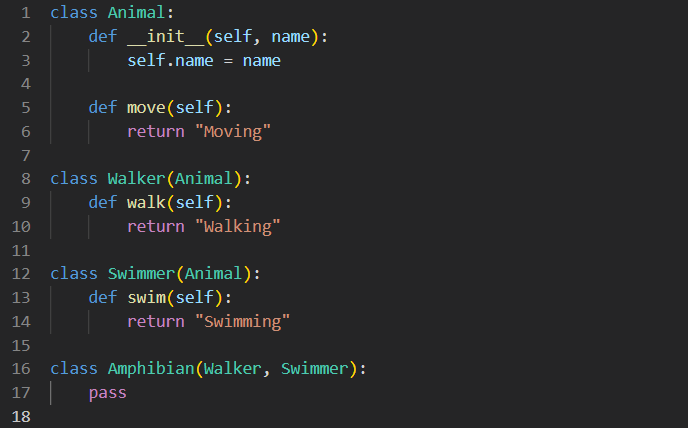


В данном случае мы **переопределяем** метод родительского класса в каждом из дочерних классов, чтобы получить свой уникальный метод в каждом классе, но с одним именем.

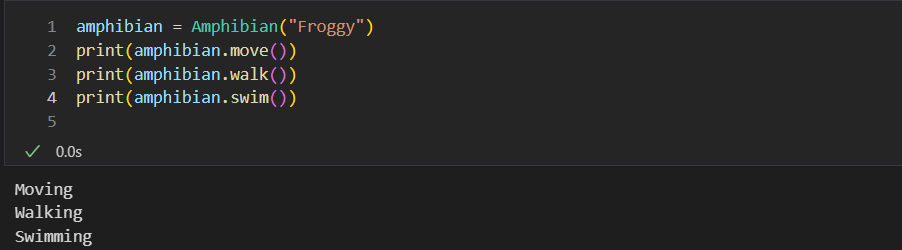


## **Гибридное наследование**

Гибридное наследование представляет собой комбинацию двух и более типов наследования, описанных выше. Это может включать, например, сочетание многоуровневого и множественного наследования.



Результат будет выглядеть следующим образом:



Таким образом, наследование в Python позволяет гибко строить иерархии классов, предоставляя мощные инструменты для повторного использования кода и создания хорошо структурированных программ. Различные виды наследования предоставляют разработчикам возможность выбирать наиболее подходящую структуру для решения конкретных задач, что делает ООП одним из самых эффективных подходов к программированию.

# **Заключение**

Наследование — это мощный инструмент, который значительно упрощает процесс создания и сопровождения программного обеспечения. Освоив этот принцип, вы сможете проектировать более эффективные, структурированные и масштабируемые приложения. Однако, как и любой инструмент, наследование требует правильного использования и понимания его возможностей и ограничений.

В ходе этой лекции мы познакомились с основными концепциями наследования: базовые и дочерние классы, переопределение методов и абстракция. Мы увидели, как наследование помогает сократить дублирование кода и упростить поддержку программных систем. Примеры показали, как можно использовать наследование для моделирования реальных объектов и создания иерархических структур в программировании. Также мы обсудили, как наследование реализовано в Python, включая использование встроенных функций, таких как super(), и применение специальных методов.

В то же время мы обратили внимание на потенциальные недостатки наследования. Например, избыточное использование иерархий классов может привести к усложнению кода, затрудняя его понимание и модификацию.

Наследование — это лишь одна из граней объектно-ориентированного программирования, но её освоение открывает путь к более глубокому пониманию принципов ООП и лучшим практикам проектирования. Применяя наследование, важно всегда помнить о целях и задачах вашей программы, выбирая наиболее подходящие решения для конкретной ситуации.

# **Список литературы**

1. https://ru.hexlet.io/courses/python-oop-basics/lessons/inheritance/theory\_unithttps://en.wikipedia.org/wiki/Global\_interpreter\_lock
2. https://sky.pro/wiki/python/oop-v-python-nasledovanie/
3. https://sky.pro/media/kak-rabotat-s-nasledovaniem-v-python/
4. https://metanit.com/python/tutorial/7.3.php
5. https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-18-osnovy-oop-inkapsulyaciya-i-nasledovanie-2023-04-17