

Наследование, инкапсуляция и полиморфизм

Тимур Анвардинов
Инженер по контролю качества, smotreshka.tv

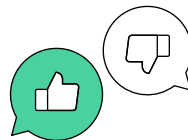


Проверка связи





Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



Поставьте в чат:

-  если меня видно и слышно
-  если нет

Рекомендации

→ Если смотрите с компьютера

- Используйте браузеры **Google Chrome** или **Microsoft Edge**
- Если есть проблемы с изображением или звуком, обновите страницу — **F5**

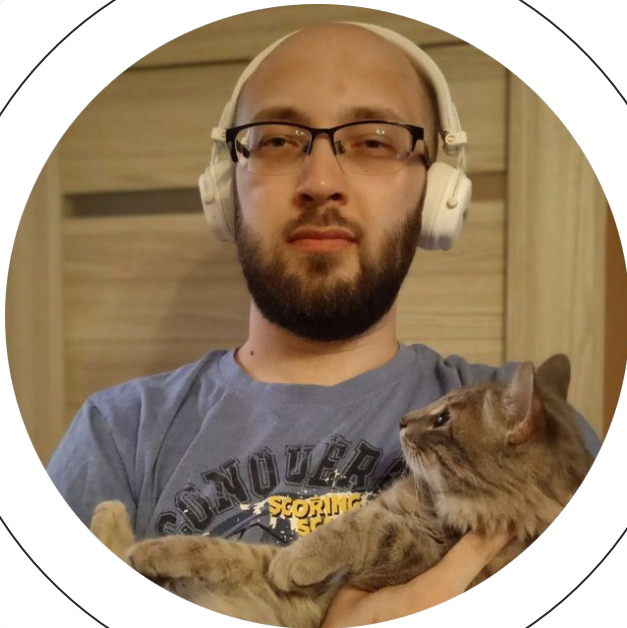
→ Если смотрите с мобильного телефона или планшета

- Перейдите с мобильного интернет-соединения на **Wi-Fi**
- Если есть проблемы с изображением или звуком, перезапустите приложение **МТС Линк** на телефоне
- Предварительно проверьте, подходит ли ваше устройство для подключения к вебинару, по [ссылке](#)

Тимур Анвартдинов

О спикере:

- Организовал с нуля автоматизацию тестирования в своей компании
- Отвечаю за процесс обучения новых сотрудников
- Вырос из студентов Нетологии в преподаватели



Сегодня вы узнаете

- 1 Что такое инкапсуляция
- 2 Что такое модификация доступа
- 3 Что такое наследование и множественное наследование
- 4 Что такое полиморфизм



План занятия

- 1 Инкапсуляция
- 2 Наследование
- 3 Полиморфизм
- 4 Итоги и ваши вопросы



Инкапсуляция



1

Инкапсуляция

— это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации.

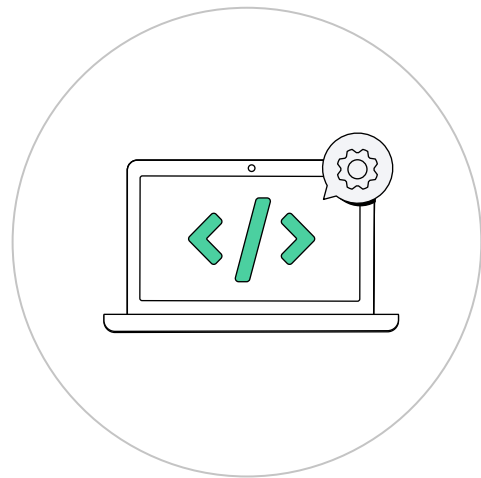
Скрываем всё

Стив Джобс был одним из первых, кто применил прием инкапсуляции к ПК, перекрыв доступ к внутреннему устройству компьютера обычному пользователю.



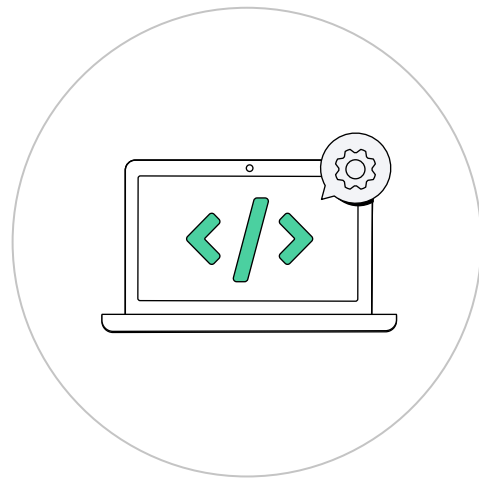
Зачем нужна инкапсуляция?

- Защита данных. Предотвращает случайное изменение важных данных извне класса
- Упрощение использования. Пользователь класса работает только с публичными методами, не зная, как устроена внутренняя логика
- Гибкость и поддержка. Можно изменять внутреннюю реализацию класса, не затрагивая внешний интерфейс



Как инкапсуляция реализована в Python?

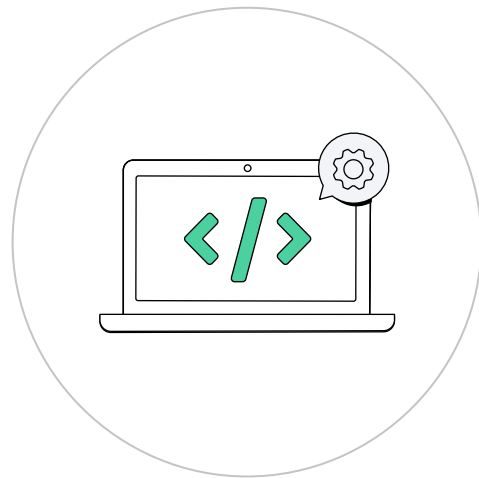
- Соглашения об именах (модификаторы доступа)
- Использование геттеров и сеттеров



Модификаторы доступа



Модификаторы доступа в Python используются для модификации области видимости переменных по умолчанию.



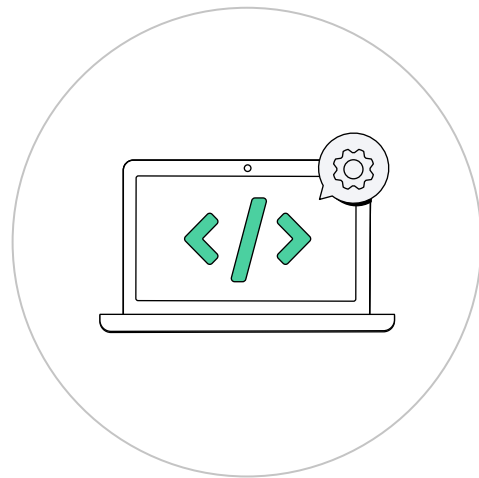
Модификаторы доступа



Модификаторы доступа в Python используются для модификации области видимости переменных по умолчанию.

Есть три типа модификаторов доступов в Python ООП:

- **public** - доступ к переменным с модификаторами публичного доступа открыт из любой точки вне класса



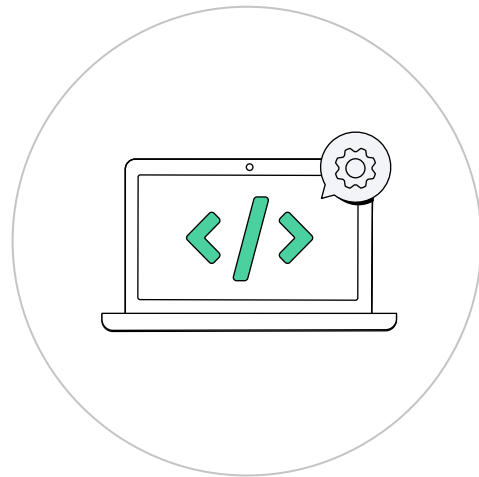
Модификаторы доступа



Модификаторы доступа в Python используются для модификации области видимости переменных по умолчанию.

Есть три типа модификаторов доступов в Python ООП:

- **public** - доступ к переменным с модификаторами публичного доступа открыт из любой точки вне класса
- **private** - доступ к приватным переменным открыт только внутри самого класса



Модификаторы доступа

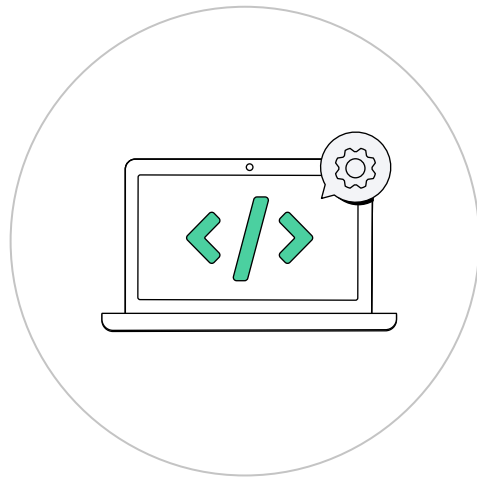


Модификаторы доступа в Python используются для модификации области видимости переменных по умолчанию.

Есть три типа модификаторов доступов в Python ООП:

- **public** - доступ к переменным с модификаторами публичного доступа открыт из любой точки вне класса
- **__private** - доступ к приватным переменным открыт только внутри самого класса
- **__protected** - доступ открыт только внутри класса и дочерних классов

Но работают они только на словах!

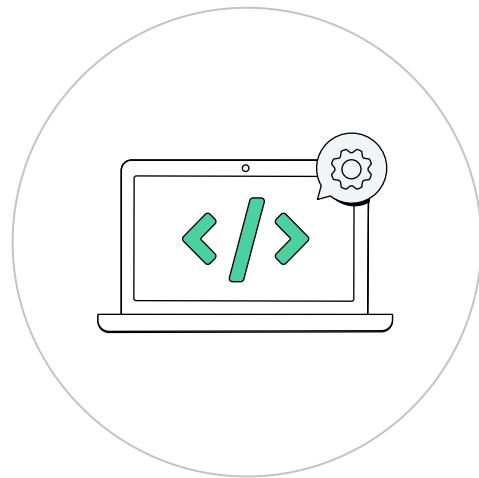



Геттеры и Сеттеры



Геттеры и сеттеры позволяют контролировать доступ к атрибутам класса. В Python это реализуется через декоратор `@property`.

- **`@property`** - для безопасного доступа к атрибуту
- **`@имя_атрибута.setter`** - для безопасного изменения атрибута





**Рассмотрим на
примере**

Наследование



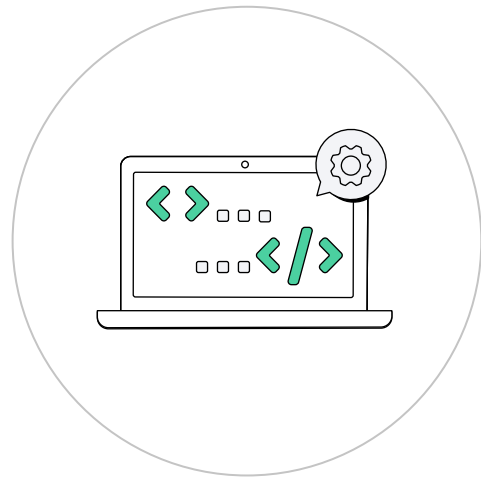
2

Наследование

— это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.

Зачем нужно наследование?

- Повторное использование кода. Подкласс может использовать атрибуты и методы родительского класса, не дублируя их
- Расширение функциональности. Подкласс может добавлять новые методы или переопределять существующие.
- Создание иерархий. Наследование помогает организовать классы в логические иерархии.



Как работает наследование?

1

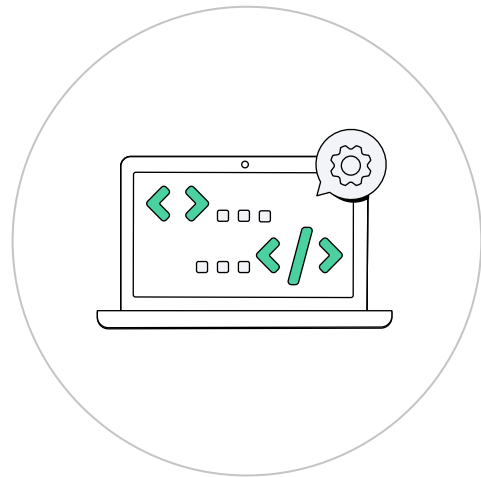
Создание базового класса:

- Базовый класс содержит общие атрибуты и методы, которые могут быть унаследованы.

2

Создание подкласса:

- Подкласс наследует все атрибуты и методы базового класса.
- Подкласс может добавлять новые атрибуты и методы, переопределять методы базового класса, использовать методы базового класса через **super()**



Наследование



`class` Primate



`class` Man

```
class Primate:  
    def eat(self, food):  
        ...  
    def run(self, time):  
        ...
```

```
class Man(Primate):  
    name = ...  
    last_name = ...  
  
    def build(self, something):  
        ...
```

Множественное наследование



`class Spider`



`class Man`



`class SpiderMan`

Что наследовать?

MRO (Method Resolution Order) — порядок, в котором Python ищет метод в иерархии классов.


```
class Primate:
    ...

class Man(Primate):
    ...

class Spider:
    ...

class SpiderMan(Spider, Man):
    ...

SpiderMan.mro()
```

**Рассмотрим на
примере**

Полиморфизм



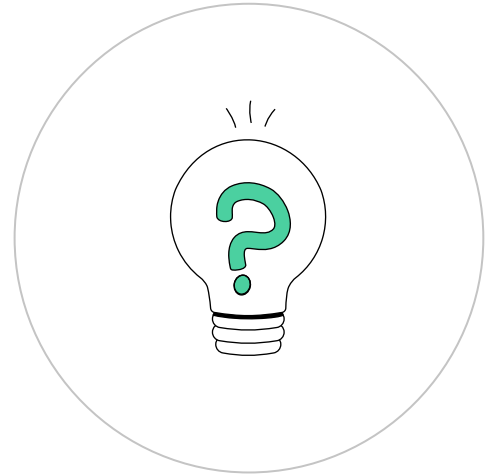
3

Полиморфизм

— это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Зачем нужен полиморфизм?

- Универсальность: Один интерфейс может использоваться для работы с объектами разных типов.
- Гибкость: Код становится более гибким и переиспользуемым.



Какие виды полиморфизма бывают?

Python поддерживает несколько видов полиморфизма:

1

Ад-хок полиморфизм:

- Разные операции выполняются с одним и тем же оператором в зависимости от типов операндов.
- Пример: оператор `+` может складывать числа, объединять строки или списки.

2

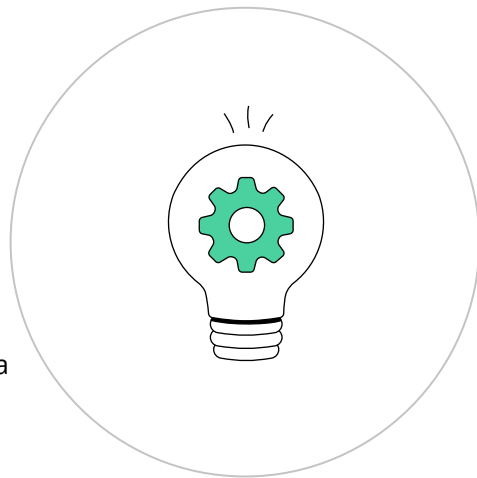
Параметрический полиморфизм:

- Функция или метод могут работать с любым типом данных, если они поддерживают требуемый интерфейс.
- Пример: функция `len()` работает со строками, списками, кортежами и другими объектами, имеющими длину.

3

Полиморфизм подтипов:

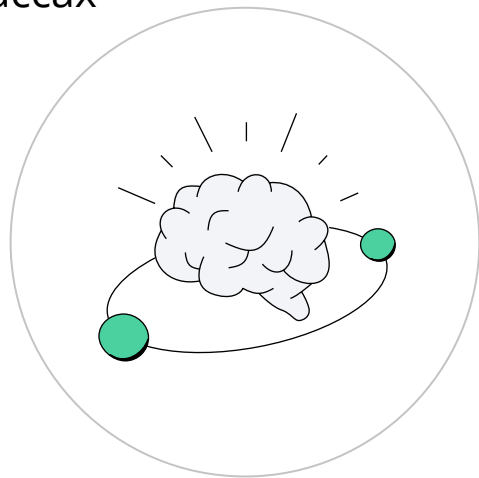
- Методы базового класса могут быть переопределены в подклассах. При вызове метода через ссылку на базовый класс будет использоваться реализация из подкласса.



Как работает полиморфизм в Python?

Какие средства в Python помогают реализовать полиморфизм

- Магические методы
- Переопределение родительских методов в дочерних классах



Магические методы



Инициализация и Конструирование:

- `__new__(cls, other)`
- `__init__(self, other)`
- `__del__(self)`



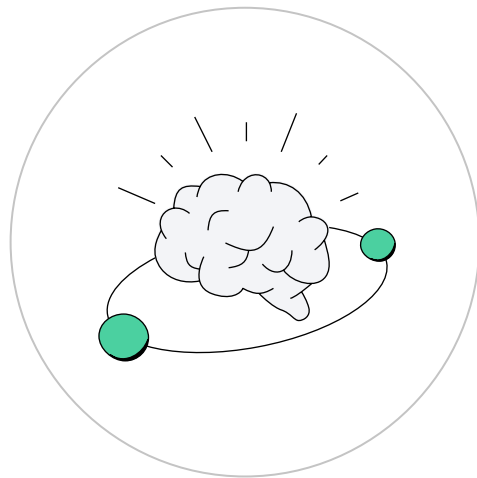
Методы сравнения:

- | | |
|-------------------------------------|----------------------------------|
| • <code>__eq__(self, other)</code> | <code>__gt__(self, other)</code> |
| • <code>__ne__(self, other)</code> | <code>__le__(self, other)</code> |
| • <code>__del__(self, other)</code> | <code>__ge__(self, other)</code> |



Числовые методы:

- | | |
|-------------------------------------|------------------------------------|
| • <code>__add__(self, other)</code> | <code>__abs__(self)</code> |
| • <code>__sub__(self, other)</code> | <code>__rdiv__(self, other)</code> |
| • <code>__mul__(self, other)</code> | <code>__rand__(self, other)</code> |



Магические методы

→ Методы преобразования:

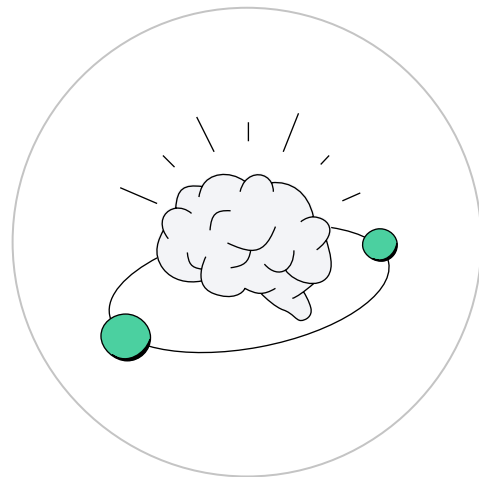
- `__int__(self)`
- `__float__(self)`

→ Методы представления:

- `__str__(self)`
- `__repr__(self)`
- `__hash__(self)`

→ Методы контейнеров:

- | | |
|---|---------------------------------------|
| • <code>__len__(self, other)</code> | <code>__reversed__(self)</code> |
| • <code>__getitem__(self, other)</code> | <code>__setitem__(self, other)</code> |
| • <code>__iter__(self, other)</code> | <code>__next__(self, other)</code> |




Магических методов очень много. Рекомендую прочитать [статью](#)

Переопределение методов в дочерних классах

Заставляем методы работать по-разному в зависимости от наличия параметров или исходя из того, из какого класса мы их вызываем.

```
class Primate:
    def eat(self, food):
        print(food)

class Man(Primate):
    def eat(self, food,
cooked=False):
        if cooked:
            print('cooked', food)
        else:
            print(food)
```



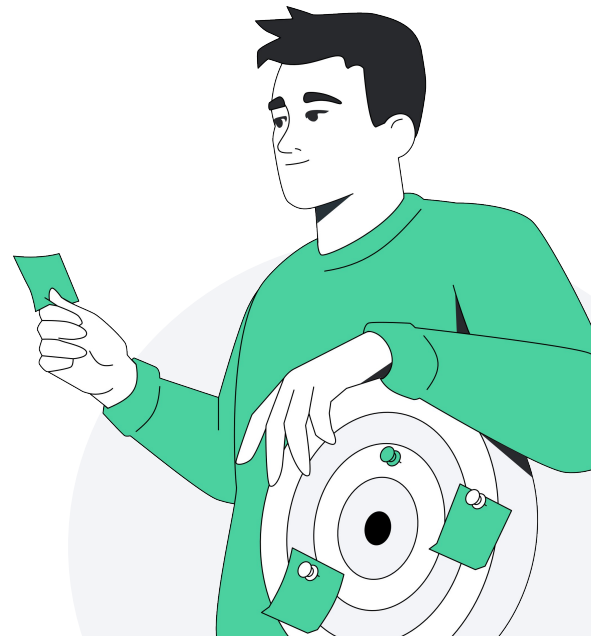
**Рассмотрим на
примере**

Итоги и ваши вопросы



Сегодня мы узнали

- Инкапсуляция. Как она реализована в Python, как и зачем ее применять.
- Наследование. Как переиспользовать свои классы, чтобы не дублировать код.
- Полиморфизм. Как работать с разными объектами через один интерфейс.





Ваши вопросы?

Наследование, инкапсуляция и полиморфизм

Тимур Анвардинов
Инженер по контролю качества, smotreshka.tv

