

ACCEМБЛЕР	C#	JAVA	WEB	PYTHON	C	C++	SQL	MONGODB	GO	VB.NET	SWIFT	KOTLIN	DART	PHP	RUST	F#
НАСТРОЙКИ																

- ▶ 📁 Глава 1. Введение в Python
- ▶ 📁 Глава 2. Основы Python
- ▼ 📁 Глава 3. Объектно-ориентированное программирование
 - ❑ Классы и объекты
 - ❑ Инкапсуляция, атрибуты и свойства
 - ❑ **Наследование**
 - ❑ Переопределение функционала базового класса
 - ❑ Атрибуты классов и статические методы
 - ❑ Класс object. Строковое представление объекта
- ▶ 📁 Глава 4. Обработка ошибок и исключений
- ▶ 📁 Глава 5. Списки, кортежи и словари
- ▶ 📁 Глава 6. Модули
- ▶ 📁 Глава 7. Строки
- ▶ 📁 Глава 8. Pattern matching
- ▶ 📁 Глава 9. Работа с файлами
- ▶ 📁 Глава 10. Работа с датами и временем

Наследование

Последнее обновление: 28.01.2022



Наследование позволяет создавать новый класс на основе уже существующего класса. Наряду с инкапсуляцией наследование является одним из краеугольных камней объектно-ориентированного программирования.

Ключевыми понятиями наследования являются **подкласс** и **суперкласс**. **Подкласс** наследует от суперкласса все публичные атрибуты и методы. Суперкласс еще называется базовым (base class) или родительским (parent class), а подкласс - производным (derived class) или дочерним (child class).

Синтаксис для наследования классов выглядит следующим образом:

```
1 class подкласс (суперкласс):
2     методы_подкласса
```

Например, у нас есть класс Person, который представляет человека:

```
1 class Person:
2
3     def __init__(self, name):
4         self.__name = name # имя человека
5
6     @property
7     def name(self):
8         return self.__name
9
10    def display_info(self):
11        print(f'Name: {self.__name}')
```

Предположим, нам необходим класс работника, который работает на некотором предприятии. Мы могли бы создать с нуля новый класс, к примеру, класс Employee:

```
1 class Employee:
2
3     def __init__(self, name):
4         self.__name = name # имя работника
5
6     @property
7     def name(self):
8         return self.__name
9
10    def display_info(self):
11        print(f'Name: {self.__name} ')
12
13    def work(self):
14        print(f'{self.name} works')
```

Однако класс Employee может иметь те же атрибуты и методы, что и класс Person, так как работник - это человек. Так, в выше в классе Employee только добавляется метод `work`, весь остальной код повторяет функционал класса Person. Но чтобы не дублировать функционал одного класса в другом, в данном случае лучше применить наследование.

Итак, унаследуем класс Employee от класса Person:

```
1 class Person:
2
3     def __init__(self, name):
4         self.__name = name # имя человека
5
6     @property
7     def name(self):
8         return self.__name
9
10    def display_info(self):
11        print(f'Name: {self.__name} ')
12
13
14    class Employee(Person):
15
16        def work(self):
17            print(f'{self.name} works')
18
19
20    tom = Employee("Tom")
21    print(tom.name) # Tom
22    tom.display_info() # Name: Tom
23    tom.work() # Tom works
```

Класс Employee полностью перенимает функционал класса Person, лишь добавляя метод `work()`. Соответственно при создании объекта Employee мы можем использовать унаследованный от Person конструктор:

```
1 tom = Employee("Tom")
```

И также можно обращаться к унаследованным атрибутам/свойствам и методам:

```
1 print(tom.name) # Tom
2 tom.display_info() # Name: Tom
```

Однако, стоит обратить внимание, что для Employee НЕ доступны закрытые атрибуты типа `__name`. Например, мы НЕ можем в методе `work` обратиться к приватному атрибуту `self.__name`:

```
1 def work(self):
2     print(f'{self.__name} works') # ! Ошибка
```

Множественное наследование

Одной из отличительных особенностей языка Python является поддержка множественного наследования, то есть один класс можно унаследовать от нескольких классов:

```
1 # класс работника
2 class Employee:
3     def work(self):
4         print("Employee works")
5
6
7 # класс студента
8 class Student:
9     def study(self):
10        print("Student studies")
11
12
13 class WorkingStudent(Employee, Student): # Наследование от классов Employee и Student
14     pass
15
```

```
16 # класс работающего студента
17 tom = WorkingStudent()
18 tom.work() # Employee works
19 tom.study() # Student studies
```

Здесь определен класс `Employee`, который представляет сотрудника фирмы, и класс `Student`, который представляет учащегося студента. Класс `WorkingStudent`, который представляет работающего студента, не определяет никакого функционала, поэтому в нем определен оператор `pass`. Класс `WorkingStudent` просто наследует функционал от двух классов `Employee` и `Student`. Соответственно у объекта этого класса мы можем вызвать методы обоих классов.

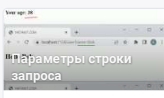
При этом наследуемые классы могут более сложными по функциональности, например:

```
1 class Employee:
2
3     def __init__(self, name):
4         self.__name = name
5
6     @property
7     def name(self):
8         return self.__name
9
10    def work(self):
11        print(f"{self.name} works")
12
13
14    class Student:
15
16        def __init__(self, name):
17            self.__name = name
18
19        @property
20        def name(self):
21            return self.__name
22
23        def study(self):
24            print(f"{self.name} studies")
25
26
27    class WorkingStudent(Employee, Student):
28        pass
29
30
31    tom = WorkingStudent("Tom")
32    tom.work() # Tom works
33    tom.study() # Tom studies
```

[Назад](#) [Содержание](#) [Вперед](#)



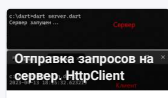
ТАКОЖЕ НА METANIT.COM



Параметры строки запроса

5 месяцев назад · 1 комме...


Параметры строки запроса query string в приложении Blazor на C#.



Отправка запросов на сервер. HttpClient

6 месяцев назад · 1 комме...


Отправка запросов на сервер HttpServer с помощью класса ...



ListView

2 месяца назад · 1 коммент...

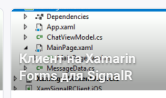
ListView в JavaFX, создание списков, получение выбранных в ...



Встроенные компоненты ввода

5 месяцев назад · 1 комме...

Встроенные компоненты ввода Blazor из пространства имен ...



Клиент на Xamarin Forms для SignalR

22 дня назад · 1 комментарий...

Клиентское приложение на Xamarin Forms для SignalR в ASP.NET Core, ...



Ассинхронное обновление

3 месяца назад · 1 коммент...

Ассинхронное обновление UI в Blazor, использование ...