

АСЕМБЛЕР	C#	JAVA	WEB	PYTHON	C	C++	SQL	MONGOVB	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

## Объектно-ориентированное программирование

### Классы и объекты

Последнее обновление: 27.01.2022



Python имеет множество встроенных типов, например, int, str и так далее, которые мы можем использовать в программе. Но также Python позволяет определять собственные типы с помощью **классов**. Класс представляет некоторую сущность. Конкретным воплощением класса является объект.

Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. Человек может выполнять некоторые действия - ходить, бегать, думать и т.д. То есть это представление, которое включает набор характеристик и действий, можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек будет представлять объект этого класса.

Класс определяется с помощью ключевого слова **class**:

```
1 class название_класса:
2     атрибуты_класса
3     методы_класса
```

Внутри класса определяются его атрибуты, которые хранят различные характеристики класса, и методы - функции класса.

Создадим простейший класс:

```
1 class Person:
2     pass
```

В данном случае определен класс Person, который условно представляет человека. В данном случае в классе не определяется никаких методов или атрибутов. Однако поскольку в нем должно быть что-то определено, то в качестве заменителя функционала класса применяется оператор **pass**. Этот оператор применяется, когда синтаксически необходимо определить некоторый код, однако мы не хотим его, и вместо конкретного кода вставляем оператор pass.

После создания класса можно определить объекты этого класса. Например:

```
1 class Person:
2     pass
3
4 tom = Person()      # определение объекта tom
5 bob = Person()      # определение объекта bob
```

После определения класса Person создаются два объекта класса Person - tom и bob. Для создания объекта применяется специальная функция - **конструктор**, которая называется по имени класса и которая возвращает объект класса. То есть в данном случае вызов Person() представляет вызов конструктора. Каждый класс по умолчанию имеет конструктор без параметров:

```
1 tom = Person()      # Person() - вызов конструктора, который возвращает объект класса Person
```

### Методы классов

Методы класса фактически представляют функции, которые определены внутри класса и которые определяют его поведение. Например, определим класс Person с одним методом:

```
1 class Person:      # определение класса Person
2     def say_hello(self):
3         print("Hello")
4
5 tom = Person()
6 tom.say_hello()    # Hello
```

Здесь определен метод say\_hello(), который условно выполняет приветствие - выводит строку на консоль. При определении методов любого класса следует учитывать, что все они должны принимать в качестве первого параметра ссылку на текущий объект, который согласно условностям называется **self**. Через эту ссылку внутри класса мы можем обратиться к функциональности текущего объекта. Но при самом вызове метода этот параметр не учитывается.

Используя имя объекта, мы можем обратиться к его методам. Для обращения к методам применяется нотация точки - после имени объекта ставится точка и после нее идет вызов метода:

```
1 объект.метод([параметры метода])
```

Например, обращение к методу say\_hello() для вывода приветствия на консоль:

```
1 tom.say_hello()    # Hello
```

В итоге данная программа выведет на консоль строку "Hello".

Если метод должен принимать другие параметры, то они определяются после параметра self, и при вызове подобного метода для них необходимо передать значения:

```
1 class Person:      # определение класса Person
2     def say(self, message):      # метод
3         print(message)
4
5
6 tom = Person()
7 tom.say("Hello METANIT.COM")    # Hello METANIT.COM
```

Здесь определен метод say(). Он принимает два параметра: self и message. И для второго параметра - message при вызове метода необходимо передать значение.

### self

Через ключевое слово **self** можно обращаться внутри класса к функциональности текущего объекта:

```
1 self.атрибут      # обращение к атрибуту
2 self.метод        # обращение к методу
```

Например, определим два метода в классе Person:

```
1 class Person:
2
3     def say(self, message):
4         print(message)
5
6     def say_hello(self):
7         self.say("Hello work")    # обращаемся к выше определенному методу say
8
9
10 tom = Person()
11 tom.say_hello()    # Hello work
```

Здесь в одном методе - say\_hello() вызывается другой метод - say():

```
1 self.say("Hello work")
```

Поскольку метод `say()` принимает кроме `self` еще параметры (параметр `message`), то при вызове метода для этого параметра передается значение.

Причем при вызове метода объекта нам обязательно необходимо использовать слово **self**, если мы его не используем:

```
1 def say_hello(self):
2     say("Hello work") # ! Ошибка
```

То мы столкнемся с ошибкой

## Конструкторы

Для создания объекта класса используется конструктор. Так, выше когда мы создавали объекты класса `Person`, мы использовали конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы:

```
1 tom = Person()
```

Однако мы можем явным образом определить в классах конструктор с помощью специального метода, который называется `__init__()` (по два прочерка с каждой стороны). К примеру, изменим класс `Person`, добавив в него конструктор:

```
1 class Person:
2     # конструктор
3     def __init__(self):
4         print("Создание объекта Person")
5
6     def say_hello(self):
7         print("Hello")
8
9
10 tom = Person() # Создание объекта Person
11 tom.say_hello() # Hello
```

Итак, здесь в коде класса `Person` определен конструктор и метод `say_hello()`. В качестве первого параметра конструктор, как и методы, также принимает ссылку на текущий объект - `self`. Обычно конструкторы применяются для определения действий, которые должны производиться при создании объекта.

Теперь при создании объекта:

```
1 tom = Person()
```

будет производится вызов конструктора `__init__()` из класса `Person`, который выведет на консоль строку "Создание объекта `Person`".

## Атрибуты объекта

Атрибуты хранят состояние объекта. Для определения и установки атрибутов внутри класса можно применять слово **self**. Например, определим следующий класс `Person`:

```
1 class Person:
2
3     def __init__(self, name):
4         self.name = name # имя человека
5         self.age = 1 # возраст человека
6
7
8 tom = Person("Tom")
9
10 # обращение к атрибутам
11 # получение значений
12 print(tom.name) # Tom
13 print(tom.age) # 1
14 # изменение значения
15 tom.age = 37
16 print(tom.age) # 37
```

Теперь конструктор класса `Person` принимает еще один параметр - `name`. Через этот параметр в конструктор будет передаваться имя создаваемого человека.

Внутри конструктора устанавливаются два атрибута - `name` и `age` (условно имя и возраст человека):

```
1 def __init__(self, name):
2     self.name = name
3     self.age = 1
```

Атрибуту `self.name` присваивается значение переменной `name`. Атрибут `age` получает значение `1`.

Если мы определили в классе конструктор `__init__`, мы уже не сможем вызвать конструктор по умолчанию. Теперь нам надо вызывать наш явным образом определенный конструктор `__init__`, в который необходимо передать значение для параметра `name`:

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

Далее по имени объекта мы можем обращаться к атрибутам объекта - получать и изменять их значения:

```
1 print(tom.name) # получение значения атрибута name
2 tom.age = 37 # изменение значения атрибута age
```

В принципе нам необязательно определять атрибуты внутри класса - Python позволяет сделать это динамически вне кода:

```
1 class Person:
2
3     def __init__(self, name):
4         self.name = name # имя человека
5         self.age = 1 # возраст человека
6
7
8 tom = Person("Tom")
9
10 tom.company = "Microsoft"
11 print(tom.company) # Microsoft
```

Здесь динамически устанавливается атрибут `company`, который хранит место работы человека. И после установки мы также можем получить его значение. В то же время подобное определение чревато ошибками. Например, если мы попытаемся обратиться к атрибуту до его определения, то программа сгенерирует ошибку:

```
1 tom = Person("Tom")
2 print(tom.company) # ! Ошибка - AttributeError: Person object has no attribute company
```

Для обращения к атрибутам объекта внутри класса в его методах также применяется слово **self**:

```
1 class Person:
2
3     def __init__(self, name):
4         self.name = name # имя человека
5         self.age = 1 # возраст человека
6
7     def display_info(self):
8         print(f"Name: {self.name} Age: {self.age}")
9
10
11 tom = Person("Tom")
12 tom.display_info() # Name: Tom Age: 1
```

Здесь определяется метод `display_info()`, который выводит информацию на консоль. И для обращения в методе к атрибутам объекта применяется слово `self`: `self.name` и `self.age`

## Создание объектов

Выше создавался один объект. Но подобным образом можно создавать и другие объекты класса:

```
1 class Person:
2
3     def __init__(self, name):
4
5         self.name = name # имя человека
6         self.age = 1 # возраст человека
7
8     def display_info(self):
9         print(f"Name: {self.name} Age: {self.age}")
10
11 tom = Person("Tom")
12 tom.age = 37
13 tom.display_info() # Name: Tom Age: 37
14
15 bob = Person("Bob")
16 bob.age = 41
17 bob.display_info() # Name: Bob Age: 41
```

Здесь создаются два объекта класса Person: tom и bob. Они соответствуют определению класса Person, имеют одинаковый набор атрибутов и методов, однако их состояние будет отличаться.

При выполнении программы Python динамически будет определять **self** - он представляет объект, у которого вызывается метод. Например, в строке:

```
1 tom.display_info() # Name: Tom Age: 37
```

Это будет объект tom

А при вызове

```
1 bob.display_info()
```

Это будет объект bob

В итоге мы получим следующий консольный вывод:

```
Name: Tom Age: 37
Name: Bob Age: 41
```

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



ТАКОЖЕ НА METANIT.COM

22 Dependencies

App.xaml

ChatRoomModels.cs

MainPage.xaml

Клиентское приложение на Xamarin Forms для SignalR

22 дня назад · 1 коммент...

Assembler MASM. Установка и начало работы, Visual Studio, ...

3 месяца назад · 4 коммент...

Отправка запросов на сервер. HttpClient

6 месяцев назад · 1 коммент...

JavaScript

C#

Python

ListView

2 месяца назад · 1 коммент...

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

5 месяцев назад · 1 коммент...

Библиотека...

5 м...

42 Комментариев

[Войти](#)

G

Присоединиться к обсуждению...

войти с помощью или через DISCUS

D

f

t

G

Имя

4 [Поделиться](#) [Лучшие](#) [Новые](#) [Старые](#)

N

name name

3 года назад edited

Новичкам. Попробую объяснить проще. ООП - это способ позволяющий программисту создавать собственные типы переменных под названием класс. Вы уже знакомы с типами данных int или string. Так вот теперь вы сможете создавать собственные типы данных. Класс может объединять в себе много других переменных. Например вы создали класс "человек", чтобы объединить для каждого человека в вашей программе данные о его росте, весе, возрасте, стаже, должности и окладе. Если таких людей 1000, а данных на каждого надо хранить 30, то без ООП вам пришлось бы ввести в вашу программу 30 000 переменных.... С ООП вам достаточно сделать один класс, в котором вы будете хранить все эти переменные для каждого из 1000 человек. А данные введут операторы ПК, пользующиеся вашей программой. Всё что надо программисту, это написать код создания переменных созданного им типа данных "человек". У программистов эта переменная называется "объект"

Например tom = Person("Tom") и всё... Этот код создаёт объект типа "человек" и даёт этому объекту значение переменной "имя" равное "Tom"

Чтобы дать Тому возраст, нужен код tom.age = 32 и всё...

Чтобы дать Тому рост, нужен код tom.h= 182 и всё...

Чтобы дать Тому вес, нужен код tom.w= 82 и всё...

Чтобы дать Тому фамилию, нужен код tom.LastName= "Ivanov" и всё...

А ещё в классах можно хранить методы. Например, чтобы подсчитать соответствует ли рост его весу или распечатать его фамилию и т.д. Например метод tom.display\_info() из примеров выше печатает на экране "Привет, меня зовут Tom"

Конечно же все эти переменные - age, h, w, LastName или методы - display\_info() программист должен написать ручками как было показано выше.

Не сомневайтесь, что когда-нибудь вы напишете код для компьютерной игры следующего содержания: myMonster.Kill(Togo\_Von\_Gada, Bomba, Nasmerts\_nafig)

17 3 Ответить • Поделиться

M

manka

3 года назад

→ name name

Спасибо огромное. Коротко и ясно, достаточно для базового понимания

4 0 Ответить • Поделиться

Веошпилёт

→ name name