

Занятие 1

ООП: введение. Классы в Python



ООП

Классы

Объекты

**Принципы
ООП**

**Классы
в Python**

Парадигмы программирования



Процедурное (функциональное) программирование — классический подход, в котором задача разбивается на более мелкие подзадачи — процедуры. Объекты реального мира приходится представлять в виде переменных, списков и других структур. При усложнении задачи такой подход приводит к дублированию и ухудшению читаемости кода, возможным ошибкам.



Объектно-ориентированное программирование — подход, основанный на описании задачи как множества взаимодействующих объектов.

Объекты



Мы видим мир как множество объектов — предметов, животных, людей. Мы выделяем существенные характеристики объекта, которые отличают его от других, и таким образом можем классифицировать. Такой прием называется **абстракция** — выделение существенных характеристик объекта.



Объекты имеют внутреннее устройство (**данные**) и обладают определенным поведением (**действия**).

Объекты



Объектно-ориентированное программирование (ООП) — программирование, основанное на моделировании задачи как множества взаимодействующих объектов.



Класс — некий шаблон, описывающий множество объектов, имеющих общие данные и общее поведение.

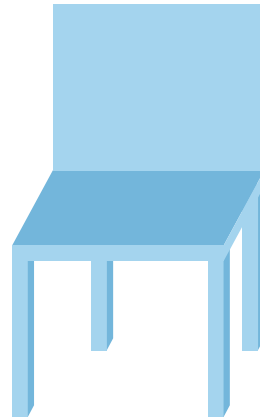


Объект — конкретный экземпляр класса, обладающий данными и поведением.

Пример



Chair (Стул)



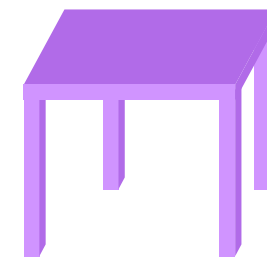
Количество ног
Цвет
Наличие спинки



Количество ног: 4
Цвет: зеленый
Наличие спинки: да



Количество ног: 2
Цвет: бежевый
Наличие спинки: да



Количество ног: 4
Цвет: розовый
Наличие спинки: нет

Свойства и методы

Класс

Данные

Действия

Класс имеет данные (свойства класса) и действия (методы класса).



Свойства — это данные класса.



Методы — это набор функций для работы с классом или объектом данного класса (действия).

Пример



Bike (Велосипед)

Свойства

Методы

Свойства:



количество колес



цвет



размер



...

Методы:



крутить педали



поворачивать руль



тормозить



...

Концепция ООП

Концепция ООП строится на основных принципах:



Инкапсуляция



Наследование



Полиморфизм

Инкапсуляция



Изменение свойств объекта только с помощью разрешенных методов и разрешенных свойств называется **инкапсуляцией**.

Этот принцип ООП заключается в сокрытии внутреннего устройства класса за **интерфейсом** — разрешенных методов и свойств.



Полиморфизм



Полиморфизм — это еще один принцип ООП, согласно которому функции могут работать с объектами разных классов. Такие функции называются полиморфными.

Например:

```
print(max(1, 2))  
print(max('a', 'b'))
```

Функция `max` — полиморфная функция.

Наследование



Наследование — это принцип ООП, согласно которому возможно использование **базового класса**, описывающего общие свойства и методы для других классов. Такие классы называют **классы наследники** или **производные классы**.

Class A

Родительский класс



Class B

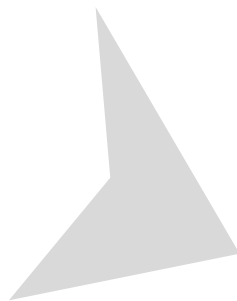
Класс наследник

Пример



Базовый класс

Shape (Фигура)



Цвет

Толщина линии

Координаты

Классы наследники

Rectangle (Прямоугольник)

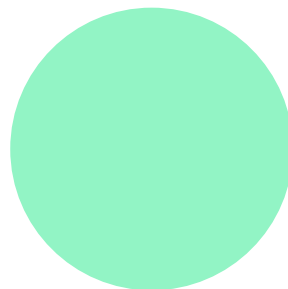


Цвет

Толщина линии

Координаты

Circle (Окружность)

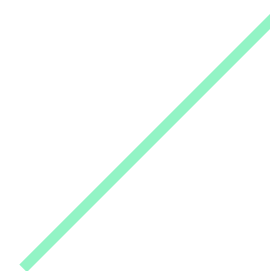


Цвет

Толщина линии

Координаты

Line (Линия)



Цвет

Толщина линии

Координаты

Промежуточные итоги



Объектно-ориентированное программирование или ООП — программирование, основанное на моделировании задачи как множества взаимодействующих объектов. Базовые понятия: **класс и объект**.

Концепция ООП строится на основных принципах:



Инкапсуляция



Наследование



Полиморфизм

Далее мы узнаем как работать с классами в Python.

Объекты в Python

Язык программирования Python является объектно-ориентированным. В Python абсолютно все является объектами. Переменные, функции, типы — все это объекты.

Например:

```
number = 1  
print(type(number))
```

Вывод:

```
<class 'int'>
```

type

С помощью вызова `type()` можно узнать к какому классу принадлежит объект:

```
print(type(1))  
print(type(2.5))  
print(type([1, 2, 3]))  
print(type(range(5)))
```

Вывод:

```
<class 'int'>  
<class 'float'>  
<class 'list'>  
<class 'range'>
```

Кстати, `type` — тоже объект:

```
print(type(type))
```

Вывод:

```
<class 'type'>
```


isinstance

Еще одним способом проверить принадлежность объекта классу является использование функции **isinstance**. Функция **isinstance** принимает на вход два аргумента: объект и класс, и возвращает True, если указанный объект является экземпляром указанного класса (или наследуется от него) и False, если не является.

Например:

```
print(isinstance(1, int))  
print(isinstance('hello', str))  
print(isinstance('123', int))  
print(isinstance([1,2,3], list))
```

Вывод:

```
True  
True  
False  
True
```

object

Класс **object** является базовым для всех классов в Python:

```
print(isinstance(1, object))  
print(isinstance(list, object))  
print(isinstance(object, object))  
print(isinstance(type, object))  
print(isinstance(object, type))
```

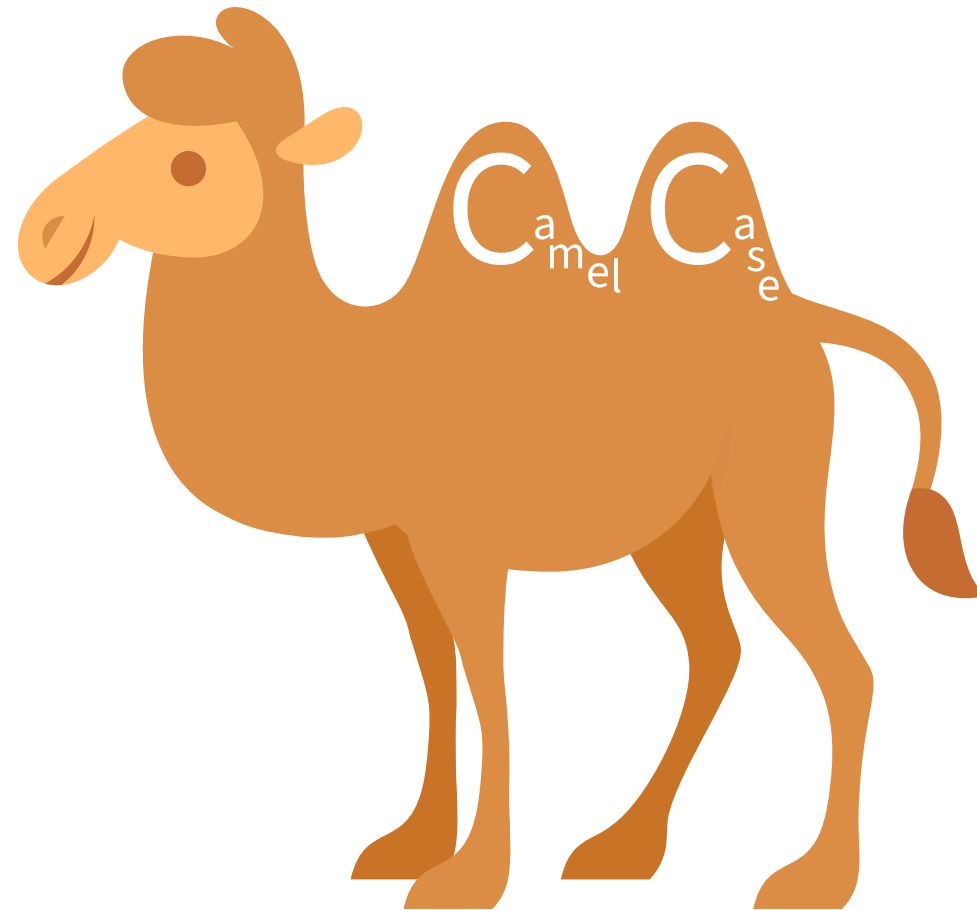
Вывод:

```
True  
True  
True  
True  
True
```

Создание класса

Чтобы создать свой класс в Python, используется следующий синтаксис:

```
class ИмяКласса:  
    # описание класса  
pass
```



По стандарту PEP8 имя класса записывается в стиле CamelCase (несколько слов пишутся слитно без пробелов, при этом каждое слово внутри пишется с прописной буквы).

Создание класса и объекта

Создадим класс `Point` для описания множества точек на плоскости. Чтобы создать объект (экземпляр класса), можно создать переменную и ей присвоить вызов класса — название класса с круглыми скобками.

```
class Point:  
    pass  
  
a = Point()  
print(type(a))  
print(isinstance(a, Point))
```

Вывод:

```
<class '__main__.Point'>  
True
```

Создание нескольких объектов

Мы можем создавать несколько объектов:

```
class Point:  
    pass  
  
a = Point()  
b = Point()  
print(type(a))  
print(isinstance(b, Point))  
print(id(a), id(b))
```


Вывод:

```
<class '__main__.Point'>  
True  
2016365270096 2016370060464
```

Данные класса

Добавим данные в класс. Теперь при создании объектов эти объекты будут иметь указанные нами в классе данные.

```
class Point:  
    x = 0  
    y = 0  
    color = 'black'  
  
a = Point()
```

```
✓  Point = {type} <class '__main__.Point'>  
01 color = {str} 'black'  
01 x = {int} 0  
01 y = {int} 0
```

Итоги



В Python все является объектом.



С помощью вызова `type(obj)` можно узнать к какому классу принадлежит объект.



С помощью вызова `isinstance(obj, class)` можно проверить принадлежность объекта классу.



Класс `object` является базовым для всех классов в Python.



Чтобы создать свой класс в Python, используется следующий синтаксис:

```
class ИмяКласса:  
    # описание класса
```



Чтобы создать объект необходимо вызвать класс — написать название класса с круглыми скобками.