

Классы и ООП



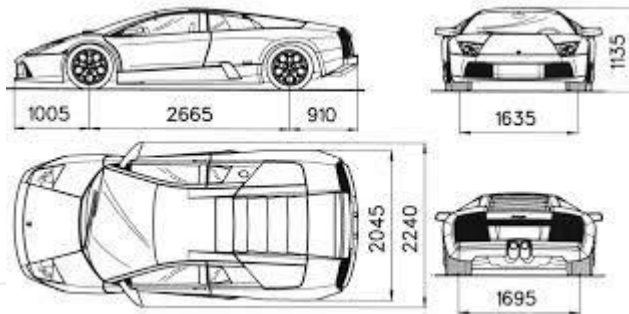
Основные понятия ООП

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт).

С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

Объект (экземпляр) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.

Интерфейс – это набор методов класса, доступных для использования другими классами.



Основные принципы ООП

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

Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

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

Существует несколько разновидностей полиморфизма. Две принципиально различных из них это параметрический полиморфизм и ad-hoc-полиморфизм, причём первая является истинной формой, а вторая — мнимой; прочие формы являются их подвидами или сочетаниями. **Параметрический полиморфизм** подразумевает исполнение одного и того же кода для всех допустимых типов аргументов, тогда как **ad-hoc-полиморфизм** подразумевает исполнение потенциально разного кода для каждого типа или подтипа аргумента. Бьёрн Страуструп определил полиморфизм как *«один интерфейс — много реализаций»*, но это определение покрывает лишь ad-hoc-полиморфизм.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс — потомком, наследником или производным классом.

А если я теперь знаю ООП, будет круто если я буду все писать классами?

Основные преимущества
объектно-ориентированного
программирования:



- Объектно-ориентированное программирование подразумевает **повторное использование**.
- Использование **модулярного подхода** в объектно-ориентированном программировании позволяет получить читаемый и гибкий код;
- В объектно-ориентированном программировании каждый класс имеет определенную задачу.
- **Инкапсуляция данных** вносит дополнительный уровень безопасности в разрабатываемую программу с использованием объектно-ориентированного подхода

Основы ООП на Python

Создавать классы в Python очень просто:

```
1 class SomeClass(object):  
2     pass  
3
```

Создать экземпляр:

```
instance_parent_class = ParentClass1()  
instance_some_class = SomeClass()  
|
```

Классы-родители перечисляются
в скобках через запятую:

```
1 class ParentClass1(object):  
2     pass  
3  
4  
5 class ParentClass2(object):  
6     pass  
7  
8  
9 class SomeClass(ParentClass1, ParentClass2):  
10     pass  
11
```

Основы ООП на Python

Свойства классов устанавливаются с помощью простого присваивания:

```
9 class SomeClass(object):
10     attr1 = 'value_attr1_1'
11     attr2 = 'value_attr1_2'
12
```

Методы объявляются как простые функции:

```
9 class SomeClass(object):
10     attr1 = 'value_attr1_1'
11     attr2 = 'value_attr1_2'
12
13     def method1(self, x):
14         pass
15
```

__init__

```
1 class Car(object):
2
3     def __init__(self, color, engine):
4         self.color = color
5         self.engine = engine
6
7
8 car1 = Car('white', '1.6')
9 car2 = Car('red', '0.8')
10
11 print(car1.color) # white
12 print(car2.engine) # 0.8
```

Метод `__init__` известен как конструктор, так как он запускается на этапе конструирования экземпляра.

Этот метод является типичным представителем большого класса методов, которые называются методами перегрузки операторов, а еще магическими методами в Python.

Зачем нужно наследование?

```
1 class Employee(object):
2     def compute_salary(self):
3         return 10
4
5     def give_raise(self):
6         pass
7
8     def retire(self):
9         pass
10
11
12 class Engineer(Employee):
13     def compute_salary(self):
14         return 20
15
```

```
1 class Car(object):
2
3     def ride(self):
4         print('Riding on a ground')
5
6     def beep(self):
7         print('Car beep')
8
9
10 class Boat(object):
11     def swim(self):
12         print('Sailing in the ocean')
13
14     def beep(self):
15         print('Boat beep')
16
17
18 class Amphibian(Car, Boat):
19     pass
20
21
22 my_car = Amphibian()
23 print(my_car.ride()) # Riding on a ground
24 print(my_car.swim()) # Sailing in the ocean
25 print(my_car.beep()) # Car beep
26
```


Контрольные вопросы

- 1) Каково основное назначение ООП в языке Python?
- 2) Где выполняется поиск унаследованных атрибутов?
- 3) В чем разница между объектом класса и объектом экземпляра?
- 4) В чем состоит особенность первого аргумента в методах классов?
- 5) Для чего служит метод `__init__`?
- 6) Как определяются суперклассы для класса?
- 7) В чем разница между атрибутами и методами?

Магические методы

- Имена методов, начинающиеся и заканчивающиеся двумя символами подчеркивания (`__X__`), имеют специальное назначение.
- Такие методы вызываются автоматически, когда экземпляр участвует во встроенных операциях.
- Классы могут переопределять большинство встроенных операторов.

```
1 class Class1(object):
2
3     def __init__(self, attr1):
4         self.attr1 = attr1
5
6     def __add__(self, other):
7         return 'Boom! {}'.format(other)
8
9
10 a = Class1(1)
11 print(a + 1)  # Boom! 1
12
```

```
1 class Class1(object):
2
3     def __init__(self, value):
4         self.value = value
5
6     def __lt__(self, other): # <
7         return False
8
9     def __gt__(self, other): # >
10        return False
11
12
13 a = Class1(1)
14 b = Class1(2)
15 print(1 < 2) # True
16 print(1 > 2) # False
17 print(a < b) # False
18 print(a > b) # False
```

Статические и классовые методы

@classmethod

Иногда в программах бывает необходимо организовать обработку данных, связанных с классами, а не с экземплярами. Например, следить за числом экземпляров класса или вести список всех экземпляров класса, находящихся в настоящий момент в памяти.

@staticmethod

Статические методы никогда автоматически не получают ссылку `self` на экземпляр, независимо от того, вызываются они через имя класса или через экземпляр. Такие методы обычно используются для обработки информации, имеющей отношение ко всем экземплярам, а не для реализации поведения экземпляров.

@staticmethod

Для создания статических методов в Python предназначен декоратор `@staticmethod`. У них нет обязательных параметров-ссылок вроде `self`. Доступ к таким методам можно получить как из экземпляра класса, так и из самого класса

```
1  class SomeClass(object):
2      @staticmethod
3      def hello():
4          print("Hello, world")
5
6
7  SomeClass.hello()  # Hello, world
8  obj = SomeClass()
9  obj.hello()  # Hello, world
10 |
```

@classmethod

Еще есть так называемые методы классов. Они аналогичны методам экземпляров, но выполняются не в контексте объекта, а в контексте самого класса (классы – это тоже объекты). Такие методы создаются с помощью декоратора `@classmethod` и требуют обязательную ссылку на класс (`cls`).

```
1 class SomeClass(object):
2     @classmethod
3     def hello(cls):
4         print('Hello, класс {}'.format(cls.__name__))
5
6
7 SomeClass.hello() # Hello, класс SomeClass
8
```

```
1 class SomeClass(object):
2     numInstances = 0
3
4     @classmethod
5     def count(cls):
6         cls.numInstances += 1
7
8     def __init__(self):
9         self.count()
10
11
12 a = SomeClass()
13 print([a.numInstances]) # [1]
14 b = SomeClass()
15 c = SomeClass()
16
17 print([a.numInstances, b.numInstances, c.numInstances]) # [3, 3, 3]
18
```

Принципы ООП в Python

```
1 class SomeClass:
2     def __private(self):
3         print("Private!")
4
5
6 obj = SomeClass()
7 obj.__private() # Private!
8
9
10
11
12 class SomeClass():
13     def __init__(self):
14         self.__param = 42
15
16
17 obj = SomeClass()
18 obj.__param # AttributeError: 'SomeClass' object has no attribute '__param'
19 obj._SomeClass__param # 42
20
```

Access to a protected member `__private` of a class

Rename element Alt+Shift+Enter More actions... Alt+Enter

Пишем классы. Person & Manager

```
1 class Person(object):
2
3     def __init__(self, name, job=None, pay=0):
4         self.name = name
5         self.job = job
6         self.pay = pay
7
8     def __str__(self):
9         return 'Person {} with job -- {}, and pay = {}'.format(self.name, self.job, self.pay)
10
11     def last_name(self, adj):
12         return '{} is {}'.format(self.name.split()[-1], adj)
13
14     def give_raise(self, percent):
15         self.pay = self.pay + self.pay*percent
16
17
18 class Developer(Person):
19
20     def __init__(self, name):
21         super(Developer, self).__init__(name, job='Dev', pay=1000)
22
23     def give_raise(self, percent):
24         self.pay = self.pay + 10000
25
```

Написать класс, который бы по всем внешним признакам был бы словарем


Написать класс, который бы по всем внешним признакам был бы словарем, но позволял обращаться к ключам как к атрибутам.

```
>>>x = DictAttr([('one', 1), ('two', 2), ('three', 3)])
>>> x
{'one': 1, 'three': 3, 'two': 2}
>>> x['three']
3
>>> x.get('one')
1
>>> x.get('five', 'missing')
'missing'
>>> x.one
1
>>> x.five
Traceback (most recent call last):
...
AttributeError
```


Спасибо

1. Марк Лутц, Изучаем Python. 4-е издание
2. <https://habr.com/ru/post/87119/>
3. <https://habr.com/ru/post/186608/>
4. <https://proglib.io/p/python-oop/>
5. <https://habr.com/ru/post/72757/>

Thanks for your
attention

 [artezio_software](#)

 info@artezio.com

www.artezio.com