

Python . Классы.

- Python имеет полноценную поддержку объектно-ориентированного программирования: можно определять собственные классы, наследоваться от встроенных и собственных классов (одного или нескольких), производные классы могут переопределять любые методы базовых классов.

Определение класса

- Определения класса в языке Python начинается с зарезервированного слова `class` и следующего за ним именем класса. Пример простейшего класса:

```
class Student :  
    pass  
  
s=Student()  
s.name="Vasya"  
s.age=22  
s.univer="AFTU"  
print s.name, s.age, s.univer      # Vasya 22 AFTU
```

Объекты могут содержать произвольное количество собственных данных.

Определение методов класса

- class Student :
 def f (self, n, y) :
 self.name=n
 self.year=y
 print self.name, "is on the", self.year, "-th year"

```
s1=Student()  
s2=Student()  
s1.f ("Vasya", 5)  
s2.f ("Petya", 6)
```

результат выполнения:
Vasya is on the 5-th year
Petya is on the 6-th year

Метод __init__()

- Если для класса определен метод __init__(), то он автоматически вызывается сразу после создания экземпляра класса.

```
class Student :  
    def __init__(self, c) :  
        self.city=c  
    def f (self, n, y) :  
        self.name=n  
        self.year=y  
        print self.name, "is on the", self.year, "-th year"  
  
s=Student("St.Petersburg")  
print s.city                                # St.Petersburg
```

Self

- Первым аргументом каждого метода класса всегда является текущий экземпляр класса. Общепринято называть этот аргумент `self` (аналог слова `this` в C++). `Self` ссылается на экземпляр класса для которого вызывается метод. В методе `__init__` (рассмотрим далее) `self` ссылается на только что созданный объект. При вызове метода `self` не указывается, Python добавит его автоматически.
- Экземпляры классов нет необходимости удалять явно, так как удаление происходит автоматически, когда на них больше нет ссылок.

```
def info():
    s=Student("St.Petersburg")
    for i in xrange(100) :
        info()
```

- Совсем необязательно, чтобы определение функции находилось в определении класса:

```
def f (self, x) :  
    print "Hello,", x  
class C :  
    f1=f  
    def f2 (self) :  
        print "Hello, students!!!"
```

x=C()	# результат выполнения:
x.f1("students!")	# Hello, students!
x.f2()	# Hello, students!!!

Атрибуты классов

- Слово «объект» в языке Python необязательно означает только экземпляр класса. В языке Python все типы данных являются объектами. И классы сами являются объектами, атрибуты которых есть все имена, помещенные в пространство имен класса при создании объекта-класса.

```
class MyClass:  
    'Simple Class'  
    i=123  
    def f(x) :  
        print "Hello!"
```

```
print MyClass.i  
print MyClass.__doc__  
MyClass.i=124  
print MyClass.i
```

```
# 123  
# Simple Class  
# 124
```

Атрибуты классов

- ```
x=MyClass()
x.f() # Hello!
MyClass.f(x) # Hello!
x.i=125
y=MyClass()
print x.i, y.i, MyClass.i # 125 124 124
```
- Атрибуты классов могут быть использованы как аналоги статических переменных классов в C++:

```
class Counter:
 count = 0
 def __init__(self):
 self.__class__.count += 1

print Counter.count # 0
c = Counter()
print c.count, Counter.count # 1 1
d = Counter()
print c.count, d.count, Counter.count # 2 2 2
```

# Частные атрибуты классов

- Говоря в терминологии C++, все атрибуты класса открытые (public). Атрибуты, являются частными, доступными только из методов объекта класса, если их имена содержат не менее двух символов подчеркивания в начале и не более одного символа подчеркивания в конце:

```
class Student :
 def __init__(self, c) :
 self.__city=c
 def __f(self, n, y) :
 self.name=n
 self.year=y
 print self.name, "is on the", self.year, "-th year"
```

```
s1=Student("St.Petersburg")
print s1.__city
s1.__f ("Vanya", "5")
```

результат выполнения:

```
print s1.__city
AttributeError: Student instance has no attribute '__city'
```

```
print s1._Student__city
s1._Student__f("Vanya", "5")
```

результат выполнения:

```
St.Petersburg
Vanya is on the 5 -th year
```

# Наследование

- class Person :

```
def __init__(self, n) :
 self.name=n
def write(self) :
 print self.name
```
- class Student (Person) :

```
def __init__(self, gr, n) :
 Person.__init__(self, n)
 self.group=gr
def write(self) :
 print self.name, self.group
```
- p=Person("Petya")  
p.write() # Petya  
s=Student(23, "Vasya")  
s.write() # Vasya 23

# Множественное наследование

- class A1 :

```
 def fb (self) :
 print "class1"
```

```
class A2 :
```

```
 def fb (self) :
 print "class2"
```

```
class B(A1, A2) :
```

```
 def f(self) :
 self.fb()
```

```
b=B()
```

```
b.f()
```

результат выполнения:

```
class1
```

# Полиморфизм

```
class Based:
 def __init__(self, n) :
 self.numb = n
 def out (self):
 print self.numb

class One(Based) :
 def multi(self,m) :
 self.numb *= m

class Two(Based) :
 def inlist (self) :
 self.inlist = list(str(self.numb))
 def out(self) :
 i = 0
 while i < len (self.inlist) :
 print (self.inlist[i])
 i += 1
```

```
obj1 = One(45)
obj2 = Two('abc')
```

```
obj1.multi(2)
obj1.out()
```

```
obj2.inlist()
obj2.out()
```

результат выполнения:

```
90
a
b
c
```

# Специальные атрибуты классов

- Объекты-классы имеют следующие специальные атрибуты:

`__name__` Имя класса.

`__module__` Имя модуля, в котором класс определен.

`__doc__` Страна документации класса или `None`, если она не определена.

`__bases__` Кортеж базовых классов в порядке их следования в списке базовых классов.

`__dict__` Словарь атрибутов класса.

```
print Two.__name__ # Two
```

```
print Two.__module__ # __main__
```

```
print Two.__doc__ # None
```

```
print Two.__bases__ # (<class __main__.Based at 0xb7d6420c>,)
```

```
print Two.__dict__ # {'__module__': '__main__', 'inlist': <function
inlist at 0xb7d8d72c>, '__doc__': None,
'out': <function out at 0xb7d8d764>}
```