Классы и итераторы

Введение

- 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.

Что такое класс?

Определения класса в языке Python начинается с зарезервированного слова class и следующего за ним имени 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.

Методы 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 5th year
Petya is on the 6th 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__ 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.

- 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 объектакласса.

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

print MyClass.i # 123
print MyClass.__doc__ # Simple Class
MyClass.i=124
print MyClass.i # 124
```

Атрибуты классов (продолжение)

```
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"
```

Частные атрибуты классов (продолжение)

```
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 5th 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) :
        elf.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
```

Полиморфизм (продолжение)

```
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__ Строка документации класса или None, если она не определенна.
- __bases__ Кортеж базовых классов в порядке их следования в списке базовых классов.
- __dict__ Словарь атрибутов класса.

Итераторы

```
class Fib:
    '''iterator that yields numbers in the Fibonacci sequence'''

    def __init__(self, max):
        self.max = max

    def __iter__(self):
        self.a = 0
        self.b = 1
        return self

    def __next__(self):
        fib = self.a
        if fib > self.max:
            raise StopIteration
        self.a, self.b = self.b, self.a + self.b
        return fib
```

Задание

Реализуйте итератор, который возвращает последовательность квадратов целых чисел меньших, чем заданное число.