Интерфейсы

Andrew Svetlov
andrew.svetlov@gmail.com
asvetlov.blogspot.com

Интерфейсы

- Duck Typing
- ABC
- Zope Interface

Duck Typing

```
class A:
  def f(self):
    print('A.f')
class B:
  def f(self):
    print('B.f')
```

```
>>> a = A()
>>> b = B();
>>> a.f()
A.f
>>> b.f()
b.f
```

Pro/Contra

- Естественная запись
- Ничего лишнего
- hasattr vs isinstance

- Неявность интерфейса
- Сложность само-документирования
- a.x => Point ???

Iterable

- __iter__
- __getitem__
  пока не IndexError

```
class A:
  def __getitem__(self, i):
    if i < 3:
      return i
    raise IndexError("out
                of range")
>>> a = A()
>>> i = iter(a)
>>> list(i)
[0, 1, 2]
```

Конструктор dict

- Аргумент может быть:
- dict
- Sequence of 2-elem sequences
- Mapping??? .keys!!!
- Iterable

Наследование

```
class Base:
  def f(self):
    raise NotImplementedError()

class A(Base):
  def f(self):
    return 1
>>> a = A()
>>> isinstance(a, Base)
True
>>> b = Base()  # ???
```

ABC (Abstract Base Classes)

```
class Base(abc.Meta):
  @abc.abstractmethod
  def f(self):
    return 0
class A(Base):
  def f(self):
    return super().f() + 1
>>> a = A()
>>> isinstance(a, Base)
True
>>> b = Base()  # ???
Can't instantiate abstract class Base with abstract methods f
>>> a.f()
1
```

collections.abc

- Hashable
- Iterable
- Iterator
- Sized
- Container
- Callable
- Set
- MutableSet
- ByteString
- Mapping
- MutableMapping
- MappingView
- KeysView
- ItemsView
- ValuesView
- Sequence
- MutableSequence

Самодельное множество

```
from collections import Set
class S(Set):
  def __init__(self, s):
    super().__init__()
    self.s = frozenset(s)
  def __contains__(self, i):
    return i in self.s
  def __iter__(self):
    return iter(self.s)
  def __len__(self):
    return len(self.s)
```

```
>>> s1 = S({1, 2})
>>> 2 in s1
True
>>> s2 = S({2, 3})
>>> s3 = s1|s2
>>> list(s3)
[1, 2, 3]
>>> s4 = s1 - s2
>>> list(s4)
[1]
```

Необязательные методы

- Наличие метода
- Проверка на None
- Исключение NotImplementedError
- Значение NotImplemented

Наличие метода и None

```
class A:
  post = None
  def do(self):
    if hasattr(self,
               'pre'):
      self.pre()
    print('A.do')
    if self.post is not None:
      self.post()
>>> a = A()
>>> a.do()
A.do
```

```
class B(A):
  def pre(self):
     print('B.pre')
  def post(self):
     print('B.post')


>>> b = B()
>>> b.do()
B.pre
A.do
B.post
```

NotImplementedError

```
class A:
  def pre(self):
    raise NotImplementedError(
       'implement pre method')

  def do(self):
    try:
      self.pre()
    except NotImplementedError:
      pass
    print('A.do')
```

```
class B(A):
  def pre(self):
     print('B.pre')
```

```
>>> a = A()
>>> a.do()
A.do
>>> b = B()
>>> b.do()
B.pre
a.do
```

NotImplemented

```
class A:
  def override(self):
    return NotImplemented
  def do(self):
    val = self.override()
    if (val is not
        NotImplemented):
      return val
    else:
      return 'default'
```

```
class B:
  def override(self):
    return 'overriden'
>>> a = A()
>>> a.do()
'default'
>>> b = B()
>>> b.do()
'overriden'
```

Zope Interface

```
class IFile(Interface):
  body = Attribute('Contents of the file.')
class ISize(Interface):
  def getSize():
    'Return the size of an object.'
class File(object):
  implements(IFile)
  body = 'foo bar'
class FileSize(object):
  implements(ISize)
  __used_for__ = IFile
  def __init__(self, context):
    self.context = context
  def getSize(self):
    return len(self.context.body)
```

Адаптеры

```
registry = AdapterRegistry()

def hook(provided, obj):
  adapter = registry.lookup1(providedBy(obj),
                             provided, '')
  return adapter(object)

adapter_hooks.append(hook)


>>> file = File()
>>> size = ISize(file)
>>> size.getSize()
```

Вопросы?

Andrew Svetlov
andrew.svetlov@gmail.com
asvetlov.blogspot.com