Лекция 7
Наследование
Работа с вебом

30 марта 2017 г.

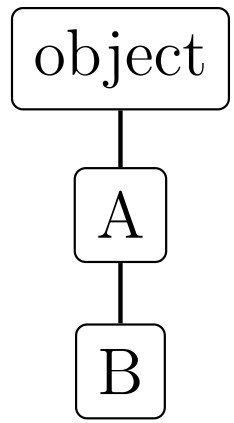
Наследование

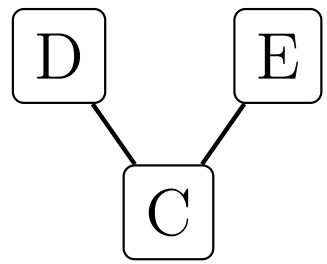
Объявление класса

```
class <NAME>(<PARENT CLASSES>):
    <BODY>
```

Примеры

```
class A(object):
    pass

class B(A):
    pass

class C(D, E):
    pass
```





Наследование атрибутов

```
class A(object):
    X = 1
    def f(self):
        print "Called A.f()"

class B(A):
    pass
```

Наследование атрибутов

```
class A(object):
    X = 1
    def f(self):
        print "Called A.f()"

class B(A):
    pass

b = B()
b.f()
print b.X
```

```
Called A.f()
1
```

Переопределение атрибутов (override)

```
class A(object):
    def f(self):
        print "Called A.f()"
class B(A):
    def f(self):
        print "Called B.f()"
```

Переопределение атрибутов (override)

```
class A(object):
    def f(self):
        print "Called A.f()"
class B(A):
    def f(self):
        print "Called B.f()"

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

```
Called A.f()
Called B.f()
```

Частичное переопределение

```
class A(object):
    NAME = "A"
    def f(self):
        print self.NAME
class B(A):
    NAME = "B"
```

Частичное переопределение

```
class A(object):
    NAME = "A"
    def f(self):
        print self.NAME
class B(A):
    NAME = "B"

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

```
A
B
```

Переопределение конструктора

```
class A(object):
    def __init__(self):
        self.x = 1
class B(A):
    def __init__(self):
        self.y = 2

b = B()
print b.x
```

```
AttributeError: ...
```

```
print b.y
```

```
2
```

Доступ к переопределенным методам

```
class A(object):
    def __init__(self):
        self.x = 1
class B(A):
    def __init__(self):
        A.__init__(self)
        self.y = 2
```

Доступ к переопределенным методам

```
class A(object):
    def __init__(self):
        self.x = 1
class B(A):
    def __init__(self):
        A.__init__(self)
        self.y = 2

b = B()
print b.x, b.y
```

```
1 2
```

Доступ к переопределенным методам

```
class A(object):
    def __init__(self):
        self.x = 1
class B(A):
    def __init__(self):
        A.__init__(self)
        self.y = 2

b = B()
print b.x, b.y
```

```
1 2
```

Нежелательный способ (обычно)

Функция `super`

`super(type, object)`

- Возвращает временный объект, который делегирует вызовы методов к родителю
- Только для классов-наследников `object`

Доступ к переопределенным методам

```
class A(object):
    def __init__(self):
        self.x = 1
class B(A):
    def __init__(self):
        super(B, self).__init__()
        self.y = 2

b = B()
print b.x, b.y
```

```
1 2
```

Расширение методов (extend)

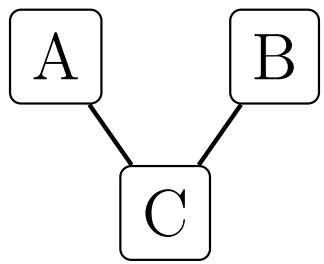
```
class A(object):
    def f(self, a, b):
        return a + b

class B(A):
    def f(self, a, b):
        return super(B, self).f(a, b) * 2

b = B()
print(b.f(1, 2))
```

```
6
```

Множественное наследование

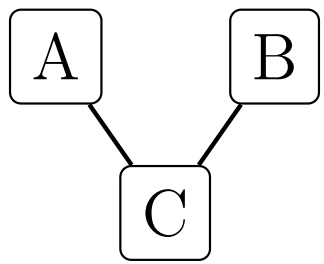
Множественное наследование

```
class C(A, B):
    pass
```



Множественное наследование

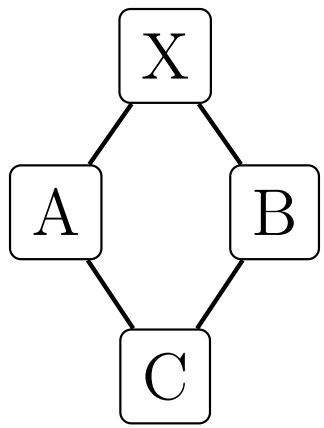
```
class C(A, B):
    pass
```



- Порядок разрешения имен?

Ромбовидное наследование

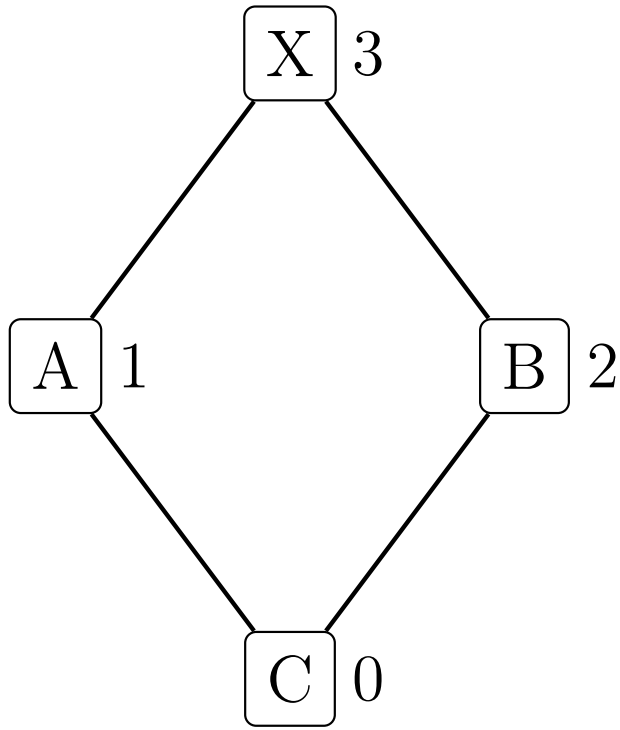
```
class A(X):
    pass
class B(X):
    pass
class C(A, B):
    pass
```



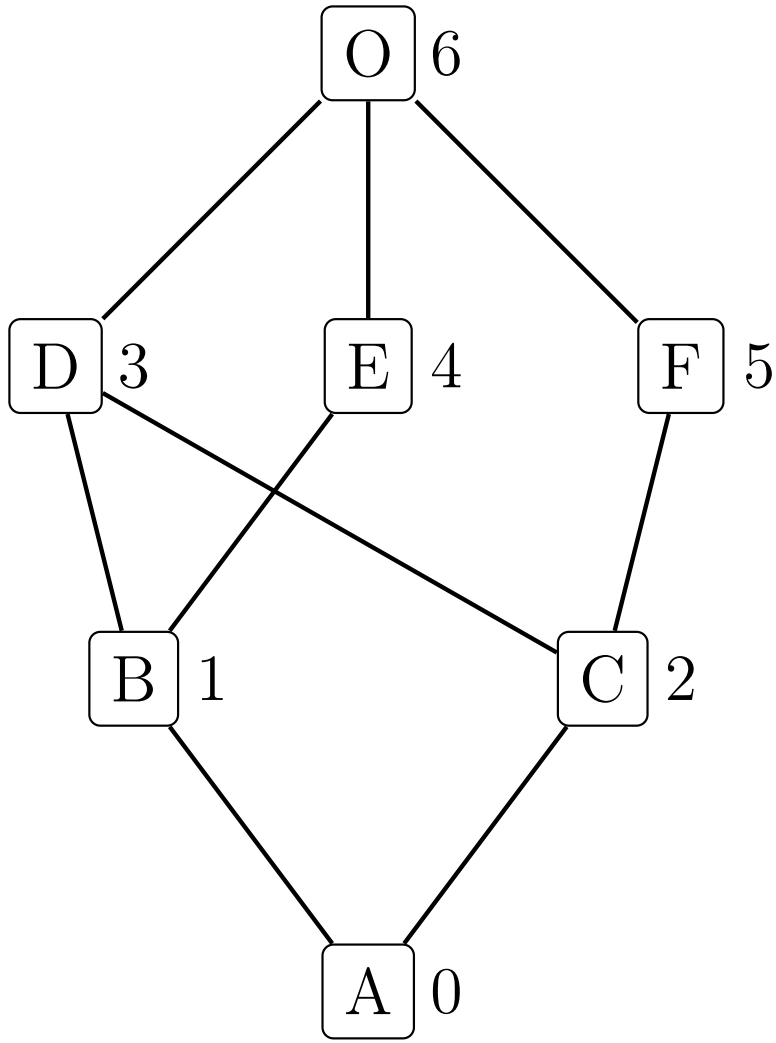
Порядок разрешения имен

MRO — Method resolution order
Используется алгоритм C3-линеаризации.

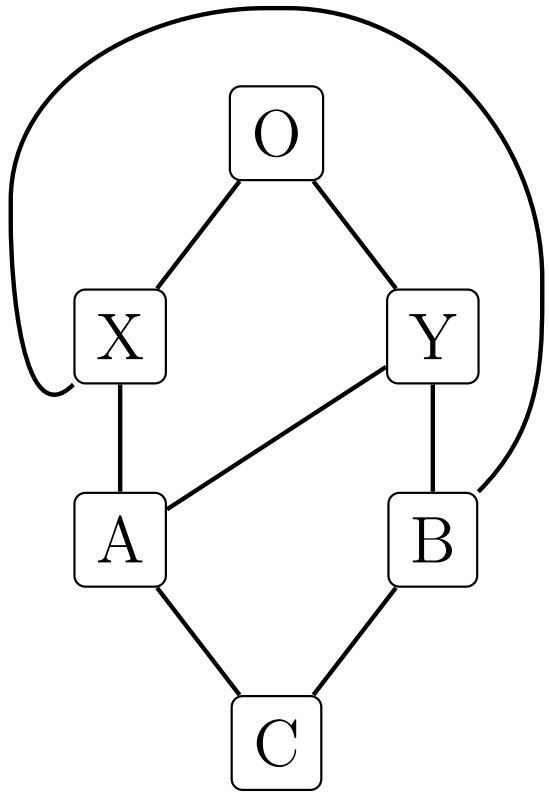
Порядок разрешения имен



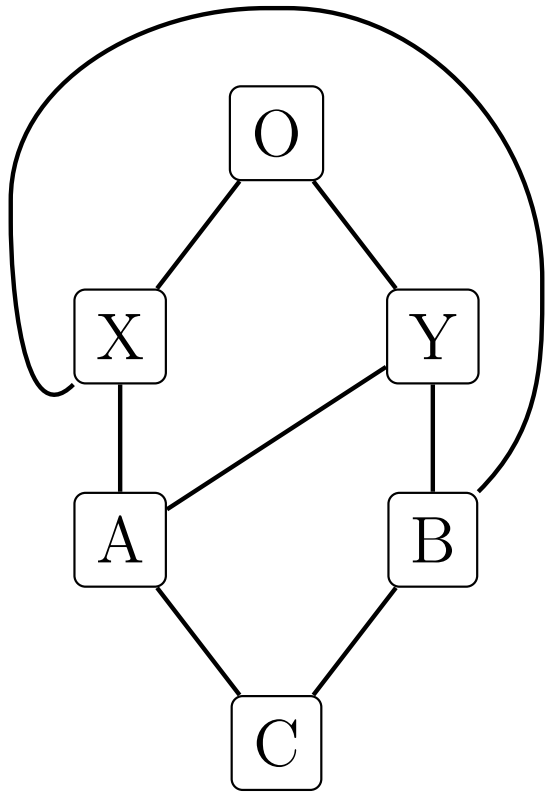
Порядок разрешения имен



Невозможная иерархия



Невозможная иерархия




```
TypeError: ... Cannot create a
consistent method resolution order
(MRO) for bases X, Y.
```

Получение порядка имен

```
class F(object): pass
class E(object): pass
class D(object): pass
class C(D,F): pass
class B(D,E): pass
class A(B,C): pass
```

Получение порядка имен

```
class F(object): pass
class E(object): pass
class D(object): pass
class C(D,F): pass
class B(D,E): pass
class A(B,C): pass
```

```
A.mro()
```

```
[__main__.A,
 __main__.B, __main__.C,
 __main__.D, __main__.E, __main__.F,
 object]
```

Порядок разрешения имен в общем

```
c = C()
c.foo
```

- *Поискать атрибут через механизм дескрипторов* (познакомимся позже)
- Поискать атрибут в `c.__dict__`
- Поискать атрибут в `C.__dict__`
- Поискать атрибут в классах-родителях `C`
- `raise AttributeError`

Доступ к переопределенным методам

```
class A(B, C):
    def f(self):
        super(A, self).f()
```

Работа с интернет-ресурсами

urllib2

```
from urllib2 import urlopen
```

urllib2

```
from urllib2 import urlopen
conn = urlopen('http://example.com')
```

urllib2

```
from urllib2 import urlopen

conn = urlopen('http://example.com')

data = conn.read()
html = data.decode('utf-8')
real_url = conn.geturl()
conn_info = conn.info()
http_code = conn.getcode()
```

urllib2

```
from urllib2 import urlopen

conn = urlopen('http://example.com')

data = conn.read()
html = data.decode('utf-8')
real_url = conn.geturl()
conn_info = conn.info()
http_code = conn.getcode()

conn.close()
```

Менеджеры контекстов для соединений

Python 2

```
from contextlib import closing

with closing(urlopen(url)) as conn:
    ...
```

Python 3

```
from urllib.request import urlopen

with urlopen(url) as conn:
    ...
```

Кодирование ссылок

```
from urllib import urlencode
urlencode((('id', 1), ('key', '&%^##')))
```

```
'id=1&key=%26%25%5E%23%23'
```

Кодирование ссылок

```
from urllib import urlencode
urlencode((('id', 1), ('key', '&%^##')))
```

```
'id=1&key=%26%25%5E%23%23'
```

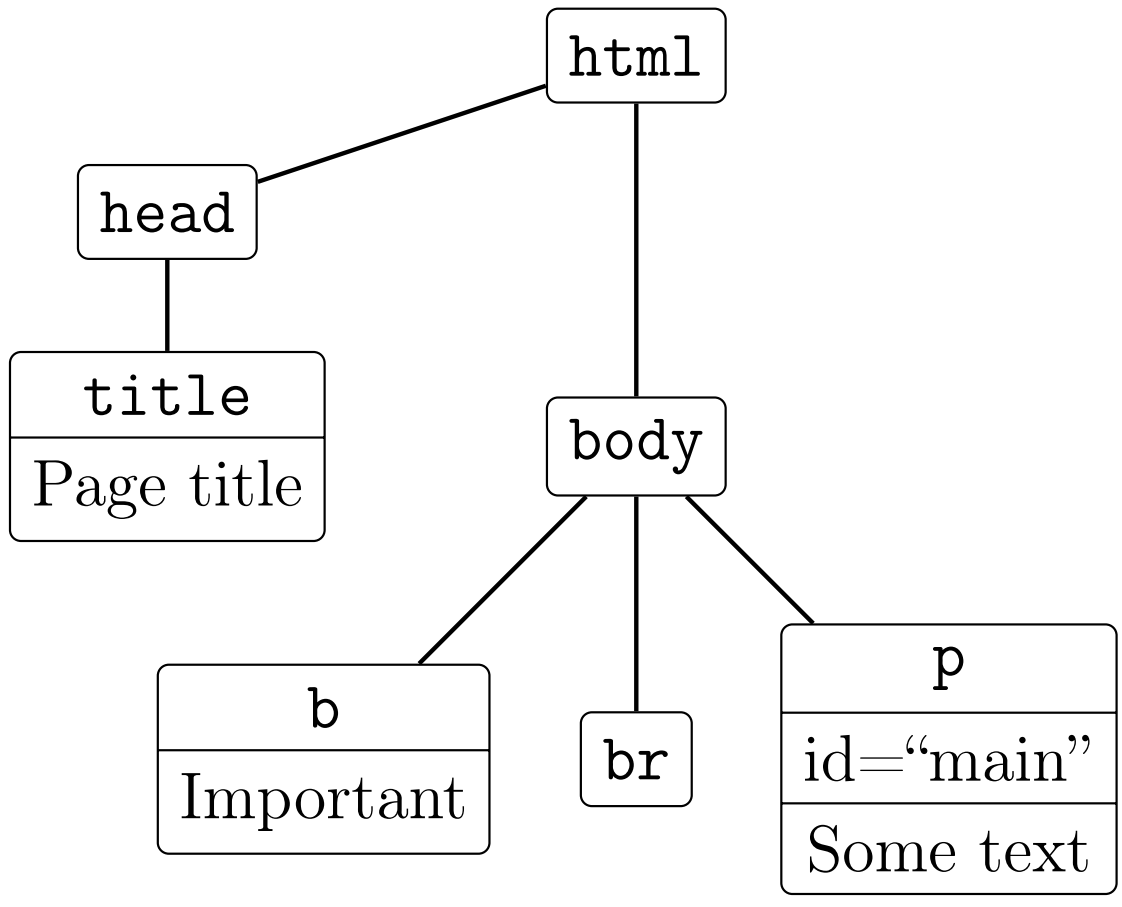
```
from urllib import quote
quote(u'Привет'.encode('utf-8'))
```

```
'%D0%9F%D1%80%D0%B8%D0%B2%D0%B5%D1%82'
```

Обработка XML и HTML

Дерево элементов

```
<html>
  <head>
    <title>
      Page title.
    </title>
  </head>
  <body>
    <b>
      Important.
    </b>
    <br/>
    <p id="main">
      Some text.
    </p>
  </body>
</html>
```



xml.etree.ElementTree

```
import xml.etree.ElementTree as ET
```

xml.etree.ElementTree

```
import xml.etree.ElementTree as ET

tree = ET.parse('sample.html')
root = tree.getroot()
```

xml.etree.ElementTree

```
import xml.etree.ElementTree as ET

tree = ET.parse('sample.html')
root = tree.getroot()

par = root[1][2]
par.tag
par.attrib
par.text
```

```
'p'
{'id': 'main'}
'\n Some text.\n '
```

xml.etree.ElementTree

```
body = None
for child in root:
    if child.tag == 'body':
        body = child
        break
texts = []
for child in body:
    if (child.tag == 'p' and
            'id' in child.attrib and
            child.attrib['id'] == 'main'):
        texts.append(child.text)

print texts
```

```
['\n Some text.\n ']
```

xml.etree.ElementTree и XPath

Текущее поддерево

```
root.findall('.')
```

```
[<Element 'html' at 0x3b7cb90>]
```

xml.etree.ElementTree и XPath

Текущее поддерево

```
root.findall('.')
```

```
[<Element 'html' at 0x3b7cb90>]
```

Все элементы, которые можно получить из текущего поддерева, встретив на пути теги <body> и <b>.

```
root.findall('./body/b')
```

```
[<Element 'b' at 0x3b7ccd0>]
```

xml.etree.ElementTree и XPath

Все элементы, которые можно получить из текущего поддерева, встретив на пути любые теги (//), а потом теги <p> и

```
root.findall('.//p/span')
```

```
[]
```

xml.etree.ElementTree и XPath

Все элементы, которые можно получить из текущего поддерева, встретив на пути любые теги (//), а потом теги <p> и

```
root.findall('.//p/span')
```

```
[]
```

Все элементы, которые можно получить из текущего поддерева, встретив на пути любые теги (//), а потом теги <p> и с атрибутом id равным main.

```
root.findall('.//p[@id="main"]')
```

```
[<Element 'p' at 0x3b7cd90>]
```

Внешние библиотеки

`xml.etree` плохо работает со «сломанным» XML

(почти любая HTML-страница — «сломанный» XML)

Внешние библиотеки

`xml.etree` плохо работает со «сломанным» XML

(почти любая HTML-страница — «сломанный» XML)

Альтернативы (не в стандартной библиотеке):

- lxml
  `lxml.html` имеет тот же синтаксис, что и `xml.etree`, но может работать с HTML.
- BeautifulSoup

HTMLParser

```
from HTMLParser import HTMLParser
```

HTMLParser

```
from HTMLParser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print "Start tag:", tag, attrs
    def handle_endtag(self, tag):
        print "End tag:", tag
    def handle_data(self, data):
        text = data.strip()
        if text:
            print "Data:", data.strip()
```

HTMLParser

```
from HTMLParser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print "Start tag:", tag, attrs
    def handle_endtag(self, tag):
        print "End tag:", tag
    def handle_data(self, data):
        text = data.strip()
        if text:
            print "Data:", data.strip()

parser = MyHTMLParser()
parser.feed(html)
```

HTMLParser

```
Start tag: html []
Start tag: head []
Start tag: title []
Data: Page title.
End tag: title
End tag: head
Start tag: body []
Start tag: b []
Data: Important.
End tag: b
```

```
Start tag: br []
End tag: br
Start tag: p
[('id', 'main')]
Data: Some text.
End tag: p
End tag: body
End tag: html
```

HTMLParser

```
class MainTextParser(HTMLParser):
    def __init__(self):
        HTMLParser.__init__(self)
        self.main_text = False
    def handle_starttag(self, tag, attrs):
        if (tag == 'p' and
                any([(k, v) == ('id', 'main')
                     for k, v in attrs)]):
            self.main_text = True
    def handle_endtag(self, tag):
        if self.main_text and tag == 'p':
            self.main_text = False
    def handle_data(self, data):
        if self.main_text:
            print data
```