Лекция 5
Модель данных.
Тестирование.

17 марта 2016 г.

Модель данных

Объекты

- Все данные в программе — объекты
  - Числа, списки, строки
  - Классы, функции

Объекты

- Все данные в программе — объекты
  - Числа, списки, строки
  - Классы, функции

- Основные свойства объектов
  - *Идентичность* (identity)
  - *Тип* (type)
  - *Значение* (value)

Свойства объектов

- Идентичность
- Тип
- Значение

Свойства объектов

- Идентичность
  - $\sim$ адрес в памяти
  - Не может изменяться
  - `id()`, `is`, `is not`
- Тип

- Значение

Свойства объектов

- Идентичность
  - $\sim$ адрес в памяти
  - Не может изменяться
  - `id()`, `is`, `is not`
- Тип
  - Возможные операции и значения
  - Не может изменяться
  - `type()`
- Значение

Свойства объектов

- Идентичность
  - $\sim$ адрес в памяти
  - Не может изменяться
  - `id()`, `is`, `is not`
- Тип
  - Возможные операции и значения
  - Не может изменяться
  - `type()`
- Значение
  - Текущее состояние объекта
  - $\sim$ состояние памяти
  - *Изменяемость* (mutability) зависит от типа

Имена и ссылки

Имя в коде $\longrightarrow$ Объект в памяти

Имена и ссылки

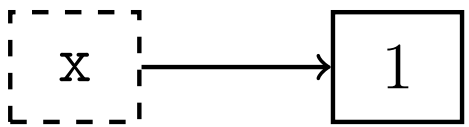
Имя в коде $\longrightarrow$ Объект в памяти

```
x = 1
```

Имена и ссылки

Имя в коде $\longrightarrow$ Объект в памяти

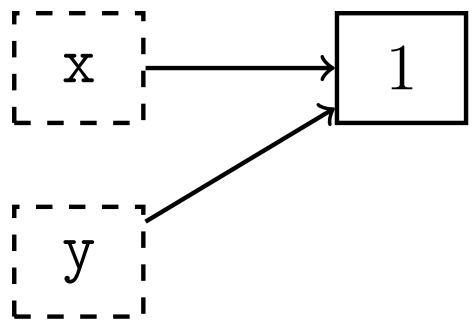
```
x = 1
```



Имена и ссылки

Имя в коде $\longrightarrow$ Объект в памяти

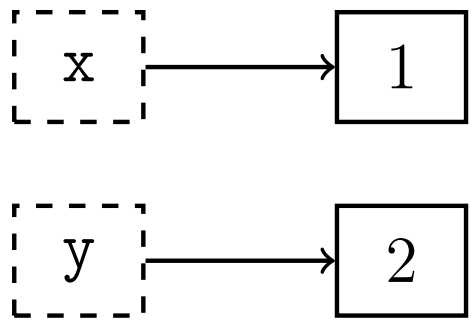
```
x = 1
y = x
```



Имена и ссылки

Имя в коде $\longrightarrow$ Объект в памяти

```
x = 1
y = x
y = 2
```



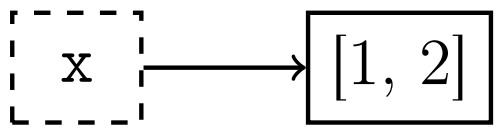
Изменяемые объекты

Значение может меняться

Изменяемые объекты

Значение может меняться

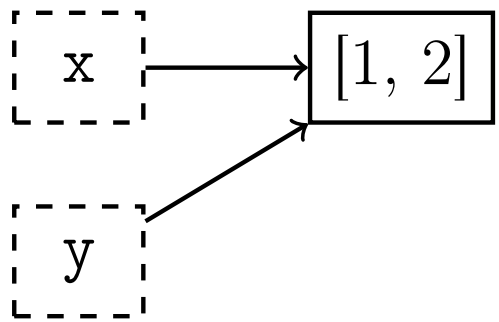
```
x = [1, 2]
```



Изменяемые объекты

Значение может меняться

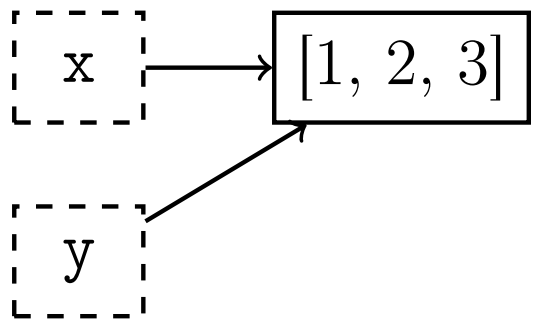
```
x = [1, 2]
y = x
```



Изменяемые объекты

Значение может меняться

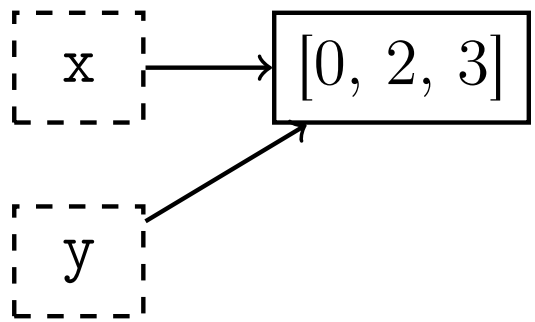
```
x = [1, 2]
y = x
y += [3]
```



Изменяемые объекты

Значение может меняться

```
x = [1, 2]
y = x
y += [3]
x[0] = 0
```



Неизменяемые объекты

Неизменяемые объекты

Некоторые операции недопустимы

Неизменяемые объекты

Некоторые операции недопустимы

```
>>> x = (1, 2)
```

Неизменяемые объекты

Некоторые операции недопустимы

```
>>> x = (1, 2)
>>> x[0] = 0
TypeError: 'tuple' object does not support item assignment
```

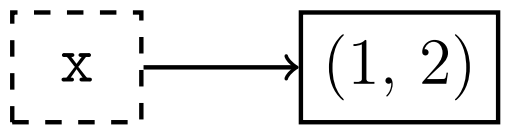
Неизменяемые объекты

Некоторые операции создают новый объект

Неизменяемые объекты

Некоторые операции создают новый объект

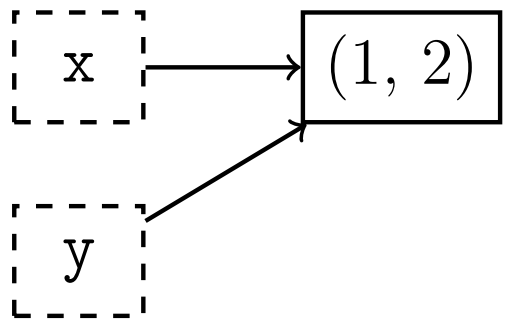
```
x = (1, 2)
```



Неизменяемые объекты

Некоторые операции создают новый объект

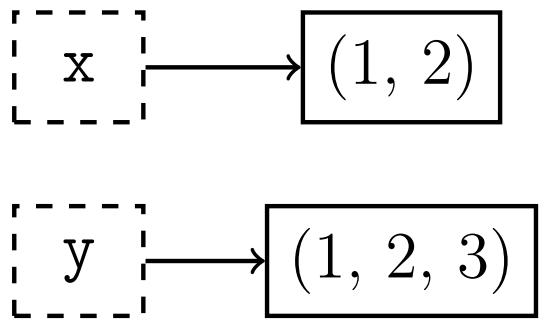
```
x = (1, 2)
y = x
```



Неизменяемые объекты

Некоторые операции создают новый объект

```
x = (1, 2)
y = x
y += (3,)
```



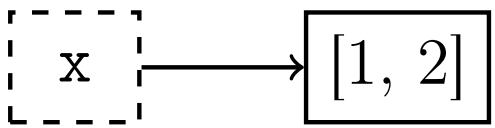
Контейнеры и изменяемость

Контейнеры — объекты со ссылками на другие объекты.
Например: списки, словари.

Контейнеры и изменяемость

Контейнеры — объекты со ссылками на другие объекты.
Например: списки, словари.

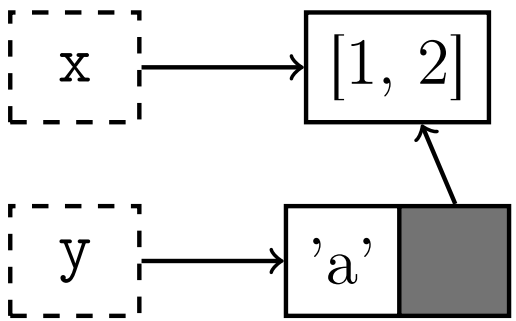
```
x = [1, 2]
```



Контейнеры и изменяемость

Контейнеры — объекты со ссылками на другие объекты.
Например: списки, словари.

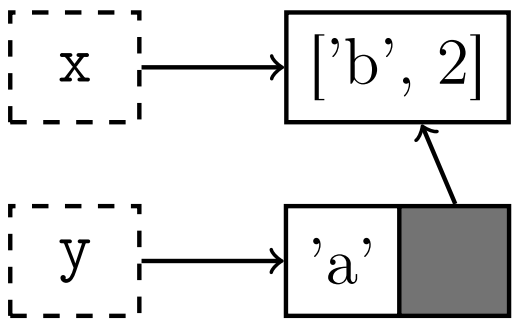
```
x = [1, 2]
y = ('a', x)
```



Контейнеры и изменяемость

Контейнеры — объекты со ссылками на другие объекты.
Например: списки, словари.

```
x = [1, 2]
y = ('a', x)
x[0] = 'b'
```



Аргументы функций

Аргументы функций

```
def f(l):
    l += ('a', 'b')
    return l

>>> x = [1, 2]
>>> f(x)
[1, 2, 'a', 'b']
>>> x
[1, 2, 'a', 'b']
>>> y = (1, 2)
>>> f(y)
(1, 2, 'a', 'b')
>>> y
(1, 2)
```

Изменяемые и неизменяемые типы

Изменяемые

`list`

`dict`

`set`

Неизменяемые

`int`

`float`

`complex`

`bool`

`str`

`tuple`

`frozenset`

Конкатенация строк

Первый вариант.

```
s = ''
for a in text:
    s += a
```

Как долго это будет работать?

Конкатенация строк

Первый вариант.

```
s = ''
for a in text:
    s += a
```

Как долго это будет работать?

Второй вариант.

```
parts = []
for a in text:
    parts.append(a)
s = ''.join(parts)
```

Конкатенация строк

Первый вариант.

```
s = ''
for a in text:
    s += a
```

Как долго это будет работать?

Второй вариант.

```
parts = []
for a in text:
    parts.append(a)
s = ''.join(parts)
```

Второй вариант быстрее первого*

* Может быть неверно в CPython.

Копирование

Копирование

- Неизменяемые объекты
  Присваивание: `y = x`

Копирование

- Неизменяемые объекты
  Присваивание: `y = x`

- Изменяемые объекты
  Модуль `copy`: `copy`, `deepcopy`.

Словари и множества

`dict` и `set`

Ключами могут быть только хэшируемые объекты.

Хэш

Объекты $\longleftrightarrow$ Целые числа

Хэш

Объекты $\longleftrightarrow$ Целые числа

`hash(...)` — хэш объекта.

Хэш

Объекты $\longleftrightarrow$ Целые числа

`hash(...)` — хэш объекта.

Хэш объекта не должен меняться.

Хэш и сравнение

Должно быть выполнено

Если `x == y`, то `hash(x) == hash(y)`

Хэш и сравнение

Должно быть выполнено

Если `x == y`, то `hash(x) == hash(y)`

`list`, `dict`, `set` — нехэшируемые

```
>>> x = [1]
>>> y = [1, 2]
>>> x == y
False
>>> x += [2]
>>> x == y
True
```

Пользовательские классы

```
>>> class C():
        pass

>>> c = C()
>>> d = {c: 'a'}
>>> c.x = 1
```

Сравнение и хэш по умолчанию

`x == y` эквивалентно `id(x) == id(y)`

`hash(x)` определяется по `id(x)`

Сравнение и хэш по умолчанию

`x == y` эквивалентно `id(x) == id(y)`
`hash(x)` определяется по `id(x)`

```
>>> x = C()
>>> y = C()
>>> x == y
False
```

Сравнение и хэш по умолчанию

`x == y` эквивалентно `id(x) == id(y)`
`hash(x)` определяется по `id(x)`

```
>>> x = C()
>>> y = C()
>>> x == y
False
>>> hash(x)
8742768675019
>>> x.a = 1
>>> hash(x)
8742768675019
```

Синглтоны

Может существовать только один такой объект

Синглтоны

Может существовать только один такой объект

Пример: `None`

Синглтоны

Может существовать только один такой объект

Пример: `None`

```
>>> x = None
>>> y = None
>>> x is y
True
```

Сборщик мусора

Сборщик мусора

- Счетчики ссылок

Сборщик мусора

- Счетчики ссылок
- Удаление по достижении счетчика 0 . . .

Сборщик мусора

- Счетчики ссылок
- Удаление по достижении счетчика 0 . . .
- . . . , возможно, не сразу

Сборщик мусора

- Счетчики ссылок
- Удаление по достижении счетчика 0 . . .
- . . . , возможно, не сразу

Менеджеры контекстов

```
with open('path/to/file') as f:
    ...
```

Сборщик мусора

- Счетчики ссылок
- Удаление по достижении счетчика 0 . . .
- . . . , возможно, не сразу

  Менеджеры контекстов

```
with open('path/to/file') as f:
    ...
```

- Циклические ссылки — OK

Байт-код

Выполнение скрипта:

- Компиляция в байт-код (.pyc)
- Исполнение

Модуль `dis` (CPython)

Библиотеки

Внешние библиотеки

PyPI — the Python Package Index.

Внешние библиотеки

PyPI — the Python Package Index.

`pip` — утилита для установки.

```
$ pip install some_package
$ pip install --upgrade other_package
```

Тестирование

Юнит-тесты

Общая идея

- Разбивать код на независимые части (юниты)
- Тестировать каждую часть отдельно

Юнит-тесты

Общая идея

- Разбивать код на независимые части (юниты)
- Тестировать каждую часть отдельно

Преимущества

- Нужно меньше тестов
- Проще отлаживать

Тесты “вручную”

```
def factorial(n):
    current = 1
    for i in range(1, n + 1):
        current *= i
    return current
```

Тесты “вручную”

```
def factorial(n):
    current = 1
    for i in range(1, n + 1):
        current *= i
    return current

def test_1():
    return factorial(1) == 1
def test_2():
    return factorial(5) == 120

if not (test_1() and test_2()):
    print "Tests failed"
```

Модуль unittest

```
import unittest

class TestFactorial(unittest.TestCase):
    def test_1(self):
        self.assertEqual(factorial(1), 1)
    def test_2(self):
        self.assertEqual(factorial(5), 120)

unittest.main()
```

(тесты должны называться `test*`)

Возможности unittest

- Отчеты
  (сколько сломалось, что сломалось)
- В чем проблема
  `assertTrue()`, `assertEqual()`
  `assertIn()`, `...`
- Поиск тестов в директории

Проверка исключений

```
class TestFactorial(unittest.TestCase):
    ...
    def test_float():
        with self.assertRaises(TypeError):
            factorial(2.5)
```

Библиотека pytest

```
def test_1():
    assert factorial(1) == 1

def test_2():
    assert factorial(5) == 120
```

```
$ py.test <filename>
```

(тесты должны называться `test*`)

Проверка кода

assert

Проверка корректности “на лету”

```
def probability_of_smth(...):
    ...
    result = ...

    assert 0 <= result <= 1, \
        "Probability must be in [0, 1]"
    return result
```

assert

```
>>> assert True
>>> assert False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

assert

```
>>> assert True
>>> assert False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError

>>> assert False, "<description>"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: <description>
```

Устройство assert

```
assert CONDITION, TEXT
```

эквивалентно

```
if not CONDITION:
    raise AssertionError(TEXT)
```

Использование assert

- В корректной программе не срабатывают
- Обычно не используется для проверки аргументов

Использование assert

- В корректной программе не срабатывают
- Обычно не используется для проверки аргументов
- Отражают инварианты программы

Использование assert

- В корректной программе не срабатывают
- Обычно не используется для проверки аргументов
- Отражают инварианты программы
- Есть опция, отключающая их проверку для ускорения работы

Утилиты для проверки

(Проверка кода без выполнения)

- pep8.py
- PyChecker
- PyFlakes
- pylint
- (PyCharm)