

# Лекция 5

## Модель памяти. Обработка аргументов.

16 марта 2017 г.

# 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)

# 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

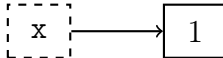
x = 1



# Имена и ссылки

Имя в коде  $\longrightarrow$  Объект в памяти

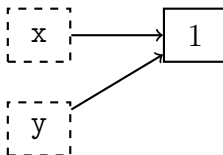
`x = 1`



# Имена и ссылки

Имя в коде  $\longrightarrow$  Объект в памяти

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



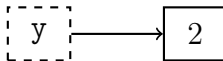
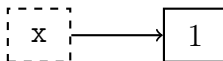
# Имена и ссылки

Имя в коде  $\longrightarrow$  Объект в памяти

x = 1

y = x

y = 2



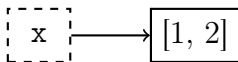
# Изменяемые объекты

Значение может меняться

# Изменяемые объекты

Значение может меняться

`x = [1, 2]`



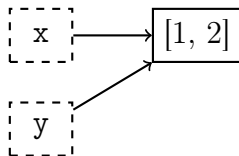


# Изменяемые объекты

Значение может меняться

`x = [1, 2]`

`y = x`



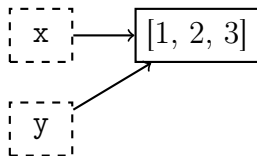
# Изменяемые объекты

Значение может меняться

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

```
y = x
```

```
y += [3]
```



# Изменяемые объекты

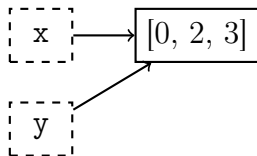
Значение может меняться

```
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

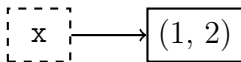
Некоторые операции создают новый объект



# Неизменяемые объекты

Некоторые операции создают новый объект

$x = (1, 2)$

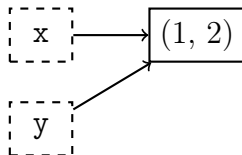


# Неизменяемые объекты

Некоторые операции создают новый объект

$x = (1, 2)$

$y = x$



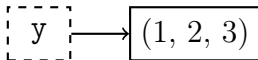
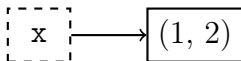
# Неизменяемые объекты

Некоторые операции создают новый объект

`x = (1, 2)`

`y = x`

`y += (3,)`



# Контейнеры и изменяемость

*Контейнеры* — объекты со ссылками на другие объекты.

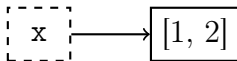
Например: списки, словари.

# Контейнеры и изменяемость

*Контейнеры* — объекты со ссылками на другие объекты.

Например: списки, словари.

`x = [1, 2]`



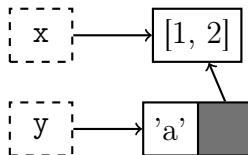
# Контейнеры и изменяемость

*Контейнеры* — объекты со ссылками на другие объекты.

Например: списки, словари.

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

```
y = ('a', x)
```

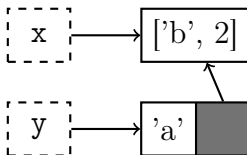


# Контейнеры и изменяемость

*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 `сору`: `сору`, `деерсору`.



# 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  $\text{hash}(x) == \text{hash}(y)$

# 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 — ОК

# Байт-код

Выполнение скрипта:

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

Модуль `dis` (CPython)

# Расширенная работа с аргументами

# Аргументы функций

Обычная функция

```
def complex(real, imag):  
    return real + 1j * imag  
print complex(1, 2)
```

```
(1+2j)
```

# Аргументы функций

Обычная функция

```
def complex(real, imag):  
    return real + 1j * imag  
print complex(1, 2)
```

(1+2j)

Аргументы по умолчанию

```
def complex(real, imag=0):  
    return real + 1j * imag  
print complex(1)
```

(1+0j)

# Порядковые аргументы

Распаковка аргументов

```
def complex(*args):  
    return args[0] + 1j * args[1]  
complex(1, 2)
```

(1+2j)



# Порядковые аргументы

## Распаковка аргументов

```
def complex(*args):  
    return args[0] + 1j * args[1]  
complex(1, 2)
```

```
(1+2j)
```

## Смешанные аргументы

```
def complex(real, *other):  
    print other  
    return real + 1j * other[0]  
print complex(1, 2, 3, 4)
```

```
(2, 3, 4)  
(1+2j)
```

# Запаковка порядковых аргументов

Запаковка аргументов

```
def complex(real, imag):  
    return real + 1j * imag  
args = (1, 2)  
print complex(*args)
```

(1+2j)

# Именованные аргументы

Передача аргументов по ключу

```
print complex(real=1, imag=2)  
print complex(3, imag=4)
```

```
(1+2j)
```

```
(3+4j)
```

# Именованные аргументы

Распаковка по ключу

```
def complex(**kwargs):  
    print kwargs  
    return (kwargs['real'] +  
            1j * kwargs['imag'])  
print complex(real=1, imag=2, hi=3)
```

```
{'real': 1, 'imag': 2, 'hi': 3}  
(1+2j)
```

# Запаковка именованных аргументов

Запаковка словаря

```
kwargs = {'real': 1, 'imag': 2}  
print complex(**kwargs)
```

```
{'real': 1, 'imag': 2}  
(1+2j)
```

## Функция с любыми аргументами

```
def any_args(*args, **kwargs):  
    ...
```