

Язык программирования



Владимир Руцкий <altsysrq@gmail.com>

План лекции

Часть I

- Что такое Python?
- Зачем нужен Python и где его используют?

Часть II

- Установка Python
- Введение в Python
- Практика

Что такое Python?

- **Python** ([ˈpaɪθən] - *пайтон*) - язык программирования (далее ЯП)
- Разрабатывался с 1990 года (для сравнения: C - с 1969, C++ - с 1983)
- Оригинальный автор: **Гвидо ван Россум** (Guido van Rossum)
- <http://python.org>

factorial.py:

```
1 def factorial(n):  
2     if n <= 1:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7 print factorial(5)
```

```
C:\> python.exe factorial.py  
120
```

Python - высокоуровневый ЯП

- Может оперировать с абстрактными объектами и структурами данных, вроде файла, дерева, базы данных

Python - ориентирован на разработчика

- Программы на Python в 5-10 раз короче программ решающих те же задачи, но написанных на C++, и в 3-5 раз короче программ на Java
- Программы на Python легко читаются
- Программы на Python лаконичны
 - "синтаксический сахар"
 - большая стандартная библиотека,
 - работа с сетью, web, работа с файлами баз данных, архивами, мультипоточность, мультипроцессорность, высокоуровневые структуры данных (комплексные числа, списки, словари, множества)
- Подходит для быстрой разработки программ, прототипирования

Python - масштабируемый

- Масштабируемость кода:
 - наборы команд объединяются в **функции**
 - функции объектов объединяются в **классы**
 - наборы функций и классов объединяются в **модули** (отдельные файлы)
 - модули группируются в **пакеты** (директории с файлами модулей)
- Масштабируемость по производительности:
 - Узкие места программ можно переписать на C или C++

Python - интерпретируемый

- Программа - набор строк кода, лежащий в одном или нескольких файлах, выполняется «на лету», без предварительной компиляции
- Позволяет вносить изменения и быстро перезапускать программу
- Минус: меньшая скорость работы, по сравнению с компилируемыми языками

Интроспекция

- Программе на Python доступна вся информация о себе: список переменных, функций, классов, информация о методах классов и т.п.
- Информация о себе по большей части можно модифицировать
 - В процессе работы программы, программа может создавать новые классы и функции и изменять уже существующие

Динамическая типизация

- Нет предварительного объявления типов - тип переменной выводится в процессе выполнения

Функция может вернуть объект любого типа

$a = f(x)$

- Строгая типизация

Недопустимо: $5 + "3"$

Python - мультипарадигменный

- Поддержка ООП
 - Классы, наследование, полиморфизм, условная инкапсуляция
- Поддержка функционального программирования
 - Лямбда-выражения, list comprehension
- Python вобрал в себя наиболее удобные возможности популярных языков программирования (ABC, Modula-3, Lisp, Tcl Smalltalk, C, Java, Icon)

Дополнительные характеристики Python

- Полностью автоматическое управление памятью
 - Сборщик мусора
- Поддержка механизма исключений

Интерпретаторы Python

- Интерпретаторы Python:
 - **CPython** (написан на C) - основная реализация
 - PyPy (написан на Python)
 - Jython (написан на Java)
 - IronPython (написан на C#)
 - и др.
- Разные интерпретаторы ориентированы на разные платформы (.Net, Java)
- Большинство интерпретаторов - под либеральной свободной лицензией
- CPython выступает в качестве стандарта

Версии Python

- Две основные ветки
 - Вторая: 2.4, 2.5, 2.6, 2.7
 - быстрее
 - больше библиотек
 - Третья: 3.1, 3.2
 - лучше синтаксис
- Каждая следующая версия немного расширяет и улучшает язык
- Внутри ветки версии обратно совместимы
- Третья версия обратно **не** совместима со второй
- Мы будем изучать **Python 2.7**, в реализации **CPython**

Библиотеки Python

- Мощная встроенная библиотека
 - Работа с Web, регулярные выражения, архивы, многозадачность, UI
- Большое количество Python-интерфейсов для популярных библиотек
 - 2D и 3D графика, OpenGL, DirectX
 - работа с базами данных, MySQL, PostgreSQL
 - работа с мультимедиа: звук, видео, изображения
 - разработка пользовательских интерфейсов, Qt, Gtk, WxWidgets

Использование Python (1/2)

- Интерактивная консоль - мощный «калькулятор»
 - работа с числами, матрицами, файлами, изображениями, статистического анализа и др.
- ЯП для небольших скриптов
 - обработать изображения, создание резервных копий
- ЯП для прототипирования
 - быстрое создание шаблона программы с UI
 - быстрая проверки работы алгоритма
- ЯП для полноценных программ
 - Gajim, BitTorrent, Dropbox, EVE Online

Использование Python (2/2)

- ЯП для web-приложений
 - много фреймворков, активно используют крупные компании, вроде Google и Яндекс
- Встраиваемый в приложения ЯП
 - Встроенная Python-консоль в которой можно оперировать с объектами приложений на языке Python
 - 3D моделирование: Blender, Maya
 - Обработка изображений: GIMP
 - Работа с ГИС данными: ESRI ArcGIS
 - Математический пакет: sagemath

Примеры использования Python

- На следующей лекции

Установка Python

- Скачиваем дистрибутив «Python 2.7.2 Windows Installer» с <http://python.org/download/>
- Устанавливаем в директорию по умолчанию (C:\Python27)
- Интерпретатор: C:\Python27\python.exe

Установка дополнительных библиотек (1/3)

- Вариант 1:
 - Скачать дистрибутив с сайта библиотеки (один .exe) и поставить
- Вариант 2:
 - Использовать встроенные менеджеры библиотек (работает не для всех библиотек)
 - Скачиваем **setuptools** с <http://pypi.python.org/pypi/setuptools> (setuptools-0.6c11.win32-py2.7.exe)
 - Устанавливаем в директорию по умолчанию
 - Дополнительные скрипты Python ставятся в **C:\Python27\Scripts** - теперь там есть **easy_install.exe**
 - C:\> cd C:\Python27\Scripts
 - C:\Python27\Scripts> **easy_install.exe pip**
 - easy_install скачивает и устанавливает pip
 - Теперь в C:\Python27\Scripts есть **pip.exe**

Установка дополнительных библиотек (2/3)

- **easy_install** и **pip** - это скрипты для удобной установки библиотек
 - работают **не** со всеми библиотеками!
- **pip** разрабатывается как замена **easy_install**
- Пример: установка библиотеки для работы с изображениями **PIL** (Python Imaging Library):

```
C:\Python27\Scripts> easy_install.exe pil
```

```
Searching for pil
```

```
Reading http://pypi.python.org/simple/pil/
```

```
...
```

```
Best match: PIL 1.1.7
```

```
...
```

```
Installed c:\python27\lib\site-packages\pil-1.1.7-py2.7-win32.egg
```

```
Processing dependencies for pil
```

```
Finished processing dependencies for pil
```

Установка дополнительных библиотек (3/3)

Пример: установка pyOpenGL

```
C:\Python27\Scripts> pip.exe install pyopengl
```

```
Downloading/unpacking pyopengl
```

```
  Downloading PyOpenGL-3.0.2a1.zip (1.4Mb): 1.4Mb downloaded
```

```
...
```

```
  warning: no previously-included files found matching 'OpenGL_accelerate'
```

```
Successfully installed pyopengl
```

```
Cleaning up...
```

Введение в Python

- Официальная документация (англ.):
<http://doc.python.org/>
- **Перевод учебного пособия оттуда для Python 2.6:**
http://ru.wikibooks.org/wiki/Учебник_Python_2.6
(по нему построено введение в Python в этой лекции)
- Книги на русском языке... мало, если будете смотреть обязательно обратите внимание на версию Python

Выполнение программ на Python

- Способы выполнения программ:

- интерактивное выполнение

```
C:\> python.exe
```

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print "Hello, world!"
```

```
Hello, world!
```

```
>>>
```

- выполнение файла со скриптом

hello.py:

```
1 print "Hello, world!"
```

```
C:> python.exe hello.py  
Hello, world!
```

Интерактивная консоль Python

- Запустите **python.exe**

```
C:\> python.exe
```

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

- «>>>» - приветствие, интерпретатор ожидает ввода команды

Hello, world!

```
1 >>> print "Hello, world!"  
2 Hello, world!  
3 >>>
```

Вычисления выражений

```
1 >>> 2 + 2
2 4
3 >>> # Это комментарий
4 ... 6 + 4
5 10
6 >>> 2 + 2 # а вот комментарий на одной строке с кодом
7 4
8 >>> (50 - 5 * 6) / 4
9 5
10 >>> # Деление целых чисел возвращает округленное к минимальному значение:
11 ... 7/3
12 2
13 >>> 7/-3
14 -3
15 >>> # Результат последнего вычисления хранится в переменной '_'
16 ... _
17 -3
18 >>> # В интерактивной консоли результат вычисления выражения пишется на экран
19 ... # Чтобы вывести на экран в скрипте можно использовать оператор 'print'
20 ... print 5 * 6
21 30
22 >>>
```

Переменные

```
1 >>> # Присваивание делается знаком '='
2 ... a = 30
3 >>> a / 3
4 10
5 >>> # Значение может быть присвоено нескольким переменным одновременно
6 ... a = b = c = 0
7 >>> a
8 0
9 >>> b
10 0
11 >>> c
12 0
13 >>> # Перед использованием переменной в выражении она должна быть определена
14 ... d
15 Traceback (most recent call last):
16   File "<stdin>", line 2, in <module>
17 NameError: name 'd' is not defined
18 >>>
```

Числовые типы

```
1 >>> # Есть поддержка чисел с плавающей точкой, длинных чисел, комплексных чисел...
2 ... 3 * 3.75 / 1.5
3 7.5
4 >>> i = 1j # мнимая единица
5 >>> i
6 1j
7 >>> i ** 2 # возведение в степень
8 (-1+0j)
9 >>> i.real
10 0.0
11 >>> i.imag
12 1.0
13 >>> (1 + 2j) / (1 + 1j)
14 (1.5+0.5j)
15 >>> 2 ** 100
16 1267650600228229401496703205376L
17 >>> 2 ** 200
18 1606938044258990275541962092341162602522202993782792835301376L
19 >>> # Есть встроенные переменные логического типа:
20 ... a = (1 == 2)
21 >>> print a
22 False
23 >>> not a
24 True
25 >>> x = 30
26 >>> 1 < x < 100
27 True
28 >>>
```

Строки

```
1 >>> # Строки можно задавать следующим образом:
2 ... 'spam eggs'
3 'spam eggs'
4 >>> 'doesn\'t'
5 "doesn't"
6 >>> "doesn't"
7 "doesn't"
8 >>> '"Yes," he said.'
9 '"Yes," he said.'
10 >>> "\"Yes,\" he said."
11 '"Yes," he said.'
12 >>> '"Isn\'t," she said.'
13 '"Isn\'t," she said.'
14 >>> "first word \
15 ... second word"
16 'first word second word'
17 >>> print "first line \nsecondline"
18 first line
19 secondline
20 >>> r"line with \n in middle"
21 'line with \n in middle'
22 >>> """Multiline with ' or "
23 ... Yes.
24 ... """
25 'Multiline with \' or "\nYes.\n'
26 >>>
```

Конкатенация строк

```
1 >>> # Конкатенация строк
2 ... word = 'Help' + 'A'
3 >>> word
4 'HelpA'
5 >>> '<' + word * 5 + '>'
6 '<HelpAHelpAHelpAHelpAHelpA>'
7 >>> 'first' 'second'
8 'firstsecond'
9 >>> ('first') 'second'
10 File "<stdin>", line 1
11     ('first') 'second'
12             ^
13 SyntaxError: invalid syntax
14 >>> # Строки --- неизменяемые
15 ... word[0] = 'x' # word[0] --- обращение к первому символу строки
16 Traceback (most recent call last):
17   File "<stdin>", line 1, in <module>
18     TypeError: 'str' object does not support item assignment
19 >>>
```

Индексация последовательностей (1/2)

```
1 >>> # Индексация в последовательностях (например строках):
2 ... # +---+---+---+---+---+
3 ... # | H | e | l | p | A |
4 ... # +---+---+---+---+---+
5 ... #   0   1   2   3   4   5
6 ... #  -5  -4  -3  -2  -1
7 ... #  v[i] - i-й элемент
8 ... #  v[i:j] - подпоследовательность начиная с i-го включительно,
9 ... #                и до j (не включительно) --- полуинтервал [i, j)
10 ... word = 'He lpA'
11 >>> word[4]
12 'A'
13 >>> word[0:2]
14 'He'
15 >>> word[2:4]
16 'lp'
17 >>> word[:2]      # Первые два символа
18 'He'
19 >>> word[2:]      # Всё, исключая первые два символа
20 'lpA'
21 >>> word[1:100]
22 'e lpA'
23 >>> word[10:]
24 ''
25 >>>
```

Индексация последовательностей (2/2)

```
1 >>> # Индексация в последовательностях (например строках):
2 ... # +---+---+---+---+---+
3 ... # | H | e | l | p | A |
4 ... # +---+---+---+---+---+
5 ... #   0   1   2   3   4   5
6 ... #  -5  -4  -3  -2  -1
7 ... #  v[i] - i-й элемент
8 ... #  v[i:j] - подпоследовательность начиная с i-го включительно,
9 ... #                и до j (не включительно) --- полуинтервал [i, j)
10 ... word = 'HeIpA'
11 >>> word[-1]      # Последний символ
12 'A'
13 >>> word[-2]      # Предпоследний символ
14 'p'
15 >>> word[-2:]      # Последние два символа
16 'pA'
17 >>> word[:-2]      # Всё, кроме последних двух символов
18 'Hel'
19 >>> word[-100:]
20 'HeIpA'
21 >>> word[-10]      # ошибка
22 Traceback (most recent call last):
23   File "<stdin>", line 1, in <module>
24 IndexError: string index out of range
25 >>>
```


Списки

```
1 >>> # Список --- встроенный в язык тип данных
2 ... a = ['spam', 'eggs', 1234, 30]
3 >>> # Список --- это упорядоченная последовательность элементов.
4 ... # Для списков действуют такие же правила индексации как для строк
5 ... a[0]
6 'spam'
7 >>> a[1:3]
8 ['eggs', 1234]
9 >>> # Но списки, в отличие от строк, изменяемые
10 ... a[1] = 3030
11 >>> a
12 ['spam', 3030, 1234, 30]
13 >>> a[:2] + a[3:] * 2
14 ['spam', 3030, 30, 30]
15 >>> [1, 2] + [3, 4]
16 [1, 2, 3, 4]
17 >>> [1, 2] * 4
18 [1, 2, 1, 2, 1, 2, 1, 2]
19 >>>
```

Функция range()

```
1 >>> # Функция range([start,] stop[, step]) создаёт список из натуральных чисел
2 ... a = range(5)
3 >>> a
4 [0, 1, 2, 3, 4]
5 >>> # Функции, классы и модули в Python имеют способы встроенного документирования
6 ... help(range)
7 Help on built-in function range in module __builtin__:
8
9 range(...)
10     range([start,] stop[, step]) -> list of integers
11
12     Return a list containing an arithmetic progression of integers.
13     range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
14     When step is given, it specifies the increment (or decrement).
15     For example, range(4) returns [0, 1, 2, 3]. The end point is omitted!
16     These are exactly the valid indices for a list of 4 elements.
17
18 >>> range(10, 20, 2)
19 [10, 12, 14, 16, 18]
20 >>>
```

Модификация списков (1/2)

```
1 >>> a = range(10)
2 >>> a
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> # Заменяем некоторые элементы:
5 ... a[1:3] = [20, 30]
6 >>> a
7 [0, 20, 30, 3, 4, 5, 6, 7, 8, 9]
8 >>> # Удалим:
9 ... a[1:3] = []
10 >>> a
11 [0, 3, 4, 5, 6, 7, 8, 9]
12 >>> # Вставим:
13 ... a[1:1] = ['bletch', 'xyzy']
14 >>> a
15 [0, 'bletch', 'xyzy', 3, 4, 5, 6, 7, 8, 9]
16 >>> # Вставим (копию) самого себя в начало
17 ... a[:0] = a
18 >>> a
19 [0, 'bletch', 'xyzy', 3, 4, 5, 6, 7, 8, 9, 0, 'bletch', 'xyzy', 3, 4, 5, 6, 7, 8, 9]
20 >>> # Очистка списка: замена всех значений пустым списком
21 ... a[:] = []
22 >>> a
23 []
24 >>> # Также переменную можно удалить оператором 'del'
25 ... del a
26 >>> a
27 Traceback (most recent call last):
28   File "<stdin>", line 1, in <module>
29 NameError: name 'a' is not defined
30 >>>
```

Модификация списков (2/2)

```
1 >>> a = range(10)
2 >>> a
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> # Получить длину последовательности:
5 ... len(a)
6 10
7 >>> len('some string')
8 11
9 >>> # Списки, как и почти все контейнеры Python, могут хранить любые объекты
10 ... a[1] = ['another', 'list']
11 >>> a
12 [0, ['another', 'list'], 2, 3, 4, 5, 6, 7, 8, 9]
13 >>> a[1][0]
14 'another'
15 >>> b = range(3)
16 >>> b
17 [0, 1, 2]
18 >>> # Некоторые операции со списками:
19 ... # добавить в конец
20 ... b.append(10)
21 >>> b
22 [0, 1, 2, 10]
23 >>> # расширить список элементами из другого контейнера
24 ... b.extend(a)
25 >>> b
26 [0, 1, 2, 10, 0, ['another', 'list'], 2, 3, 4, 5, 6, 7, 8, 9]
27 >>> # развернуть список
28 ... b.reverse()
29 >>> b
30 [9, 8, 7, 6, 5, 4, 3, 2, ['another', 'list'], 0, 10, 2, 1, 0]
31 >>> # отсортировать список
32 ... b.sort()
33 >>> b
34 [0, 0, 1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, ['another', 'list']]
35 >>>
```

Оператор while

```
1 >>> # Ряд Фибоначчи:
2 ... # сумма двух элементов определяет следующий элемент
3 ... a, b = 0, 1 # множественное присваивание: a = 0, b = 1
4 >>> while b < 10:
5 ...     print b
6 ...     a, b = b, a + b
7 ...
8 1
9 1
10 2
11 3
12 5
13 8
14 >>> # while УСЛОВИЕ:
15 ... #     команды
16 ... # Блоки в Python определяются отступом!
17 ... pass # оператор "ничего не делать"
18 >>> while True: # бесконечный цикл, можно прервать по Ctrl+C
19 ...     pass
20 ...
21 ^CTraceback (most recent call last):
22   File "<stdin>", line 1, in <module>
23 KeyboardInterrupt
24 >>>
```

Оператор if

```
1 >>> # raw_input() можно использовать для чтения ввода с клавиатуры
2 ... s = raw_input("Введите, пожалуйста, целое число: ")
3 Введите, пожалуйста, целое число: 42
4 >>> s
5 '42'
6 >>> # s - строка, переведём её к целочисленному типу
7 ... x = int(s)
8 >>> x
9 42
10 >>> # Условный оператор
11 ... if x < 0:
12 ...     print 'Отрицательное значение'
13 ... elif x == 0:
14 ...     print 'Ноль'
15 ... elif x > 0:
16 ...     print 'Больше нуля'
17 ... else:
18 ...     print 'Такого быть не может!'
19 ...
20 Больше нуля
21 >>>
```

Оператор for

```
1 >>> # Оператор for используется для выполнения какого-то действия
2 ... # для всех элементов последовательности.
3 ... # for ПЕРЕМЕННЫЕ in ИТЕРИРУЕМОЕ-ВЫРАЖЕНИЕ:
4 ... #     команды
5 ... a = ['cat', 'window', 'defenestrated']
6 >>> for x in a:
7 ...     print x, len(x)
8 ...
9 cat 3
10 window 6
11 defenestrated 12
12 >>> # Изменять содержимое контейнера по которому итерируется цикл
13 ... # часто небезопасно
14 ... for x in a[:]: # создадим копию списка
15 ...     if len(x) > 6:
16 ...         a.insert(0, x)
17 ...
18 >>> a
19 ['defenestrated', 'cat', 'window', 'defenestrated']
20 >>>
```

Операторы break, continue

```
1 >>> # В циклах можно использовать операторы break и continue,  
2 ... # а также конструкцию else  
3 ... for n in range(2, 10):  
4 ...     for x in range(2, n):  
5 ...         if n % x == 0:  
6 ...             print n, 'равно', x, '*', n//x  
7 ...             break  
8 ...     else:  
9 ...         # циклу не удалось найти множитель  
10 ...         print n, '- простое число'  
11 ...  
12 2 - простое число  
13 3 - простое число  
14 4 равно 2 * 2  
15 5 - простое число  
16 6 равно 2 * 3  
17 7 - простое число  
18 8 равно 2 * 4  
19 9 равно 3 * 3  
20 >>>
```


Словари

```
1 >>> # Словарь --- контейнер, хранящий пары (ключ, значение), и позволяющий
2 ... # быстро находить по ключу соответствующее значение
3 ... tel = {'jack': 4098, 'sape': 4139}
4 >>> tel['jack']
5 4098
6 >>> tel['guido'] = 4127
7 >>> tel
8 {'sape': 4139, 'jack': 4098, 'guido': 4127}
9 >>> del tel['sape']
10 >>> tel
11 {'jack': 4098, 'guido': 4127}
12 >>> tel.keys()
13 ['jack', 'guido']
14 >>> tel.values()
15 [4098, 4127]
16 >>> tel.items()
17 [('jack', 4098), ('guido', 4127)]
18 >>> for name, phone in tel.items():
19 ...     print name, 'has phone number', phone
20 ...
21 jack has phone number 4098
22 guido has phone number 4127
23 >>> 'guido' in tel
24 True
25 >>> 'sam' in tel
26 False
27 >>>
```

Функции

```
1 >>> # Определение функции
2 ... def fib(n):      # вывести числа Фибоначчи меньше (вплоть до) n
3 ...     """Выводит ряд Фибоначчи, ограниченный n."""
4 ...     a, b = 0, 1
5 ...     while b < n:
6 ...         print b,
7 ...         a, b = b, a+b
8 ...
9 >>> # Теперь вызовем определенную нами функцию:
10 ... fib(2000)
11 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
12 >>> help(fib)
13 Help on function fib in module __main__:
14
15 fib(n)
16     Выводит ряд Фибоначчи, ограниченный n.
17
18 >>>
```

Функции с аргументами по умолчанию (1/2)

```
1 >>> def f(a, b, c=5, d=[1,3]):
2 ...     print a, b, c, d
3 ...
4 >>> f(1)
5 Traceback (most recent call last):
6   File "<stdin>", line 1, in <module>
7   TypeError: f() takes at least 2 arguments (1 given)
8 >>> f(1, 2)
9 1 2 5 [1, 3]
10 >>> f(1, 2, 'c')
11 1 2 c [1, 3]
12 >>> f(1, 2, d='D', c='C')
13 1 2 C D
14 >>> def f(a, b, *args):
15 ...     print a, b,
16 ...     for x in args:
17 ...         print x,
18 ...
19 >>> f(1, 2)
20 1 2
21 >>> f(1, 2, 3, 4, 5, 6)
22 1 2 3 4 5 6
23 >>>
```

Функции с аргументами по умолчанию (2/2)

```
1 >>> # Значения по умолчанию вычисляются только один раз!
2 ... def f(a, L=[]):
3 ...     L.append(a)
4 ...     return L
5 ...
6 >>> f(1)
7 [1]
8 >>> f(2)
9 [1, 2]
10 >>> f(3)
11 [1, 2, 3]
12 >>> def f(a, L=None):
13 ...     if L is None:
14 ...         L = []
15 ...     L.append(a)
16 ...     return L
17 ...
18 >>> f(1)
19 [1]
20 >>> f(2)
21 [2]
22 >>>
```

Неименованные аргументы

```
1 >>> # Есть возможность для получения произвольного количества аргументов
2 ... def f(a, *args, **kwargs):
3 ...     print args
4 ...     print kwargs
5 ...
6 >>> f(1, 'a', 'b', 3, k1=1, k2=2, k3=3)
7 ('a', 'b', 3)
8 {'k3': 3, 'k2': 2, 'k1': 1}
9 >>> # Кортеж --- контейнер аналогичный списку, но не изменяемый
10 ... a = (1, 2, 3)
11 >>> a
12 (1, 2, 3)
13 >>> a[0:2]
14 (1, 2)
15 >>> a[1] = 2
16 Traceback (most recent call last):
17   File "<stdin>", line 1, in <module>
18 TypeError: 'tuple' object does not support item assignment
19 >>> b = (1, [])
20 >>> # Кортеж содержит список объектов, и этот список изменить нельзя
21 ... # Но можно менять сами объекты
22 ... b[1].extend([30, 40])
23 >>> b
24 (1, [30, 40])
25 >>>
```

Объекты Python (1/3)

```
1 >>> # В Python все хранимые в памяти вещи являются объектами
2 ... # Тип объекта можно получить с помощью функции `type`
3 ... type(1)
4 <type 'int'>
5 >>> type(1+2j)
6 <type 'complex'>
7 >>> type('aaa')
8 <type 'str'>
9 >>> # Каждый объект имеет идентификатор, который можно получить
10 ... # с помощью функции `id`
11 ... id(1)
12 17528760
13 >>> id(1+2j)
14 17674480
15 >>> id(2-1)
16 17528760
17 >>> def f():
18 ...     pass
19 ...
20 >>> type(f)
21 <type 'function'>
22 >>> id(f)
23 18917688
24 >>>
```

Объекты Python (2/3)

```
1 >>> # Оператор присваивания '=' на самом деле связывает имя и определённый объект
2 ... a = [1, 2, 3] # связываем имя 'a' с объектом списком
3 >>> id(a) # ID-объекта, на который ссылается имя 'a'
4 18908584
5 >>> b = a # связываем имя 'b' с объектом, на который ссылается 'a'
6 >>> id(b) # ID-объекта, на который ссылается 'b', такое же как и у 'a'!
7 18908584
8 >>> a is b # проверяет, являются ли два объекта одним и тем же
9 True
10 >>> a is [1, 2, 3]
11 False
12 >>> # Так как 'a' и 'b' ссылаются на один объект, то при изменении его через
13 ... # одно имя, он окажется изменённым и по другому имени
14 ... a[0] = 'steel'
15 >>> a
16 ['steel', 2, 3]
17 >>> b
18 ['steel', 2, 3]
19 >>> # Числа, строки и некоторые другие объекты являются неизменяемыми,
20 ... # при попытке их изменения создаётся новый объект.
21 ... a = 1
22 >>> id(a)
23 17528760
24 >>> b = a
25 >>> a += 1
26 >>> id(a)
27 17528736
28 >>> a is b
29 False
30 >>>
```

Объекты Python (3/3)

```
1 >>> def f(a):
2 ...     a.append(30)
3 ...
4 >>> # Аргументы в функцию передаются по ссылке
5 ... a = []
6 >>> f(a)
7 >>> a
8 [30]
9 >>> def g(b):
10 ...     b += 1 # перезапишет на что ссылается локальный `b`
11 ...     print b
12 ...
13 >>> a = 10
14 >>> g(a)
15 11
16 >>> a
17 10
18 >>> # Имя функции --- такая же ссылка на объект, как и имя переменной
19 ... F = g
20 >>> F(10)
21 11
22 >>> F is g
23 True
24 >>>
```