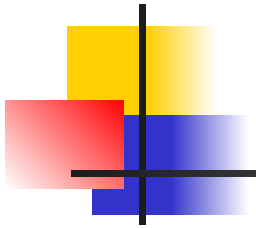
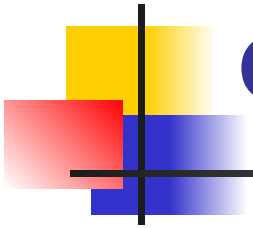


Лекция 1 «Основы программирования на Python»

К.т.н. доц. Ляшенко Алексей Сергеевич



Философия Python

Красивое лучше уродливого.

Явное лучше неявного.

Простое лучше сложного.

Сложное лучше усложнённого.

Последовательное лучше вложенного.

Разрежённое лучше, чем плотное.

Удобочитаемость существенна.

Частные случаи не настолько существенны, чтобы нарушать правила.

Однако практичность важнее регулярности.

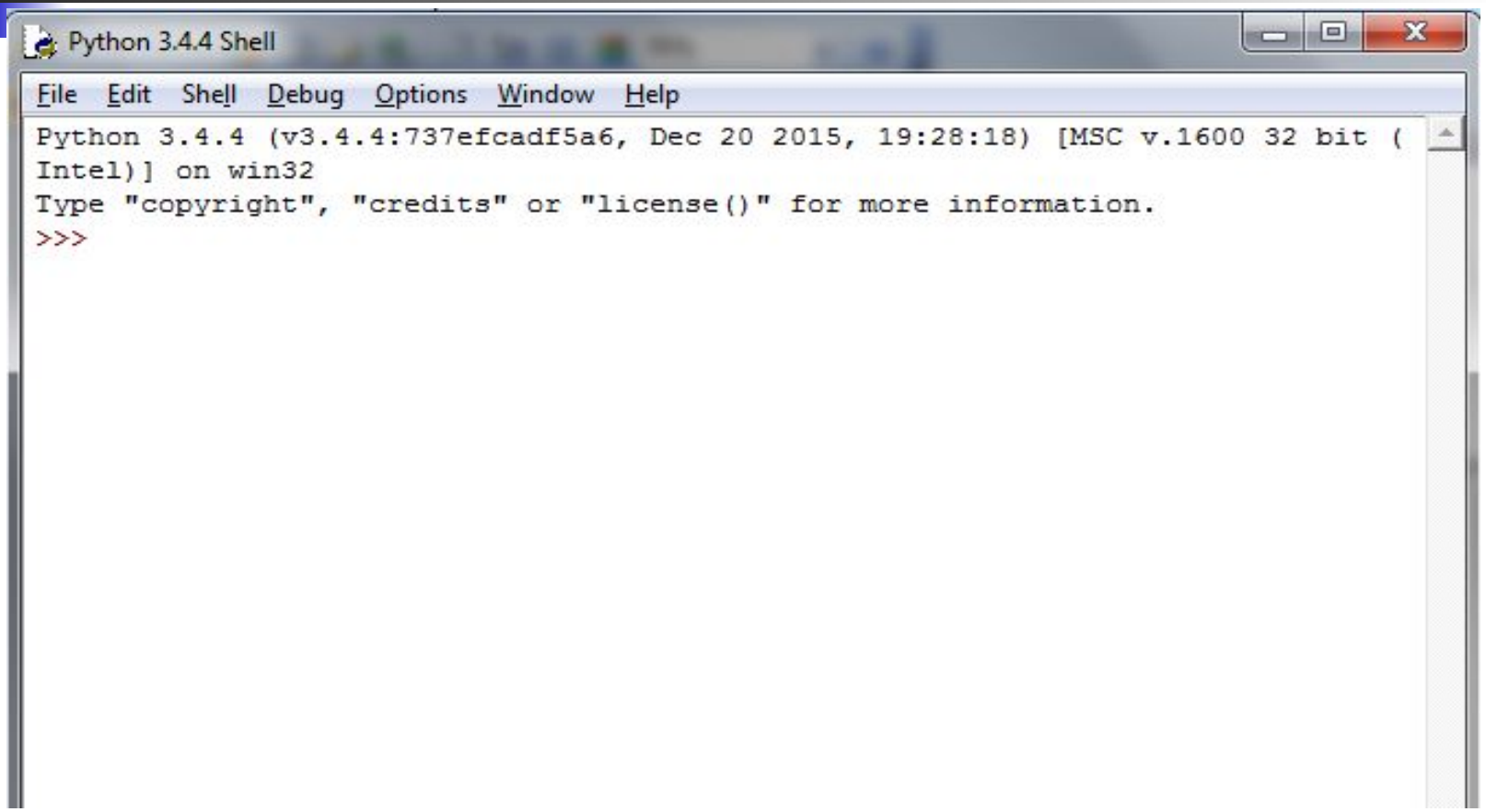
Ошибки никогда не должны умалчиваться.

Если явно не указано — умалчивать.

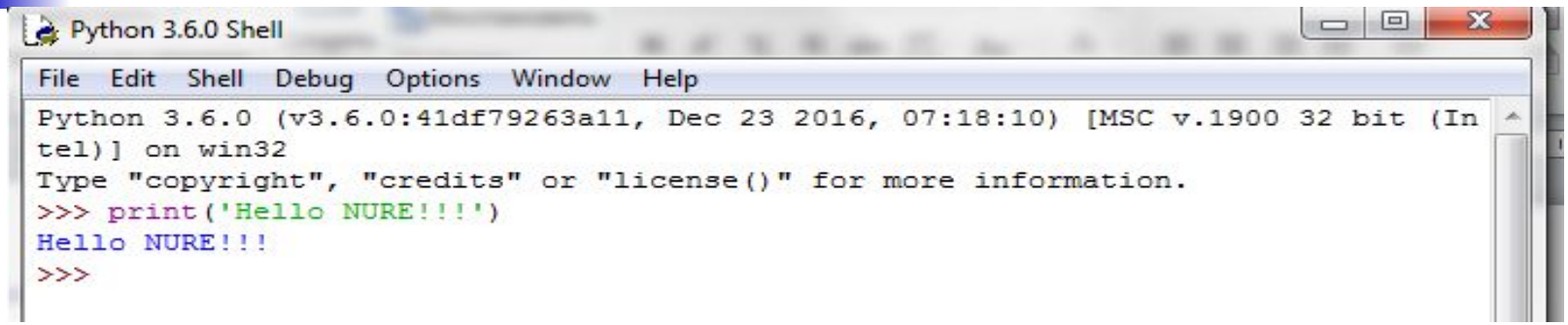
В случае неоднозначности сопротивляйтесь искушению угадать.

Должен существовать один — и, желательно, только один — очевидный способ.
Хотя он может быть с первого взгляда неочевиден если ты не голландец (намёк на Гвидо ван Россума)
Сейчас — лучше, чем никогда.
Но никогда — часто бывает лучше, чем прямо сейчас.
Если реализацию идеи тяжело объяснить, она плоха.
Если реализацию идеи легко объяснить, она может быть хороша.
Пространства имён — великолепная идея, их должно быть много.

Python. IDLE


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>

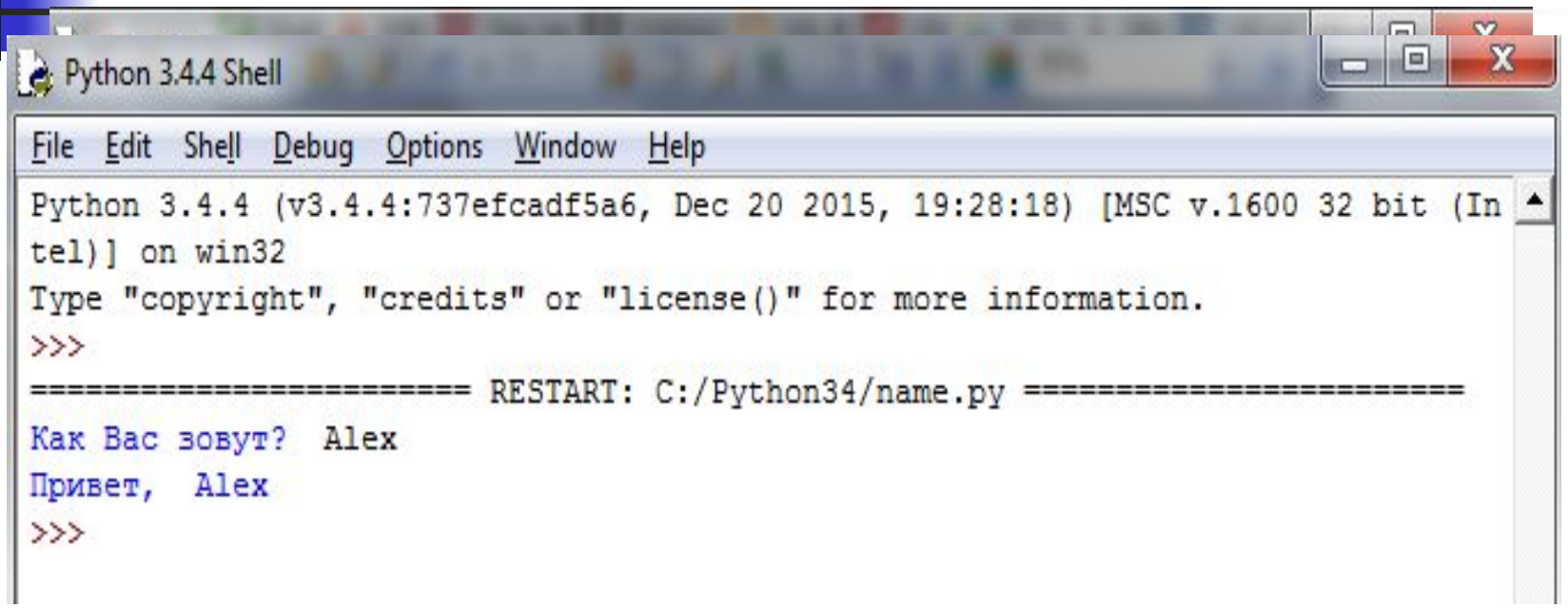
Python. Print.

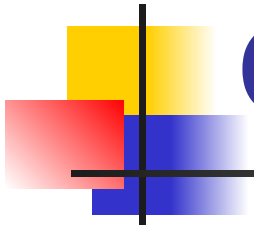



```
>>>print(2+8)
10
>>> print(2**3)
8
>>>print(8/3)
2.66666666665
```

```
>>> print(9-7)
2
>>> x=2+7; print(x)
9
>>>print(int(8/3))
2
```

Python.


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Python34/name.py ==========
Как Вас зовут? Alex
Привет, Alex
>>>



Синтаксис языка Python

- Конец строки является концом инструкции (точка с запятой не требуется).
- Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков. И про читаемость кода не забывайте. Отступ в 1 пробел, к примеру, не лучшее решение. Используйте 4 пробела (или знак табуляции, на худой конец).
- Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции.

Основная инструкция:

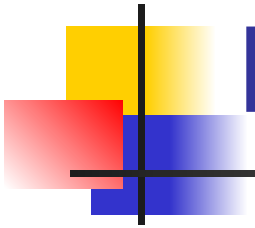
    вложеный блок инструкций

Ключевые слова (1)

- **False** - ложь.
- **True** - правда.
- **None** - "пустой" объект.
- **and** - логическое И.
- **with / as** - менеджер контекста.
- **assert** условие - возбуждает исключение, если условие ложно.
- **break** - выход из цикла.
- **class** - пользовательский тип, состоящий из методов и атрибутов.
- **continue** - переход на следующую итерацию цикла.
- **def** – определение функции.
- **del** - удаление объекта.
- **elif** - в противном случае, если.
- **else** - см. for/else или if/else.
- **except** - перехватить исключение.
- **finally** - вкупе с инструкцией try, выполняет инструкции независимо от того, было ли исключение или нет.

Ключевые слова (2)

- **for** - цикл for.
- **from** - импорт нескольких функций из модуля.
- **global** - позволяет сделать значение переменной, присвоенное ей внутри функции, доступным и за пределами этой функции.
- **if** - если.
- **import** - импорт модуля.
- **in** - проверка на вхождение.
- **is** - ссылаются ли 2 объекта на одно и то же место в памяти.
- **lambda** - определение анонимной функции.
- **nonlocal** - позволяет сделать значение переменной, присвоенное ей внутри функции, доступным в объемлющей инструкции.
- **not** - логическое НЕ.
- **or** - логическое ИЛИ.
- **pass** - ничего не делающая конструкция.
- **raise** - возбудить исключение.
- **return** - вернуть результат.
- **try** - выполнить инструкции, перехватывая исключения.
- **while** - цикл while.
- **yield** - определение функции-генератора.



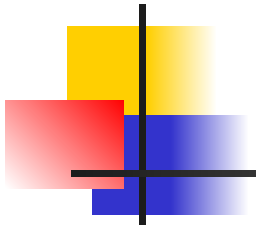
Модуль keyword

- **keyword.kwlist** - список всех доступных ключевых слов.
- **keyword.iskeyword(строка)** - является ли строка ключевым словом.

Встроенные функции, выполняющие преобразование типов (1)

- **bool**(x) - преобразование к типу bool, использующая стандартную процедуру проверки истинности. Если x является ложным или опущен, возвращает значение False, в противном случае она возвращает True.
- **bytearray**([источник [, кодировка [ошибки]]]) - преобразование к bytearray. Bytearray - изменяемая последовательность целых чисел в диапазоне 0≤X<256. Вызванная без аргументов, возвращает пустой массив байт.
- **bytes**([источник [, кодировка [ошибки]]]) - возвращает объект типа bytes, который является неизменяемой последовательностью целых чисел в диапазоне 0≤X<256. Аргументы конструктора интерпретируются как для bytearray().
- **complex**([real[, imag]]) - преобразование к комплексному числу.
- **dict**([object]) - преобразование к словарю.
- **float**([X]) - преобразование к числу с плавающей точкой. Если аргумент не указан, возвращается 0.0.

Встроенные функции, выполняющие преобразование типов (2)



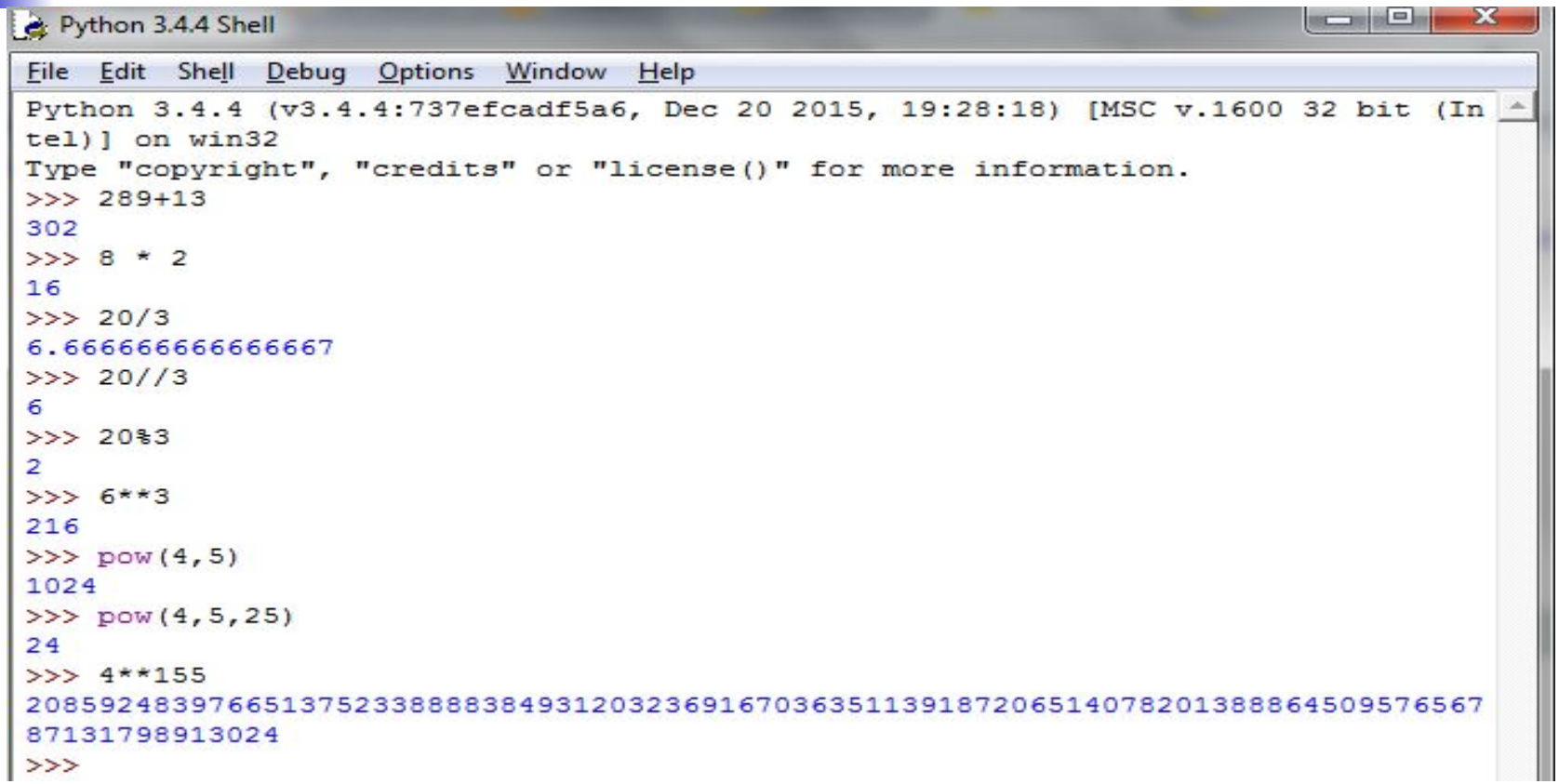
- **frozenset**([последовательность]) - возвращает неизменяемое множество.
- **int**([object], [основание системы счисления]) - преобразование к целому числу.
- **list**([object]) - создает список.
- **memoryview**([object]) - создает объект memoryview.
- **object**() - возвращает безликий объект, являющийся базовым для всех объектов.
- **range**([start=0], stop, [step=1]) - арифметическая прогрессия от start до stop с шагом step.
- **set**([object]) - создает множество.
- **slice**([start=0], stop, [step=1]) - объект среза от start до stop с шагом step.
- **str**([object], [кодировка], [ошибки]) - строковое представление объекта. Использует метод __str__.
- **tuple**(obj) - преобразование к кортежу.

Числа в Python 3

- Числа в Python 3 целые, вещественные, комплексные. Работа с числами и операции над ними.

x+y	Сложение
x-y	Вычитание
x*y	Умножение
x/y	Деление
x//y	Получение целой части от деления
x%y	Остаток от деления
-x	Смена знака числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Возведение в степень
pow(x, y[, z])	x^y по модулю (если модуль задан)

Примеры


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 289+13
302
>>> 8 * 2
16
>>> 20/3
6.666666666666667
>>> 20//3
6
>>> 20%3
2
>>> 6**3
216
>>> pow(4,5)
1024
>>> pow(4,5,25)
24
>>> 4**155
2085924839766513752338888384931203236916703635113918720651407820138886450957656787131798913024
>>>

Битовые операции

Над целыми числами также можно производить битовые операции

x | y	Побитовое *или*
x ^ y	Побитовое *исключающее или*
x & y	Побитовое *и*
x << n	Битовый сдвиг влево
x >> y	Битовый сдвигв право
~x	Инверсия битов

Системы счисления

Python для этого предоставляет несколько функций:

- **int**([object], [основание системы счисления]) - преобразование к целому числу в десятичной системе счисления. По умолчанию система счисления десятичная, но можно задать любое основание от 2 до 36 включительно.
- **bin**(x) - преобразование целого числа в двоичную строку.
- **hex**(x) - преобразование целого числа в шестнадцатеричную строку.
- **oct**(x) - преобразование целого числа в восьмеричную строку.

Вещественные числа (float)

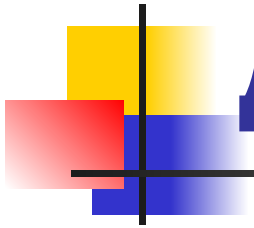
Вещественные числа поддерживают те же операции, что и целые. Однако (из-за представления чисел в компьютере) вещественные числа неточны, и это может привести к ошибкам:

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.9999999999999999
```

Для высокой точности используют другие объекты (например Decimal и Fraction)).

Также вещественные числа не поддерживают длинную арифметику:

```
>>> x=4**1000
>>> x+0.1
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
 x+0.1
OverflowError: int too large to convert to float
```



Дополнительные методы

- **float.as_integer_ratio**() - пара целых чисел, чьё отношение равно этому числу.
- **float.is_integer**() - является ли значение целым числом.
- **float.hex**() - переводит float в hex (шестнадцатеричную систему счисления).
- classmethod **float.fromhex**(s) - float из шестнадцатеричной строки.

```
>>> (8.5).hex()
'0x1.100000000000p+3'
>>> float.fromhex('0x1.100000000000p+3')
8.5
```

Математические модули

Помимо стандартных выражений для работы с числами (а в Python их не так уж и много), в составе Python есть несколько полезных модулей.

Модуль math предоставляет более сложные математические функции.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(64)
8.0
```

Модуль random реализует генератор случайных чисел и функции случайного выбора.

```
>>> import random
>>> random.random()
0.1408232543817861
```

Комплексные числа (complex)

В Python встроены также и комплексные числа:

```
>>> x=complex(2,3)
>>> print(x)
(2+3j)
>>> y=complex(3,4)
>>> print(y)
(3+4j)
>>> z=x+y
>>> print(z)
(5+7j)
```

Строки в Python. Литералы строк

- упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Строки в апострофах и в кавычках

```
>>> z='stroka"s'
>>> z="storka's"
```

Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

Строки в Python. Литералы строк (2)

Экранированные последовательности - служебные символы

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh…	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

Строки в Python. Литералы строк (3)

Строки в тройных апострофах или кавычках

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

```
>>> b="'строка которая
занимает много
строк'"
>>> print(b)
строка которая
занимает много
строк
```

Строки в Python. Базовые операции

- Конкатенация (сложение)

```
>>> s1='dog'
>>> s2='&cat'
>>> print(s1+s2)
dog&cat
```

- Дублирование строки

```
>>> print(s1*3)
dogdogdog
```

- Длина строки (функция len)

```
>>> len('&cat')
4
```

- Доступ по индексу

```
>>> S1='Alex'
>>> S1[0]
'A'
>>> S1[2]
'e'
>>> S1[-2]
'e'
```

Строки в Python. Базовые операции

- **Извлечение среза**

  Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание; символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

```
>>> s='srez stroki Alex'
>>> s[3:7]
'z st'
>>> s[2:-2]
'ez stroki Al'
>>> s[:5]
'srez `
>>> s[1:]
'rez stroki Alex'
>>> s[:]
'srez stroki Alex'
>>> s[::-1]
'xelA ikorts zers'
>>> s[2::2]
'e toiAe'
```

Списки (list). Функции и методы списков

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

```
>>> list('spisok Alex')
['s', 'p', 'i', 's', 'o', 'k', ' ', 'A', 'l', 'e', 'x']
```

Список можно создать и при помощи литерала:

```
>>> s=[]
>>> l=['a','l',['ex'],2]
>>> s
[]
>>> l
['a', 'l', ['ex'], 2]
```

Функции и методы списков

list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Словари (dict) и работа с ними. Методы словарей

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами. С помощью литерала:

```
>>> d={}
>>> d
{}
>>> d={'odin':1, 'dva':2}
>>> d
{'dva': 2, 'odin': 1}
```

Словари (dict) и работа с ними. Методы словарей

- С помощью функции **dict**:

```
>>> d=dict(short='dict',long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d=dict([(1,1), (2,5)])
>>> d
{1: 1, 2: 5}
```

Словари (dict) и работа с ними.

- С помощью метода fromkeys:

```
>>> d=dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
>>> d=dict.fromkeys(['a', 'b'],100)
>>> d
{'a': 100, 'b': 100}
```