

# Неделя 1. Введение в Python

## Цели лекций Недели 1й

- ознакомиться с основными встроенными типами данных Python, доступной функциональностью;
- научиться работать с передачей Python-скрипту параметров, введенных в консоли;
- научиться работать с документацией, встроенной справкой;
- понять механизм изучения нового языка программирования: КАК его изучать, на что обращать внимание.

## План лекций

План лекций .....	1
О конспекте лекций.....	3
Использование Python в качестве калькулятора.....	4
Основные операции, введение в типы данных.....	4
Операции - уточнение отличий для 2.x и 3.x .....	4
Переменные в Python .....	5
Числовые типы данных (целые числа, числа с плавающей точкой и др.) - и операции с ними, немного о преобразовании типов.....	6
Тип данных СТРОКА (str) .....	7
Справка о строках.....	7
Специальные функции:.....	8
Справка о функциях:.....	9
Некоторые символы, операторы, функции, задачи .....	9
Тип данных СПИСОК (List) .....	12
Справка о LIST .....	12
Справка о функциях :.....	12
Функции.....	12
Об изменяемости списка (mutable type) .....	16
Тип данных КОРТЕЖ (Tuple).....	17
Справка о TUPLE.....	17
Функции.....	18
Тип данных СЛОВАРЬ (Dictionary) .....	19
Справка о DICT .....	19
Проверка типа и справка по функциям: .....	19
Функции.....	19
Демонстрация изменяемости словаря.....	21

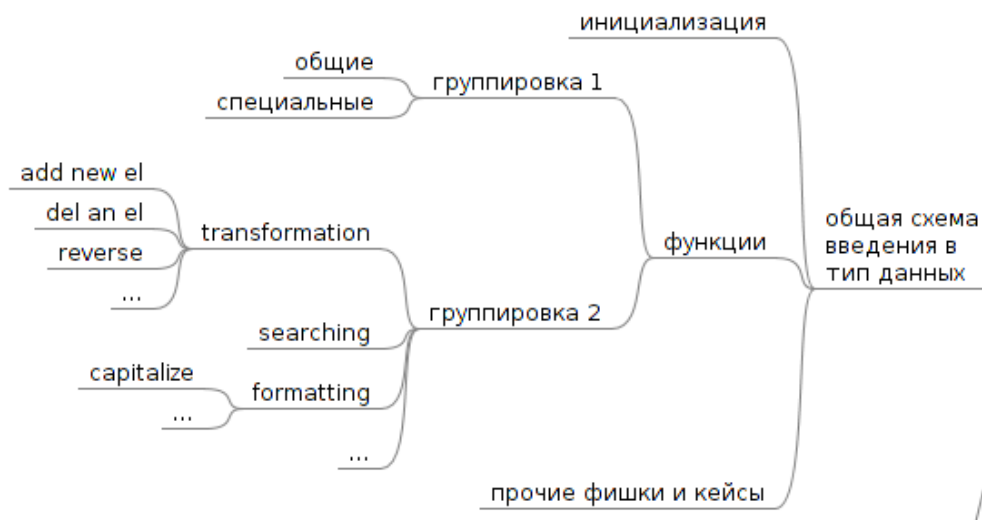
Тип данных ЛОГИЧЕСКИЙ (Bool) .....	22
Описание .....	22
Выдержки из лекции .....	24
Оператор not: .....	24
Дополнительный источник .....	25
Встроенные возможности языка Python .....	25
builtins-s .....	25
Некоторые часто используемые функции и конструкции .....	25
Как выводить справку .....	27
Ubuntu Shell: некоторые команды .....	27
Передача параметров из консоли .....	28
If statement, Циклы while и for .....	29
Перехват ошибок .....	32

## О конспекте лекций

По какому принципу изложен материал в конспекте лекций (КЛ):

КЛ освещает ключевые/проблемные моменты видео-лекций, записанных инструктором, в т.ч. включая в себя краткие теоретические выдержки, выложенные на платформе edX вместе с видео. В КЛ включены дополнительные материалы для ознакомления.

Данная лекция посвящена, в основном, основным типам данных в Python. Каждый тип данных желательно изучать по следующей схеме:



При этом, данная лекция — это введение в Python, и информация о типах данных, доступных основных и специальных функциях не является исчерпывающей. Крайне рекомендуется изучать документацию, смотреть встроенную справку, экспериментировать!

В лекционном материале есть 90% информации, необходимой для успешного прохождения тестов и выполнения контрольных заданий. Остальное — самостоятельная работа с документацией, что, по нашему мнению, также крайне важно для развития навыков программирования.

## Использование Python в качестве калькулятора

### Основные операции, введение в типы данных

Интерпретатор работает как простой калькулятор. Вы можете набрать выражение, и он выведет результат. Python включает в себя вполне ожидаемые типы: целые числа (числа без дробной части), вещественные числа (числа с десятичной точкой) и более экзотические типы (комплексные числа, числа с фиксированной точностью, рациональные числа и т.д.)

Числа Python поддерживают набор самых обычных математических операций. Например:

"+" - сложение

"-" - вычитание

"\*" - умножение

"\*\*" - возведение в степень

"/" - деление

"//" - деление с округлением вниз

"%" - остаток от деления

```
>>> 2+2
4
>>> 2-1
1
>>> 4*3
12
>>> 5/2
2.5
>>> 5/3
1.6666666666666667
>>> 5/5
1.0
>>> 5.4
5.4
>>> .0
0.0
>>> 5/3
1.6666666666666667
>>> 5//3
1
>>> 5//2
2
>>> 5%2
1
```

Для группирования используют скобки "(")

О приоритете операций: <https://docs.python.org/3.3/reference/expressions.html#operator-precedence>

### Операции - уточнение отличий для 2.x и 3.x

#### x / y

Классическое и истинное деление. В Python 2.7 и в более ранних версиях этот оператор выполняет операцию классического деления, когда дробная часть результата усекается при делении целых чисел и сохраняется при делении вещественных чисел. В Python 3.0 этот выполняет операцию истинного деления, которая всегда сохраняет дробную часть независимо от типов операндов.

## X // Y

Деление с округлением вниз. Этот оператор впервые появился в Python 2.2 и доступен в обеих версиях Python, 2.7 и 3.0. Он всегда отсекает дробную часть, округляя результат до ближайшего наименьшего целого независимо от типов операндов.

Больше примеров здесь: [https://pythlife.blogspot.com/2012/09/blog-post\\_5355.html](https://pythlife.blogspot.com/2012/09/blog-post_5355.html)

## Переменные в Python

Все переменные в Python одновременно являются и ссылками. Эта особенность Python избавляет от необходимости создавать дополнительные объекты типа указателей. При создании переменной в Python автоматически создается и ссылка на этот объект.

Знак равенства "=" используется для присваивания значения переменной. Значение можно присвоить одновременно нескольким переменным.

```
>>>
>>> width = 20
>>> height = 5*9
>>> width * height
900
>>> x = y = z = 1
>>> x
1
>>> y
1
>>> z
1
>>>
```

Помимо выражений для выполнения операций с числами в составе Python есть несколько полезных модулей:

- math содержит более сложные математические функции;
- random реализует генератор случайных чисел и функцию случайного выбора

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.sqrt(16)
4.0
>>>
>>> import random
>>> random.random()
0.15534513197089317
>>> random.choice([1, 2, 3, 4])
1
>>> random.choice([1, 2, 3, 4])
4
>>>
>>>
```

Немного глубже:

- [Официальный туториал на эту тему](#)

## Числовые типы данных (целые числа, числа с плавающей точкой и др.) - и операции с ними, немного о преобразовании типов

В Python - **динамическая** типизация, т.е. тип переменной/константы определяется после введения (присвоения) значения переменной/константы; иными словами, тип определяется во время выполнения. При этом, Python является языком со **строгой** типизацией, то есть, к примеру, нельзя складывать '12' (строка) и 12 (число) – необходимо явное преобразование типов (о преобразованиях – далее в лекции), т.е. тип переменной имеет значение. Для сравнения: в JavaScript, например, можно объединить строку '12' и целое число 3 для получения строки '123', что возможно без явного преобразования типов.

### Переопределение переменной:

```
>>> a = 5
>>> b
497
>>> a = a + 1
>>> a
6
>>> a = a + 3
>>> a
9
```

```
>>> a += 2
>>> a
11
>>> a -= 1
>>> a
10
```

### Преобразование типов данных:

```
>>> type(f)
<class 'float'>
>>> f = 10
>>> type(f)
<class 'int'>
>>> f1 = int(f)
>>> f1
10
>>> type(f1)
<class 'int'>
>>> float(5)
5.0
>>> int(45.678)
45
```

### Округление чисел:

```
>>> round(123.456)
123.0
>>> round(123.456, 2)
123.46
```

Все приведенные выше функции являются встроенными (builtins). Для выполнения дополнительных функций необходимо обращаться к встроенным библиотекам: math, random и др., - которые нужно явно импортировать.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> import random
>>> random.random()
0.035261753337078594
>>>
```

## Тип данных СТРОКА (str)

### Справка о строках

Чтобы создать литерал строки, ее необходимо заключить в апострофы ('), в кавычки (") или в тройные кавычки (""", """). Строковый литерал должен завершаться кавычками того же типа, как использовались в начале.

Строки хранятся как последовательности символов, доступ к которым можно получить с помощью целочисленного индекса, начиная с нуля. В языке Python предусмотрена возможность индексации в обратном порядке. Можно обращаться не только к одиночным символам, но и к целым подстрокам - срезам. В общем виде: `string[n:m:step]`, где `n` указывает индекс начала среза, а `m` - индекс конца среза, но символ с индексом `m` в срез не включается, `step` - шаг с которым формируется срез, по умолчанию опускается и равен 1. Если не указать начальный или конечный индекс среза то будет подразумеваться начало или конец строки соответственно.

Более детально о срезах здесь: <https://docs.python.org/3/library/stdtypes.html#string-methods>

```
>>>
>>> 'hello'
'hello'
>>> "hello"
'hello'
>>> """hello"""
'hello'
>>>
>>> s = "Hello world"
>>> s[0]
'H'
>>> s[1]
'e'
>>> s[-1]
'd'
>>> s[1:4]
'ell'
>>> s[1:]
'ello world'
>>> s[:-2]
'Hello wor'
>>>
```

```
>>>
>>> s = 'spam'
>>> s + 'adc'
'spamadc'
>>> s
'spam'
>>> s * 3
'spamspamspam'
>>>
```

Строки поддерживают операцию конкатенации, которая записывается в виде знака "+" (объединение двух строк в одну строку), и операцию повторения (новая строка создается за счет многократного повторения другой строки - "\*" на целое число)

Все операции над строками в результате создают новую строку, потому что строки в языке Python являются неизменяемые - после того, как строка создана, ее нельзя изменить.

### Специальные функции:

Функция	Описание
s.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные - в нижний
s.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные - в нижний
s.upper()	Преобразование строки к верхнему регистру
s.lower()	Преобразование строки к нижнему регистру
s.strip()	Удаление пробельных символов в начале и в конце строки
s.lstrip()	Удаление пробельных символов в начале строки
s.rstrip()	Удаление пробельных символов в конце строки
s.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
s.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
s.isalnum()	Состоит ли строка из цифр или букв
s.isalpha()	Состоит ли строка из букв
s.isdigit()	Состоит ли строка из цифр

Больше информации <https://docs.python.org/3/library/stdtypes.html#string-methods>

*Особенности: неизменяемый (immutable) тип данных.*



```

>>> s='34'
>>> type(s)
<class 'str'>
>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

```

## Некоторые символы, операторы, функции, задачи

Функция / оператор / задача	Фрагмент кода	Дополнительноеписание / фрагмент кода
Экранирование \ 	<pre> &gt;&gt;&gt; "Hello \" world!" 'Hello " world!' </pre>	Символ экранирования (для использования символа кавычек как части строки, не для инициализации данных строкового типа).
Инициализация строки	Инициализация пустой строки: <pre> &gt;&gt;&gt; str() '' </pre>	Пример явного преобразования числового типа к строковому: <pre> &gt;&gt;&gt; str(6) '6' &gt;&gt;&gt; type('s') &lt;class 'str'&gt; </pre>
Перенос строки \ 	<pre> &gt;&gt;&gt; s = "Hello \n world!" &gt;&gt;&gt; print(s) Hello  world! &gt;&gt;&gt; s = "Hello \n\n world!" &gt;&gt;&gt; print(s) Hello  world! </pre>	Перенос строки (при вызове функции print() )
Кавычки: вывод строк в консоли	<pre> &gt;&gt;&gt; "Hello world!" 'Hello world!' &gt;&gt;&gt; 'Hello world!' 'Hello world!' &gt;&gt;&gt; "Hello' world!" "Hello' world!" &gt;&gt;&gt; 'Hello" world!' 'Hello" world!' </pre>	Варианты использования кавычек.

print строки	<pre>&gt;&gt;&gt; s = "Hello world!" &gt;&gt;&gt; s 'Hello world!' &gt;&gt;&gt; print s Hello world!</pre>	Обратить внимание на отсутствие кавычек при выводе строки командой print.
+ (конкатенация)	<pre>&gt;&gt;&gt; 'Hello' + ' world!' 'Hello world!' &gt;&gt;&gt; s = 'Hello' + ' world!' &gt;&gt;&gt; s 'Hello world!'</pre>	
* («умножение», повторение, repeat)	<pre>&gt;&gt;&gt; 'ha-' * 3 'ha-ha-ha-'</pre>	Строка конкатенируется сама с собой.
Доступ к символу строки по индексу	<pre>&gt;&gt;&gt; s 'Hello world!' &gt;&gt;&gt; s[0] 'H' &gt;&gt;&gt; s[1] 'e'</pre>	<p>Подобно доступу к элементу массива в Java, C#...</p> <p>Попытка получения доступа к несуществующему элементу:</p> <pre>&gt;&gt;&gt; s[20] Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; IndexError: string index out of range</pre> <p>Здесь длина строки меньше, чем введенный индекс, поэтому получили IndexError (тема стектрейса — далее в этой лекции).</p>
len(string) # длина строки	<pre>&gt;&gt;&gt; s 'Hello world!' &gt;&gt;&gt; len(s) 12</pre>	
Слайс (slice)	<pre>&gt;&gt;&gt; s 'Hello world!' &gt;&gt;&gt; s[11] '!' &gt;&gt;&gt; s[0:5] 'Hello' &gt;&gt;&gt; s[5] ' ' &gt;&gt;&gt; s[2:5] 'llo' &gt;&gt;&gt; s[2:10] 'llo worl' &gt;&gt;&gt; s[:5] 'Hello' &gt;&gt;&gt; s[6:] 'world!' &gt;&gt;&gt; s[6:11] 'world'</pre> <p>Отрицательная индексация:</p> <pre>&gt;&gt;&gt; s[-1] '!' &gt;&gt;&gt; s[-2] 'd' &gt;&gt;&gt; s[6:-1] 'world'</pre>	<p>s — переменная с предыдущего кейса.</p> <p>Принцип работы слайса:</p> <pre>+---+---+---+---+---+   P   y   t   h   o   n   +---+---+---+---+---+ 0  1  2  3  4  5  6 -6 -5 -4 -3 -2 -1</pre>
Преобразование строки: s.title(), s.capitalize(), s.title(), s.capitalize(),	<pre>&gt;&gt;&gt; s.title() 'Hello World!' &gt;&gt;&gt; s.capitalize() 'Hello world!' &gt;&gt;&gt; s.upper() 'HELLO WORLD!'</pre>	Возвращает копию строки! Исходную строку не меняет (строка — немодифицируемый тип данных)!

s.upper(), и т.д.	<code>'HELLO WORLD!'</code>	
Информация о строке: s.count(symbol), s.index(symbol)	<pre>&gt;&gt;&gt; s 'Hello world!' &gt;&gt;&gt; s.count('l') 3 &gt;&gt;&gt; s.count('h') 0 &gt;&gt;&gt; s.count('o') 2 &gt;&gt;&gt; s.count('ll') 1</pre>	Вхождение символа или подстроки (совокупности символов) в исходную строку.
string.replace('what', 'with_what')		<p>Пример применения: проверка пользовательского ввода на введение числовых символов (в данном листинге допустимо дробное число):</p> <pre>def input_parameter(parameter_name='a'):     while True:         p = input("Enter the parameter of the equation: %s = " % parameter_name)         if p.replace('.', '').isdigit() and float(p) != 0:             return float(p)         else:             print("Please enter the number of nonzero!")</pre> <p>Для того, чтобы ввод отрицательного числа был допустимым, вводим доп.реплейс (красную на желтую часть кода):</p> <pre>if p.replace('.', '').replace('-', '').isdigit() and float(p) != 0:     return float(p)</pre>

## Форматирование строк

Примеры форматирования строки в Python (примеры):

```
import math
r = 10
s = math.pi * r ** 2
print("Square: " % s)
```

```
def f(a, b, c=0, d=0):
    print('a = {} b = {} c = {} d = {}'.format(a,b,c,d))
```

```
def hello(name, age=0, title='Mr.'):
    print("Hello %s %s" % (title,name))
```

Ввиду наличия более оптимальных возможностей **не рекомендуется** для форматирования строк использовать **конкатенацию**:

не нужно делать так:

```
ourString = "Here is " + a + ", " + b + " and " + c
```

В PEP8 использован подход к форматированию строки с использованием спецификатора (плейсхолдера) %:

```
firstPrinciple = "PEP8"
secondPrinciple = "Zen"
ourString = ("We should follow the %s and %s principles" %
             (firstPrinciple, secondPrinciple))
```

ВАЖНО: плейсхолдер числа - %d, плейсхолдер строки - %s:

```
>>> "%s %d %s!" % ("Hello", 1, "world")
'Hello 1 world!'
```

Доки о форматировании: <https://docs.python.org/3/tutorial/inputoutput.html#fancier-output-formatting>

## Тип данных СПИСОК (List)

### Справка о LIST

Особенность: Изменяемый тип данных.

### Справка о функциях :

```
>>> l = [3]
>>> type(l)
<class 'list'>
>>> dir(l)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

## Функции

Принятые обозначения: `our_list` – пользовательские элементы

Функция / операция / задача	Фрагмент кода	Описание
Инициализация <code>[]</code>	<pre>&gt;&gt;&gt; [1, 2, 3] [1, 2, 3] &gt;&gt;&gt; l = [1, 2, 3] &gt;&gt;&gt; l = [1, 2, 3, 'cat'] &gt;&gt;&gt; l [1, 2, 3, 'cat']  &gt;&gt;&gt; list() []</pre>	Список в Python не является строго типизированным.  Возможна инициализация пустого списка функцией <code>list</code> (также

	<pre>&gt;&gt;&gt; [] []  &gt;&gt;&gt; list('a') ['a']</pre>	<p>используется для преобразоварния типов данных в данные типа список).</p> <p>IDE (PyCharm, например) больше по нраву инициализация пустого списка при помощи функции list(), но не при помощи квадратных скобок [].</p>
type(our_list)	<pre>&gt;&gt;&gt; l = [1, 5, 7] [1, 5, 7] &gt;&gt;&gt; type(l) &lt;class type 'list'&gt;</pre>	Узнать тип данных
dir(our_list)	<pre>dir(our_list)</pre>	Узнать доступные функции для типа данных.
+ (объединение)	<pre>&gt;&gt;&gt; [1, 2] + [4, 5] [1, 2, 4, 5] &gt;&gt;&gt; [1, 2] + [4, 'cat'] [1, 2, 4, 'cat'] &gt;&gt;&gt; l = [1, 2] + [4, 'cat'] &gt;&gt;&gt; l [1, 2, 4, 'cat']</pre>	Объединение списков.
* (повторение)	<pre>&gt;&gt;&gt; [1] * 10 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]</pre>	Расширение списка за счет дублирования исходного элемента списка.
Доступ к элементу по индексу	<pre>&gt;&gt;&gt; l [1, 2, 4, 'cat'] &gt;&gt;&gt; l[0] 1 &gt;&gt;&gt; l[-1] 'cat'</pre>	Поддерживаются слайсы, отрицательная индексация.
len(our_list)	<pre>&gt;&gt;&gt; l [1, 2, 4, 'cat'] &gt;&gt;&gt; len(l) 4</pre>	Длина списка.
del our_list[our_index]	<pre>&gt;&gt;&gt; l = ['m', 'a', 'n', 'g', 'o'] &gt;&gt;&gt; del l[3] &gt;&gt;&gt; l ['m', 'a', 'n', 'o']</pre>	Удалить определенный элемент списка по его индексу.
our_list.remove(value)	<pre>&gt;&gt;&gt; l ['m', 'a', 'n', 'o'] &gt;&gt;&gt; l.remove('m') &gt;&gt;&gt; l ['a', 'n', 'o']</pre>	Удалить определенный объект из списка по его значению (обратить внимание на отличие от оператора/функции del)
our_list.pop()	<pre>&gt;&gt;&gt; l ['a', 'n', 'o'] &gt;&gt;&gt; l.pop() 'o'</pre>	Удаляет и возвращает последний элемент списка.
our_list.index(value)	<pre>&gt;&gt;&gt; l = [1, 2, 'cat', 'raccoon', 1, 1, 2] &gt;&gt;&gt; l.index(1) 0 &gt;&gt;&gt; l.index(1, 0) 0</pre>	<p>Узнать индекс элемента (по первому совпадению);</p> <p>Необязательные элементы – начальный и конечный индексы,</p>

	<pre>&gt;&gt;&gt; l.index(1, 1) 4 &gt;&gt;&gt; l.index(1, 5) 5</pre>	задающие интервал поиска совпадений.
help(our_list.index)	<pre>&gt;&gt;&gt; help(l.index) Help on built-in function index:  index(...)     l.index(value, [start, [stop]]) -&gt;     integer -- return first index of value.     Raises ValueError if the value is     not present.</pre>	Справка по определенной функции
our_list.append(value)	<pre>&gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog', 'raccoon'] &gt;&gt;&gt; l.append(1) &gt;&gt;&gt; l.append(1) &gt;&gt;&gt; l.append(2) &gt;&gt;&gt; l [1, 2, 'cat', 10, 'dog', 'raccoon', 1, 1, 2]</pre>	Добавление элемента в конец списка.
our_list.insert(index, value)	<pre>&gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog', 'raccoon'] &gt;&gt;&gt; l.insert(6, 'rabbit') &gt;&gt;&gt; l [1, 2, 'cat', 10, 'dog', 'raccoon', 'rabbit']</pre>	Добавить элемент в список по определенному индексу. Обратит внимание на отличие от функции append().
our_list.extend(other_list)	<pre>&gt;&gt;&gt; l1 = ['123', 'bonobo'] &gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog'] &gt;&gt;&gt; l1 = ['123', 'bonobo'] &gt;&gt;&gt; l.extend(l1) &gt;&gt;&gt; l [1, 2, 'cat', 10, 'dog', '123', 'bonobo']</pre>	Расширение списка (объединение списков «в первый список», единственный). Важно: в отличие от функции extend, операция объединения списков (оператор «+») не видоизменяет исходные списки:
our_list.count(value)	<pre>&gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog', 2] &gt;&gt;&gt; l.count(2) 2 &gt;&gt;&gt; l.count(1) 1 &gt;&gt;&gt; l.count('cat') 1</pre>	Определить вхождение элементов (количество вхождений)
our_list.sort()	<pre>&gt;&gt;&gt; l1 = [1,4,3,9,6] &gt;&gt;&gt; l1.sort() &gt;&gt;&gt; l1 [1,3,4,6,9]  &gt;&gt;&gt; l2 = ['a', 'x', 'b'] &gt;&gt;&gt; l2.sort() &gt;&gt;&gt; l2 ['a', 'b', 'x']  &gt;&gt;&gt; l3 = [1, 2, 'cat', 10, 'dog', 2] &gt;&gt;&gt; l3.sort() Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; TypeError: unorderable types: str() &lt; int()</pre>	Принцип работы сортировки в Python3.x: только <b>однородные списки</b> , поскольку операции сравнения могут теперь работать только с величинами одного типа. (в Python2.x сортировало сначала - числа, потом – буквы (согласно ASCII таблицам)). <b>Внимание:</b> функция sort модифицирует исходный список.

<code>our_list.reverse()</code>	<pre>&gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog', 2] &gt;&gt;&gt; l.reverse() &gt;&gt;&gt; l [2, 'dog', 10, 'cat', 2, 1]</pre>	<p>Реверс элементов. Функция модифицирует исходный список.</p> <p>Работает и со смешанными списками.</p>
<code>"symbol".join(our_list)</code>	<pre>&gt;&gt;&gt; l = ['cat', 'dog'] &gt;&gt;&gt; ''.join(l) 'catdog' &gt;&gt;&gt; '_'.join(l) 'cat_dog' &gt;&gt;&gt; ', '.join(l) 'cat,dog'</pre>	<p><b>Получение строки</b> из списка.</p> <p>Все элементы списка должны быть строками; в противном случае генерируется исключение:</p> <pre>&gt;&gt;&gt; l = [1, 2, 'cat', 10, 'dog', 2] &gt;&gt;&gt; ''.join(l) Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; TypeError: sequence item 0: expected string, int found</pre>
<code>our_string.split(symbol)</code>	<pre>&gt;&gt;&gt; s = 'Hello world' &gt;&gt;&gt; s.split() ['Hello', 'world'] &gt;&gt;&gt; s.split('l') ['He', '', 'o wor', 'd'] &gt;&gt;&gt; s.split('ll') ['He', 'o world']</pre>	<p>Разбиение строки на элементы списка.</p> <p>Аргументом функции могут быть только элементы строчного типа. Например, при попытке передать функции список, генерируется исключение. Сам символ /separator/ (переданный в аргумент функции) в итоге «выпадает». <b>Сепаратор не должен быть пуст, иначе генерируется исключение:</b></p> <pre>&gt;&gt;&gt; s = 'Hello world' &gt;&gt;&gt; s.split() # no separator indicated ['Hello', 'world'] &gt;&gt;&gt; s = 'Hello world' &gt;&gt;&gt; s.split('') # empty separator indicated Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; ValueError: empty separator</pre>
Мин,макс сумма	<pre>&gt;&gt;&gt; l = ['cat', 'dog'] &gt;&gt;&gt; min(l) 'cat' &gt;&gt;&gt; max(l) 'dog' &gt;&gt;&gt; l = [1, 2, 3] &gt;&gt;&gt; min(l) 1 &gt;&gt;&gt; max(l) 3 &gt;&gt;&gt; sum(l) 6</pre>	<p>Аргументы функций <code>max</code> и <code>min</code> необязательно должны <b>быть числовыми</b>: сравнение осуществляется на основе кодов символов согласно ASCII-таблицам.</p> <p><b>Только однородные списки!</b></p>
<code>range(number)</code>	<pre>&gt;&gt;&gt; range(1, 10) range(1, 10) &gt;&gt;&gt; list(range(1,10)) [1, 2, 3, 4, 5, 6, 7, 8, 9] &gt;&gt;&gt; list(range(10, 1, -1)) [10, 9, 8, 7, 6, 5, 4, 3, 2] &gt;&gt;&gt; list(range(10))</pre>	<p><b>Неизменяемый итерируемый тип данных</b></p> <p>Производит элементы заданной последовательности.</p> <p>(в Python 2.x – функция - возвращает список элементов заданной последовательности).</p> <p><b>Справка о типе данных функции:</b></p> <pre>&gt;&gt;&gt; help(range) Help on range object: range(...) range(stop) -&gt; range object range(start, stop[, step]) -&gt; range</pre>

	<pre>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] &gt;&gt;&gt; list(range(10, 1)) [] &gt;&gt;&gt; list(range(10, 1, -1)) [10, 9, 8, 7, 6, 5, 4, 3, 2] &gt;&gt;&gt; list(range(1, 10, 2)) [1, 3, 5, 7, 9] &gt;&gt;&gt; list(range(2, 10, 2)) [2, 4, 6, 8]</pre> <p>Получить только четные числа последовательности чисел от 2 до 9:</p> <pre>&gt;&gt;&gt; list(range(2, 10, 2)) [2, 4, 6, 8]</pre>	<p><b>object</b> Return an object that produces a sequence of integers from start (inclusive) to stop(exclusive) by step. range(i, j) produces i, i+1, i+2, ..., j-1; start defaults to 0, the end point is omitted range(4) produces 0, 1, 2, 3. These are exactly the valid indices for a list of 4 elements. When step is given, it specifies the increment (or decrement).</p> <p>При помощи функции list() получаем результат, ожидаемый в Python2.x. Но при этом в цикле for range используется напрямую, функцию list() не требуется. Декремент можно реализовать указанием отрицательного шага и старта с последнего элемента – так решается, к примеру, задача итерирования инверсированного списка:</p> <pre>&gt;&gt;&gt; l = ['b', 'o', 'b', 'b', 'y'] &gt;&gt;&gt; for i in range(len(l)-1, 0, -1): ...     print(l[i]) ... y b b o</pre> <p>хотя обращение к элементу итерируемого типа данных по индексу чаще всего не является наиболее оптимальным способом (альтернатива – слайсы, <b>enumerate</b> – см.далее в этой лекции).</p>
Работа с многомерными списками	<pre>&gt;&gt;&gt; l1 = [1, 2, 3] &gt;&gt;&gt; l2 = ['a', 'b', 'c'] &gt;&gt;&gt; l1.append(l2) &gt;&gt;&gt; l1 [1, 2, 3, ['a', 'b', 'c']] &gt;&gt;&gt; l1[3][0] 'a'</pre>	<p>Работа с «многомерными» списками реализуется благодаря вложенности «списков в списке».</p>

### Об изменяемости списка (mutable type)

Python поддерживает работу со слайсами списков. Примеры ниже также демонстрируют изменяемость списка (возможно изменять элемент, обращаясь к нему по индексу).

Кейс	Фрагмент кода
Добавление новых элементов в список при помощи слайса («разбросать» строку 'cat' в последовательность символов, и	<pre>&gt;&gt;&gt; l [1, 2, 4, 10, 'dog'] &gt;&gt;&gt; l[2:3] = 'cat' &gt;&gt;&gt; l [1, 2, 'c', 'a', 't', 10, 'dog']</pre>



добавить символы последовательности в список, начиная со 2-го элемента)	
Замена несколько элементов списка одним элементом.	(после кода выше)
Пример демонстрирует изменяемость (модифицируемость) списка.	<pre>&gt;&gt;&gt; l[2:4] = ['cat'] &gt;&gt;&gt; l [1, 2, 'cat', 't', 10, 'dog']</pre>

Модифицируемость списка поддерживается также специальными функциями модификации (del, например):

```
>>> l = [1, 2, 'cat']
>>> del l[2]
>>> l
[1, 2]
```

ВАЖНО: при изменении копии списка меняется также исходный список (ссылка на один и тот же объект):

```
>>> l = ['o', 'j', 'o']
>>> k = l
>>> k[0] = 'l'
>>> l
['l', 'j', 'o']
```

## Тип данных КОРТЕЖ (Tuple)

### Справка о TUPLE

Особенности:

- Неизменяемый тип; в примере ниже генерируется исключительная ситуация, т. к. нельзя назначить элемент через обращение к индексу (как и в строках).

```
>>> t = (1, 2, 3)
>>> t[0]
1
>>> t[0] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- Специальных функций для изменения кортежа не существует. Для добавления новых элементов объединяем кортежи при помощи обычного оператора сложения:

```

>>> t = (1, 2, 3)
>>> t = t + (4, 5, 6)
>>> t
(1, 2, 3, 4, 5, 6)
>>> t += (7, 8,)
>>> t
(1, 2, 3, 4, 5, 6, 7, 8)

```

Справка о функциях:

```

>>> dir(t)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']

```

## Функции

Операция / задача	Фрагмент кода
Инициализация  ( )	Возможна также инициализация словаря функцией tuple(), которая также используется для преобразований к типу данных «кортеж». <pre> &gt;&gt;&gt; (1, 2, 3) (1, 2, 3) &gt;&gt;&gt; t = (1,2,3) &gt;&gt;&gt; t (1, 2, 3) </pre>
Кортеж, состоящий из одного элемента	<pre>(1,)</pre>
Сложение	<pre> &gt;&gt;&gt; t = (1,2,3) &gt;&gt;&gt; t + (5,6) (1, 2, 3, 5, 6) </pre>
Определить количество вхождений	<pre> &gt;&gt;&gt; t (1, 2, 3, 5, 6) &gt;&gt;&gt; t.count(1) 1 </pre>
Определить длину	<pre> &gt;&gt;&gt; t (1, 2, 3, 5, 6) &gt;&gt;&gt; len(t) 5 </pre>
Определить индекс элемента	<pre> &gt;&gt;&gt; t (1, 2, 3, 5, 6, 1, 2, 2) &gt;&gt;&gt; t.count(2) 3 &gt;&gt;&gt; t.index(3) 2 </pre>
Слайсы кортежей	Также поддерживаются: <pre> &gt;&gt;&gt; t = (1, 2 ,3) &gt;&gt;&gt; t[:1] (1,) </pre>

## Тип данных СЛОВАРЬ (Dictionary)

### Справка о DICT

Особенности:

- Изменяемый тип данных.
- По сути, похож на ассоциативный массив PHP (пары «ключ – значение»).
- Тип данных неупорядочен (не сортирован). Интерпретатор упорядочивает случайным образом. То есть при работе со словарем мы не можем рассчитывать на то, что он вернет определенную пару «ключ-значение» по определенному порядковому номеру. Так, функция `popitem()` удаляет и возвращает первую пару словаря, но при каждом ее вызове будет удалена и возвращена случайная пара – точнее, первая пара случайным образом сгенерированной последовательности пар.
- {}
- ключ-значение, взятие значения по индексу
- функции:
  - `keys`, `values`, `items`
  - `pop`, `popitem`, `update`
  - `get`, оператор `'in'`

Методы `items()`, `values()`, `keys()` в Python 3.x у объектов `dict` возвращают итерируемый объект (в отличие от Python 2.x, где возвращался список ).

Проверка типа и справка по функциям:

```
>>> d = {}
>>> type(d)
<class 'dict'>
>>> dir(d)
['_class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__gt__',
 '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get',
 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

### Функции

Принятые обозначения: `our_list` – пользовательский элемент

Функция / операция / задача	Фрагмент кода / пояснение	Описание (с доп. фрагментами кода)
Инициализация  { }	<pre>&gt;&gt;&gt; d = {'cats': 12, 'dogs': 5} &gt;&gt;&gt; d {'cats': 12, 'dogs': 5}</pre>	Возможна также инициализация словаря функцией <code>dict()</code> , которая также используется для преобразований к типу данных «словарь».
<code>dict()</code>	<pre>&gt;&gt;&gt; dict(((1,2),)) {1: 2} &gt;&gt;&gt; dict(((1, 2), (2, 3))) {1: 2, 2: 3}</pre>	Преобразование кортежа кортежей к типу словарь.

Расширение словаря: добавление элемента	<pre>&gt;&gt;&gt; d {'cats': 13, 'dogs': 5} &gt;&gt;&gt; d['raccoons'] = 1 &gt;&gt;&gt; d {'cats': 13, 'dogs': 5, 'raccoons': 1} &gt;&gt;&gt; d['raccoons'] 1</pre>	
<code>our_dict['our_key']</code>	см. предыдущий кейс	<p><b>Доступ к элементу;</b> определение значения словаря по ключу.</p> <p><b>Внимание:</b> для доступа к элементу не используем индексы, в отличие от списка, строки!</p>
<code>our_dict.keys()</code>	<pre>&gt;&gt;&gt; d = {'a':12, 'b':6} &gt;&gt;&gt; d.keys() dict_keys(['a', 'b']) &gt;&gt;&gt; dd = d.keys() &gt;&gt;&gt; type(dd) &lt;class 'dict_keys'&gt;</pre>	<p>Производит объект типа <b>dict_keys</b></p> <p>(в Python2.x – функция - возвращает список ключей)</p>
<code>our_dict.values()</code>	<pre>&gt;&gt;&gt; dv = d.values() &gt;&gt;&gt; dv dict_values([12, 6])</pre>	<p>Производит объект типа <b>dict_values</b></p> <p>(в Python2.x – функция - возвращает список значений)</p>
<code>our_dict.items()</code>	<pre>&gt;&gt;&gt; di=d.items() &gt;&gt;&gt; di dict_items([('a', 12), ('b', 6)]) &gt;&gt;&gt; type(di) &lt;class 'dict_items'&gt;</pre>	<p>Производит объект типа <b>dict_items</b></p> <p>(в Python2.x – функция - Возврат ключей и значений <b>в виде списка кортежей</b>)</p>
<code>our_dict.pop('our_key')</code>	<pre>&gt;&gt;&gt; d.pop('a') 12</pre>	Удаляет значение указанного ключа и возвращает удаленное значение
<code>our_dict.popitem()</code>	<pre>&gt;&gt;&gt; d {'cats': 13, 'rabbits': 56, 'raccoons': 1} &gt;&gt;&gt; d.popitem() ('cats', 13) &gt;&gt;&gt; d {'rabbits': 56, 'raccoons': 1}</pre> <pre>&gt;&gt;&gt; d {'rabbits': 56, 'raccoons': 1} &gt;&gt;&gt; d["cats"] Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; KeyError: 'cats'</pre>	<p>Удаляет и возвращает первый элемент <b>в виде кортежа</b> (ключ и значение) в произвольном порядке (первый элемент при каждом вызове функции определяется случайным образом).</p>
<code>our_dict.get('our_key', default_value)</code>	<p>default_value – необязательный параметр</p> <pre>&gt;&gt;&gt; d {'rabbits': 56, 'raccoons': 1} &gt;&gt;&gt; d["cats"] Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; KeyError: 'cats' &gt;&gt;&gt; d.get("cats") None &gt;&gt;&gt; print d.get("cats") None &gt;&gt;&gt; d.get("cats", 0) 0 &gt;&gt;&gt;</pre> <p>знач по умолч, если нет ключа так ого</p>	<p>При обращении к несуществующему ключу через геттер ошибка не возникает – возвращается пустое значение (если не указано дефолтное значение, т.е. значение по умолчанию); при обращении к нему напрямую генерируется исключение:</p> <pre>&gt;&gt;&gt; d.get("raccoons", 0) 1</pre>
<code>our_dict.has_key('our_key')</code>	!!!нет в Python 3.x!!!	<p>Использовать <b>in</b></p> <p>Проверка наличия определенного ключа.</p>

		Возвращает значение булевого типа
<code>dict1.update(dict2)</code>	<pre>&gt;&gt;&gt; dict1 = {'key1': 'value1', 'key2': 'value2'} &gt;&gt;&gt; dict2 = {'key3': 'value3', 'key4': 'value4'} &gt;&gt;&gt; dict1.update(dict2) &gt;&gt;&gt; dict1 {'key3': 'value3', 'key2': 'value2', 'key1': 'value1', 'key4': 'value4'} &gt;&gt;&gt; dict2 {'key3': 'value3', 'key4': 'value4'}</pre>	Объединение словарей
Добавление элемента в список		<pre>&gt;&gt;&gt; d {'cats': 13, 'dogs': 5} &gt;&gt;&gt; d['raccoons'] Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; KeyError: 'raccoons' &gt;&gt;&gt; d['raccoons'] = 1 &gt;&gt;&gt; d {'cats': 13, 'dogs': 5, 'raccoons': 1} &gt;&gt;&gt; d['raccoons'] 1</pre>
Преобразование списка списков к типу словарь: правильная запись	<pre>&gt;&gt;&gt; dict(((1,2),))</pre>	Неправильная запись: <pre>&gt;&gt;&gt; dict(((1,2))) Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; TypeError: cannot convert dictionary update sequence element #0 to a sequence &gt;&gt;&gt; dict((1,2)) Traceback (most recent call last):   File "&lt;stdin&gt;", line 1, in &lt;module&gt; TypeError: cannot convert dictionary update sequence element #0 to a sequence</pre>

ВАЖНО: при переборе элементов словаря возвращаются ключи:

```
>>> dict
{'key2': 'value2', 'key1': 'oops'}
>>> for o in dict:
...     print o
...
key2
key1
```

### Демонстрация изменяемости словаря

Обращаясь к элементу словаря, меняем его:

```
>>> d = {'cats': 11, 'dogs': 5}
>>> d['cats'] += 2
>>> d
{'cats': 13, 'dogs': 5}
```

ВАЖНО: нужно быть осторожным при работе с копией словаря: как и в случае со списками, исходный экземпляр изменяется при изменении копии:

```
>>> dict = {'key1': 'value1', 'key2': 'value2'}
>>> dict2 = dict
>>> dict2['key1'] = 'other_value'
>>> dict2
{'key2': 'value2', 'key1': 'other_value'}
>>> dict
{'key2': 'value2', 'key1': 'other_value'}
```

## Тип данных ЛОГИЧЕСКИЙ (Bool)

### Описание

В языке Python "ложь" (False) представлена целочисленным значением 0, "истина" (True) целочисленным значением 1:

```
>>> int(True)
1
>>> int(False)
0
```

Кроме того, интерпретатор Python распознает любую пустую структуру данных как "ложь", а любую непустую структуру данных - как "истину".

- числа, отличные от нуля, являются "истинной"
- другие объекты являются "истинной", если они не пустые

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(-8)
True
>>> bool('')
False
>>> bool('spam')
True
>>> bool([])
False
>>> bool([1])
True
>>> bool({})
False
>>> bool({'a': 1})
True
>>>
>>> 
```

Любые объекты языка Python поддерживают операции сравнения:

- ">" - больше
- "<" - меньше
- ">=" - больше или равно
- "<=" - меньше или равно
- "==" - равно
- "!=" - не равно

```

>>>
>>> x = 2 + 2
>>> x == 4
True
>>> x == 5
False
>>> x != 5
True
>>> x > 5
False
>>> x < 5
True
>>> x >= 4
True
>>> x <= 4
True
>>> 

```

Python сравнивает типы следующим образом:

- числа сравниваются по величине;
- строки сравниваются лексикографически, символ за символом;
- при сравнении списков и кортежей сравниваются все компоненты слева направо;
- словари сравниваются как отсортированные списки (ключ, значение).
- **Важно:** в Python3.x операторы сравнения `>`, `<`, `>=`, `<=` вызывают исключение `TypeError` при

сравнении разных типов данных.

```

>>> 1 == '1'
False
>>> 1 != '1'
True
>>> 1 > '1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() > str()

```

Ниже приведены **логические операторы** в порядке уменьшения приоритета. Операторы `or` и `and` всегда возвращают один из своих операндов. Причем второй операнд вычисляется, только если это необходимо для получения результата.

`not x` - если `x` ложно вернет `True`, иначе `False`

`x and y` - если `x` ложно вернет `x`, иначе `y`

`x or y` - если `x` ложно вернет `y`, иначе `x`

```

>>>
>>> not []
True
>>> not 4
False
>>> [1] or [2]
[1]
>>> [] or [2]
[2]
>>> [1] and [2]
[2]
>>> [] and [2]
[]
>>> 

```

## Выдержки из лекции

Тип данных Bool неизменяем.

Логический тип в Python:

```
>>> bool
<class 'bool'>
```

Сравнение буквенных значений (в порядке возрастания по алфавиту):

```
>>> 's' <= 'sd'
True
>>> 's' <= 'd'
False
```

Проверка (обратить внимание):

```
>>> bool(0)
False
```

Возврат кода символа согласно ASCII-таблице (сравнение происходит по данному коду):

```
>>> ord('s')
115
>>> ord('d')
100
```

Некоторые функции, с которыми мы же сталкивались в рамках курса, возвращали буль. Например, оператор проверки наличия (вхождения) символа, подстроки **in**:

```
>>> s = 'cat'
>>> 'c' in s
True
>>> 'ca' in s
True
>>> 'cs' in s
False
>>> 'cs' in s
False
```

Другой пример — проверка на digit/alpha:

```
>>> s = '123'
>>> s.isdigit()
True
>>> s = '123dfg'
>>> s.isdigit()
False
```

Преобразование значимых и незначимых строк к булю:

```
>>> bool('a')
True
>>> bool('')
False
>>> bool(0)
False
>>> bool([])
False
>>> bool(())
False
```

Оператор not:

```
>>> not True
False
>>> not False
True
>>> not(1==0)
True
```



ВАЖНО: при проверке а-ля «не неправда» рекомендуется использовать конструкцию **not False**, а не конструкцию **!= False**.

### Дополнительный источник

Особенности операторов **and** и **or** [http://ru.diveintopython.net/apihelper\\_andor.html](http://ru.diveintopython.net/apihelper_andor.html)



Оригинал: **Встроенные возможности языка Python**

### builtins-s

Некоторые типы данных, функции и переменные всегда доступны интерпретатору и могут использоваться внутри любого модуля. Чтобы получить доступ к этим функциям, не требуется импортировать дополнительные модули, тем не менее все они содержатся внутри модуля `__builtins__`

Просмотр всех встроенных возможностей языка: `>>> dir(__builtins__)`

Более подробное описание и использование встроенных функций можно прочитать на **Модуль math**:

```
>>> import math
>>> math
<module 'math' (built-in)>
>>> math.pi
3.141592653589793
```

Импортированные модули уже упоминались в лекциях (например, модуль `random`).

### Некоторые часто используемые функции и конструкции

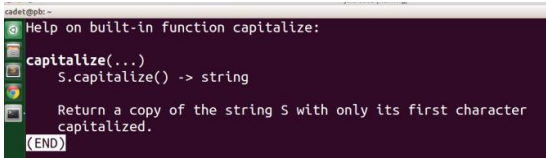
*our\_list* – пользовательские элементы

Функция, оператор	Фрагмент кода	Описание
<code>our_element.isdigit()</code>	Аналог: <code>if type(i) is int</code>	Является ли элемент числом (символы – только цифры). Чтобы понять, можем ли мы безопасно преобразовывать строку в число.
<code>our_element.isalpha()</code>		Является ли элемент строкой (символы – только буквы)
преобразования	<pre>&gt;&gt;&gt; int(12.0) 12 &gt;&gt;&gt; int(12.1) 12 &gt;&gt;&gt; bool([]) False &gt;&gt;&gt; bool({}) False &gt;&gt;&gt; bool([1]) True &gt;&gt;&gt; str(1) '1' &gt;&gt;&gt; str(42) '42' &gt;&gt;&gt; str(True) 'True' &gt;&gt;&gt; str([1,2,3,]) '[1, 2, 3]'</pre>	простейшие преобразования
<code>tuple(l)</code>	<pre>&gt;&gt;&gt; l [1, 2, 3, 's'] &gt;&gt;&gt; tuple(l) (1, 2, 3, 's')</pre>	преобразование l к типу кортеж

dict(l)	<pre>&gt;&gt;&gt; dict( ((1,2), (3,4)) ) {1: 2, 3: 4} &gt;&gt;&gt; d = dict( ((1,2), (3,4)) ) &gt;&gt;&gt; d.keys() [1, 3] &gt;&gt;&gt; d.items() [(1, 2), (3, 4)] &gt;&gt;&gt; d = dict( ((3,4), (1,2)) ) &gt;&gt;&gt; d {1: 2, 3: 4} &gt;&gt;&gt; d.items() [(1, 2), (3, 4)]</pre>	«Преобразование» к типу словарь (скорее, инициализация – конструктор словаря): в аргумент функции должен попасть итерируемый тип данных.
list(l)	<pre>&gt;&gt;&gt; t (1, 2, 3, 's') &gt;&gt;&gt; list(t) [1, 2, 3, 's']</pre>	преобразование к типу список
chr(число)	<pre>&gt;&gt;&gt; chr(60) '&lt;' &gt;&gt;&gt; chr(51) '3' &gt;&gt;&gt; chr(50) '2'</pre>	возвращает символ кода
ord(символ)	<pre>&gt;&gt;&gt; ord('a') 97</pre>	преобразование, обратное chr; получить код символа
тип данных NoneType (None)	<pre>&gt;&gt;&gt; type(None) &lt;class 'NoneType'&gt;</pre>	<p>Есть ряд стандартных функций, которые возвращают объект типа None – к примеру, функции модификации:</p> <pre>&gt;&gt;&gt; l = [1,1] &gt;&gt;&gt; l.append(2) &gt;&gt;&gt; print(l) [1, 1, 2] &gt;&gt;&gt; print(l.append(2)) None</pre> <p>Аналог void языков Java, C#... (отсутствие возвращаемого типа).</p>
our_element.count('j')	<pre>&gt;&gt;&gt; s = 'mmm' &gt;&gt; s.count('mm') 1</pre>	расчет вхождения j в our_element по очереди, без пересечений
%s		плейсхолдер для <b>строки</b>
%d	<pre>print ("Square is %d" % s)</pre>	<p>плейсхолдер для <b>числа</b></p> <p>в примере: s – некая переменная числового типа</p>
//	<pre>&gt;&gt;&gt; 3.0 // 2.0 1.0</pre>	операция целочисленного деления (деление с округлением вниз)
Оператор in	<p>Пример со строкой:</p> <pre>&gt;&gt;&gt; s = 'cat' &gt;&gt;&gt; 'c' in s True</pre> <p>Со списком:</p> <pre>&gt;&gt;&gt; l = [1,4,6] &gt;&gt;&gt; l [1, 4, 6] &gt;&gt;&gt; 1 in l True</pre> <p>С ключами словаря:</p>	Возвращает bool.

	<pre>&gt;&gt;&gt; d = {} &gt;&gt;&gt; 'cat' in d False</pre>	
--	--	--

### Как выводить справку

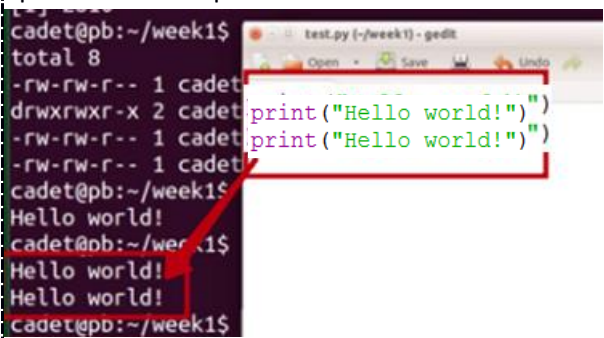
Возможность	Фрагмент кода	Описание
модуль <code>__builtins__</code>	<code>dir(__builtins__)</code>	узнать все встроенные возможности языка
<code>dir(s)</code>	<pre>&gt;&gt; s = 'fff' &gt;&gt; dir(s) ...</pre>	выводит доступные функции над типом данных, указанным в аргументе <code>dir</code> (тип данных определяется по типу данных переменной, переданной в функцию)
<code>help(function)</code>	<pre>&gt;&gt;&gt; s = 'hello world' &gt;&gt;&gt; s.capitalize() 'Hello world' &gt;&gt;&gt; help(s.capitalize)</pre>	Вывод справки о конкретной функции: 
<code>__doc__</code>	см.; справка конкретной функции	<pre>In [33]: random.randint.__doc__ Out[33]: 'Return random integer in range [a, b], including both end points.'</pre>

### Ubuntu Shell: некоторые команды

Команда	Фрагмент кода	Описание
<code>touch</code>		создать файл
<code>rm</code>		удалить файл
<code>rm -r</code>		удалить директорию
<code>mkdir</code>	<code>mkdir folder</code>	создать директорию
<code>cd</code>	<pre>cd folder cd ..</pre>	Изменить директорию Выйти из директории
	<code>D:</code>	переключиться на другой диск
<code>ls</code>		вывести существующие поддиректории
<code>ls -l</code>		вывести существующие поддиректории в виде списка
<code>mv</code>	<code>mv test.py test1/test_new.py</code>	переместить файл, при этом файл будет переименован
<code>cp</code>		скопировать файл
<code>mv</code>	<code>mv test.py test1/test_new.py</code>	Переместить и/или переименовать
<code>gedit</code>	добавить знак имперсанта <code>&amp;</code> , чтобы редактор был «оторван» от консоли (с целью «не засорять» консоль)	открыть указанный файл в редакторе <code>gedit</code>

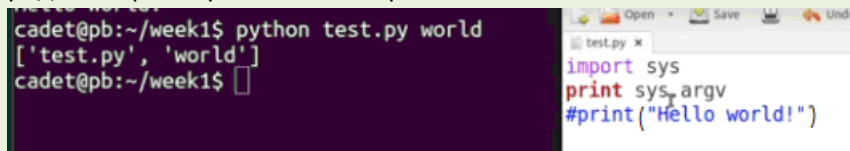
exit()		выйти из интерпретатора python
ctrl + c		прервать процесс

### Передача параметров из консоли

Команда / конструкция	Фрагмент кода	Описание
python	python test.py <i>#запуск исполнения кода файла test.py в консоли</i>	запуск интерпретатора питона; если указать файл – запустится выполнение кода, прописанного в файле. 
sys.argv	import sys print(sys.argv)	Запуск питон-скрипта из консоли

### Работа с кодом лекции: код (справа) - вывод (слева)

Позиционная передача параметров в питон-скрипт:



**ВНИМАНИЕ:** передаваемые с консоли параметры – строчного типа! Для операций с параметрами необходимо преобразование к числовому типу в явном виде:



Дополнительная проверка параметров, введенных в консоли; выход из программы:

```

cadet@pb:~/week1$ python test.py 42sm
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    r = int(sys.argv[1])
ValueError: invalid literal for int() with
cadet@pb:~/week1$ python test.py 42sm
['test.py', '42sm']
cadet@pb:~/week1$

```

```

import sys
import math
print sys.argv
r = sys.argv[1]
if r.isdigit():
    r = int(sys.argv[1])
else:
    exit(1)
s = math.pi * r**2
print ('Square: %d" % s)

```

```

test.py
import sys
import math
print sys.argv
r = sys.argv[1]
if r.isdigit():
    r = int(sys.argv[1])
else:
    exit(1)
s = math.pi * r**2
print ('Square: %d" % s)

```

Ремарка: Добавили else, иначе – безусловный выход:

Пример обращения ко второму элементу списка sys.argv; обратить внимание на форматирование строки:

```

cadet@pb:~/week1$ python test.py world
hello world!

```

```

test.py
import sys
w = sys.argv[1]
print ("Hello %s!" % w)

```

If statement, Циклы while и for

Функция, оператор	Фрагмент кода	Описание
pass	<p>Другие statements:</p> <p><a href="https://docs.python.org/3/reference/simple_stmts.html">https://docs.python.org/3/reference/simple_stmts.html</a></p> <ul style="list-style-type: none"> <li>6. Simple statements <ul style="list-style-type: none"> <li>6.1. Expression statements</li> <li>6.2. Assignment statements</li> <li>6.3. The assert statement</li> <li>6.4. The pass statement</li> <li>6.5. The del statement</li> <li>6.6. The print statement</li> <li>6.7. The return statement</li> <li>6.8. The yield statement</li> <li>6.9. The raise statement</li> <li>6.10. The break statement</li> <li>6.11. The continue statement</li> <li>6.12. The import statement</li> <li>6.13. The global statement</li> <li>6.14. The exec statement</li> </ul> </li> </ul>	«Пустой оператор»; не делает ничего. Полезен, к примеру, при перехвате ошибок: если ловим исключение, программа будет продолжать свою работу, если в except-блоке указано утверждение pass.
continue		Прерывание итерации цикла; цикл продолжается, даже когда прерывается выполнение определенной итерации цикла

		(осуществляется переход к следующей итерации). Цикл не прекращается. Выражение используется только в работе с циклами!
break		Прерывание цикла. Выражение используется только в работе с циклами!
«распаковка» (подробнее и расширенное о ней – в лекциях Недели № 2)	<pre>&gt;&gt;&gt; a, b = 1, 2 &gt;&gt;&gt; a 1 &gt;&gt;&gt; b 2 &gt;&gt;&gt; a, b = (1, 2) &gt;&gt;&gt; a, b = [1, 2] &gt;&gt;&gt; a, b = 'qa' &gt;&gt;&gt; a 'q' &gt;&gt;&gt; b 'a'</pre>	<p>«Коллективное» присваивание переменных</p> <p>Правило распаковки:</p> <pre>for i,k in ((1,2), (3,4)):     print(i,k)</pre> <pre>cadet@pb:~/week15 python for_enum.py 1 2 3 4</pre>
enumerate	<pre>for s in list(enumerate(string_word)):     print(s)</pre> <p>После преобразования:</p> <pre>&gt;&gt;&gt; list(enumerate("qwerty")) [(0, 'q'), (1, 'w'), (2, 'e'), (3, 'r'), (4, 't'), (5, 'y')] &gt;&gt;&gt;</pre> <p>Избавление от «инкремента вручную»:</p> <pre>word = 'qwerty' for i,s in list(enumerate(word)):     print(i, s)</pre> <p>В следующем случае <b>не нужно брать значение по индексу</b> (mauvais ton!)</p> <pre>word = 'qwerty' for i,s in list(enumerate(word)):     print(i, s, word[i]) # не стоит!</pre>	<p>Возвращает список кортежей; в каждом кортеже – номер символа и символ из переданного в функцию строчного типа; избавились от необходимости «ручного» инкремента</p> <p>Альтернатива – «ручной инкремент», «от чего избавились» (лучше использовать enumerate!):</p> <pre>cadet@pb:~/week15 python for_enum.py i = 0 for s in 'qwerty':     print(i,s)     i += 1</pre> <pre>0 q 1 w 2 e 3 r 4 t 5 y</pre>
Пользовательский ввод: <code>input()</code>	<pre>secret_world = 'monty' word_in = None while secret_world != word_in:     word_in = input("Enter word: ")</pre>	Пользовательский ввод имеет строковый тип; при необходимости, необходимо преобразовывать к числовому ( <code>int(input())</code> )

Проверка типа данных <b>type()</b> (применение)	<pre> secret_world = 'monty' word_in = None while secret_world != word_in:     word_in = input("Enter word: ")  divided_by_two = [] divided_by_three = []  for item in (2,4,7,1,'45', 5.87,3,6):     if type(item) is not int:         break     if item % 2 == 0:         divided_by_two.append(item)     if item % 3 == 0:         divided_by_three.append(item) else:     print(divided_by_two)     print(divided_by_three) </pre>	Демонстрация бизнес-логики: проверка типа данных элемента последовательности (кортежа).
---	---	---



## Перехват ошибок

Функция, оператор, «фича»	Фрагмент кода	Описание
<b>try,</b> <b>except</b>	<p>исхепт <b>IndexError</b>:</p> <p>(не перехватит ошибку нейма и другие ошибки; перехватит только ошибку индекса); в нашем примере ошибка возникает, если не введено необходимое значение для sys.argv. Пример «перехвата»:</p> <pre>try:     r = sys.argv[1] except IndexError:     print("Error!")     exit(1)</pre> <p>В Python реализован «перехват»:</p> <p><b>ValueError</b></p> <p><b>NameError</b></p> <p><b>KeyError</b></p> <p><b>SyntaxError</b></p> <p><b>ZeroDivisionError</b></p> <p><b>TypeError</b></p> <p><b>UnboundLocalError</b></p> <p><b>IndentationError</b></p> <p><b>AttributeError</b></p> <p><b>NotImplementedError</b></p> <p><b>IOError</b></p> <p><b>ImportError</b></p> <p><b>AccessError</b></p> <p><b>SecurityError</b></p> <p><b>ArgumentError</b></p> <p><b>UnicodeEncodeError</b></p> <p><b>UnicodeError</b></p> <p><b>RuntimeError</b></p> <p>...</p> <p><b>Полезные</b></p> <p><b>ссылки:</b> <a href="https://docs.python.org/3/library/exceptions.html">https://docs.python.org/3/library/exceptions.html</a></p> <p><a href="https://docs.python.org/3/tutorial/errors.html">https://docs.python.org/3/tutorial/errors.html</a></p>	<p>Предназначение: локализовать точки программы, в которых могут возникнуть ошибки, и перехватить эти ошибки (и сразу совет: не нужно помещать в блок трай-ексепт большой блок кода).</p> <pre>import sys import math  try:     r = sys.argv[1] except IndexError:     print("Error!")  if r.isdigit():     r = int(r) else:     print("radius is not number!")     exit(1)  s = math.pi * r**2 print("Square: %d" % s)</pre> <p>Обращение к оригинальной ошибке и вывод ее текста – при помощи конструкции <b>as e</b> :</p> <pre>try:     r = sys.argv[1] except IndexError as e:     print("Error! ", e)</pre> <p>cadet@nb:~/week1\$ python exception.py Error! list index out of range</p>