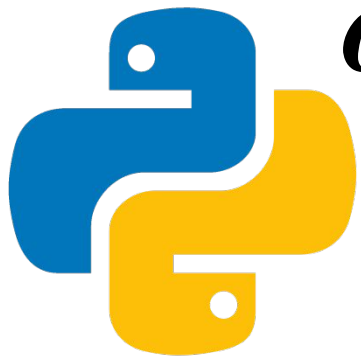


ТЕМА 2: Типы
данных языка
«Python» и
возможности их
применения



Объявление переменной в «Python»

Язык программирования «Python» относится к языкам с неявной сильной динамической типизацией.

```
>>> a=3
```

При инициализации переменной, на уровне интерпретатора происходит следующее:

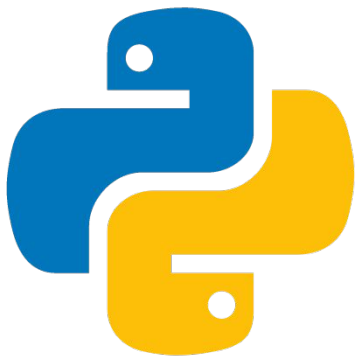
- 1) создается ячейка, в которую помещается цифра 3;
- 2) данный объект имеет некоторый идентификатор, значение которого равно 3, тип – целое число;
- 3) оператор «= \Rightarrow » создает ссылку между переменной и целочисленным объектом 3 (переменная ссылается на объект 3).

Для того, чтобы посмотреть на идентификатор объекта, который отражает уникальный адрес объекта, применяют функцию *id()*:

```
>>> id(a)  
263375008
```

Тип переменной можно определить, применяя функцию *type()*:

```
>>> type(a)  
<class 'int'>
```



Типы данных, применяемые в «Python»

1. None Type (неопределенное значение переменной) – объект со значением None, обозначающий отсутствие значения, следует отметить, что применительно к логическому контексту значение None интерпретируется как False;

2. Boolean Type – логический тип данных, обозначаемый через bool и принимающий значения False или True, которые ведут себя как числа 0 или 1;

3. Numeric Type – числовой тип данных, к которому относятся:

а) int – целые числа, размер которых может быть ограничен объемом оперативной памяти;

б) float – вещественные числа или числа с плавающей точкой (в качестве разделителя используется точка);

в) complex – комплексные числа.

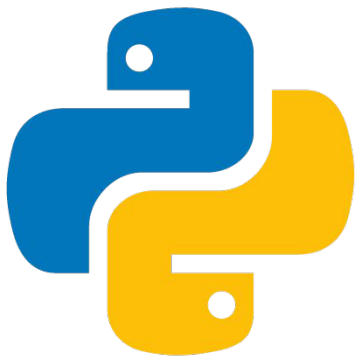
4. Sequence Type – тип данных список, который может быть представлен одним из следующих видов:

а) list – список;

б) tuple – кортеж;

в) range – диапазон.

5. Text Sequence Type – строковый тип данных, обозначаемый через str;



Типы данных, применяемые в «Python»

6. Binary Sequence Type – бинарные списки, включающие в себя:

а) **bytes** – байты;

б) **bytearray** – массивы байт;

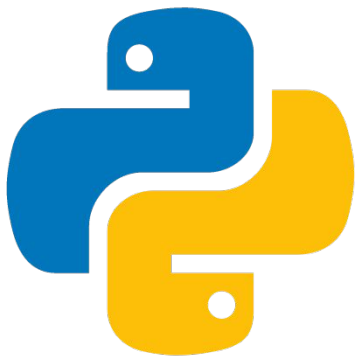
в) **memoryview** – специальные объекты, предназначенные для доступа к внутренним данным.

7. Set Type – тип данных множества, состоящий из:

а) **set** – множество;

б) **frozen set** – неизменяемое множество.

8. Mapping Types – тип данных словари, обозначаемый через **dict**.



Изменяемость и неизменяемость типов данных в «Python»

1. Изменяемые (*mutable*) типы данных, включающие в себя:

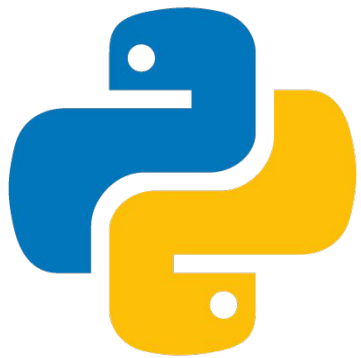
- а) списки (*list*);
- б) множества (*set*);
- в) словари (*dict*).

2. Неизменяемые (*immutable*) типы данных, к которым относятся:

- а) целые числа (*int*);
- б) числа с плавающей точкой (*float*);
- в) комплексные числа (*complex*);
- г) логические переменные (*bool*);
- д) кортежи (*tuple*);
- е) строки (*str*);
- ж) неизменяемые множества (*frozen set*).

Изменяемый тип данных позволяет менять значение объекта, например, создадим список [2, 5], после чего заменим второй элемент на 4:

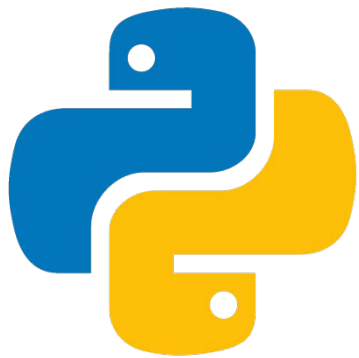
```
>>> a=[2, 5]
>>> id(a)
42312568
>>> a[1]=4
>>> a
[2, 4]
>>> id(a)
42312568
```



Арифметические операции, выполняемые с числовым типом данных

Арифметические операторы

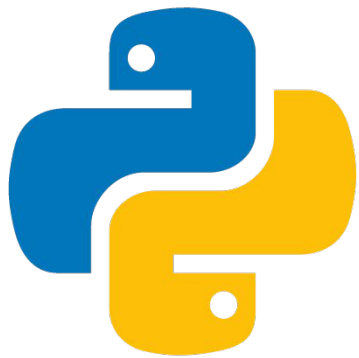
Оператор	Название	Пример	Результат
$a + b$	сложение	$10 + 5$	15
$a - b$	вычитание	$10 - 3$	7
$a * b$	умножение	$3 * 4$	12
a / b	деление	$6 / 2$	3
$a // b$	деление нацело	$7 // 3$	2
$a \% b$	остаток от деления	$7 \% 2$	1
$a ** b$	возведение в степень	$2 ** 3$	8



Арифметические операции, выполняемые с числовым типом данных

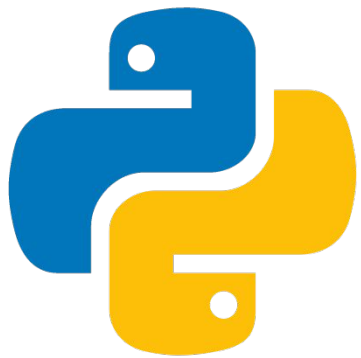
Операторы сравнения

Оператор	Описание
==	проверяет, равны ли оба операнда, если да, то условие становится истинным
!=	проверяет, равны ли оба операнда, если нет, то условие становится истинным
<>	проверяет, равны ли оба операнда, если нет, то условие становится истинным
>	проверяет, больше ли значение левого операнда, чем значение правого, если да, то условие становится истинным
<	проверяет, меньше ли значение левого операнда, чем значение правого, если да, то условие становится истинным
>=	проверяет, больше или равно значение левого операнда, чем значение правого, если да, то условие становится истинным
<=	проверяет, меньше или равно значение левого операнда, чем значение правого, если да, то условие становится истинным



Арифметические операции, выполняемые с числовым типом данных

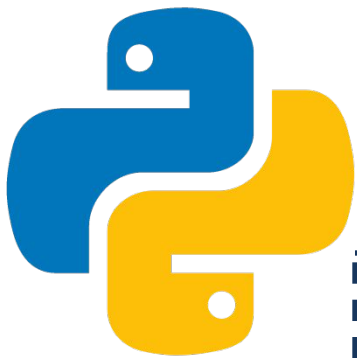
Операторы присваивания	
Оператор	Описание
=	присваивает значение правого операнда левому
+=	прибавит значение правого операнда к левому и присвоит эту сумму левому операнду
-=	отнимает значение правого операнда от левого и присваивает результат левому операнду
*=	умножает правый операнд с левым и присваивает результат левому операнду
/=	делит левый операнд на правый и присваивает результат левому операнду
%=	делит по модулю операнды и присваивает результат левому
**=	возводит в левый операнд в степень правого и присваивает результат левому операнду
//=	производит целочисленное деление левого операнда на правый и присваивает результат левому операнду



Арифметические операции, выполняемые с числовым типом данных

Функции для обработки чисел

Функция	Название	Пример	Результат
$abs(x)$	получение абсолютного значения (модуля) числа	$abs(-4)$	4
$round(x, n)$	округление числа, количество знаков задается параметром n , если он не указан, то округляется до целого	$round(3.85, 1)$	3.9



Перевод из одной системы счисления в другую «Python»

1) `int([object], [основание системы счисления])` – преобразование к целому числу в десятичной системе счисления, например:

```
>>> int('1001111', 2)
79
```

2) `bin(x)` – преобразование целого числа в двоичную систему счисления:

```
>>> bin(79)
'0b1001111'
```

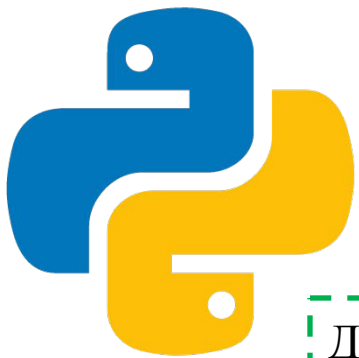
Здесь первая цифра **0** говорит о том, что это **не десятичное число**, а буква **b** говорит о том, что это число записано в двоичном формате (binary)

3) `hex(x)` – преобразование целого числа в шестнадцатеричную систему счисления:

```
>>> hex(76)
'0x4c'
```

4) `oct(x)` – преобразование целого числа в восьмеричную систему счисления:

```
>>> oct(125)
'0o175'
```

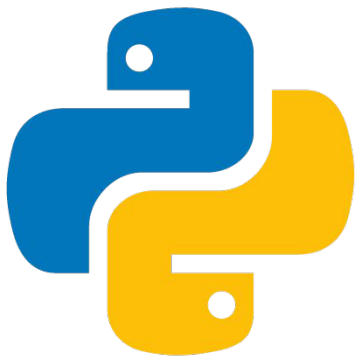


Работа с комплексными числами в «Python»

Для создания комплексного числа в языке «Python» используется функция `complex(a, b)`, в которую в качестве первого аргумента передается действительная часть, в качестве второго – мнимая, например: Создать комплексное число в «Python» можно следующими способами:

```
>>> z=1+2j  
>>> print(z)  
(1+2j)
```

```
>>> x=complex(5,8)  
>>> print(x)  
(5+8j)
```



Работа с комплексными числами в «Python»

Над комплексными числами можно выполнять следующие арифметические операции:

1. Сложение:

```
>>> z=1+2j
>>> x=5+8j
>>> x+z
(6+10j)
```

2. Вычитание:

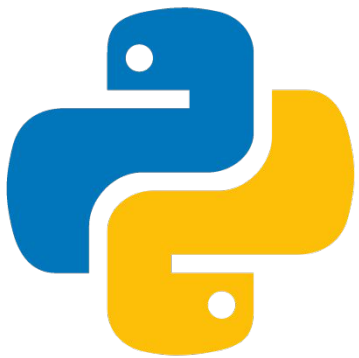
```
>>> z=1+2j
>>> x=5+8j
>>> x-z
(4+6j)
```

3. Умножение:

```
>>> z=1+2j
>>> x=5+8j
>>> x*z
(-11+18j)
```

4. Деление:

```
>>> z=1+2j
>>> x=5+8j
>>> x/z
(4.2-0.4j)
```



Работа с комплексными числами в «Python»

Над комплексными числами можно выполнять следующие арифметические операции:

5. Возведение в степень:

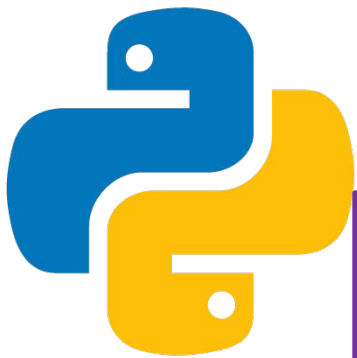
```
>>> z=1+2j
>>> x=5+8j
>>> x**z
(0.883722259595132-0.8783545387446259j)
```

6. Извлечение целой и мнимой части:

```
>>> x=5+8j
>>> x.real
5.0
>>> x.imag
8.0
```

7. Получение комплексно-сопряженного числа:

```
>>> x=5+8j
>>> x.conjugate()
(5-8j)
```



Работа с библиотекой «math» в «Python»

В состав стандартного пакета «Python» входит библиотека *math*, которая предоставляет обширный функционал для работы с числами. Для начала работы с данной библиотекой ее необходимо импортировать, применяя команду:

```
>>> import math
```

Функции библиотеки *math*:

1) *math.ceil(x)* – выводит ближайшее целое число, большее, чем x :

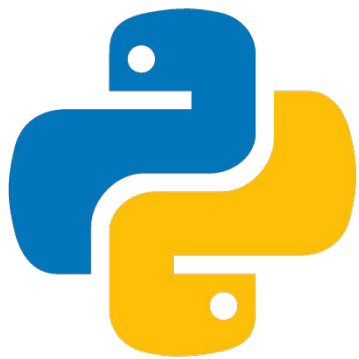
```
>>> math.ceil(8.3)
```

```
9
```

2) *math.floor(x)* – выводит ближайшее целое число, меньшее, чем x :

```
>>> math.floor(8.3)
```

```
8
```

Работа с библиотекой «math» в «Python»

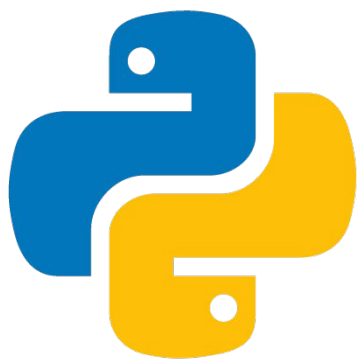
Функции библиотеки *math*:

3) *math.fabs(x)* – выводит абсолютное значение (модуль) числа x :

```
>>> math.fabs(-8.3)  
8.3
```

4) *math.factorial(x)* – вычисляет факториал числа x :

```
>>> math.factorial(3)  
6
```



Работа с библиотекой «math» в «Python»

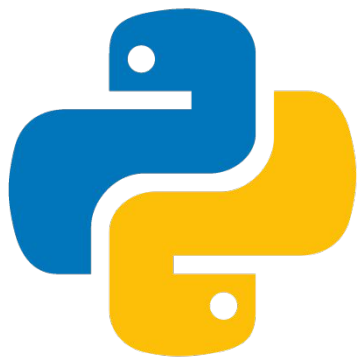
Функции библиотеки *math*:

5) *math.exp(x)* – вычисляет экспоненту числа x :

```
>>> math.exp(3)  
20.085536923187668
```

6) *math.log2(x)* – вычисляет логарифм числа x по основанию 2:

```
>>> math.log2(5)  
2.321928094887362
```



Работа с библиотекой «math» в «Python»

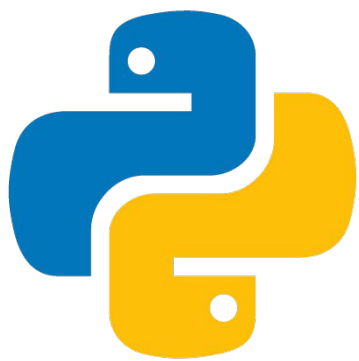
Функции библиотеки *math*:

7) *math.log10(x)* – вычисляет десятичный логарифм числа *x*:

```
>>> math.log10(4)
0.6020599913279624
```

8) *math.log(x[, base])* – вычисляет логарифм числа *x* по основанию *e*, однако дополнительно можно указать основание логарифма:

```
>>> math.log(5)
1.6094379124341003
>>> math.log(4, 8)
0.6666666666666667
>>> math.log2(8)
3.0
>>> math.log10(10000)
4.0
```



Работа с библиотекой «math» в «Python»

Функции библиотеки *math*:

9) *math.pow(x, y)* – вычисляет числа x в степени y :

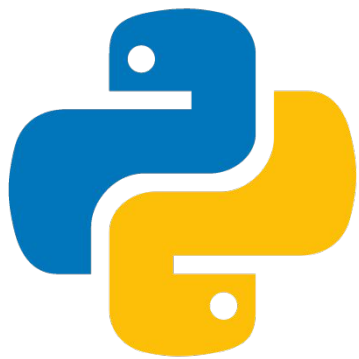
```
>>> math.pow(2, 3)  
8.0
```

10) *math.sqrt(x)* – вычисляет квадратный корень числа x :

```
>>> math.sqrt(16)  
4.0
```

11) *math.pi* – вычисляет значение константы числа π :

```
>>> math.pi  
3.141592653589793
```



Работа с библиотекой «math» в «Python»

Функции библиотеки *math*:

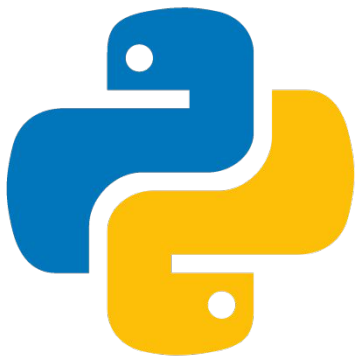
12) *math.e* – вычисляет значение экспоненты:

```
>>> math.e  
2.718281828459045
```

Для удобства вычисления значений перечисленных тригонометрических функций в языке программирования «Python» существует функция *math.degrees()*, позволяющая переводить радианы в градусы, а также функция *math.radians()*, выполняющая перевод градусов в радианы

13) *math.sin()*, *math.cos()*, *math.tan()*, *math.asin()*, *math.acos()*, *math.atan()* – вычисляют значения тригонометрических функций: синуса, косинуса, тангенса, арксинуса, арккосинуса и арктангенса соответственно, при этом значения указываются в радианах.

```
>>> math.radians(90) # перевода 90 градусов в радианы  
1.5707963267948966  
>>> math.sin(1.5707963267948966) # синус 90 градусов  
1.0
```

Работа с типом данных «список» в «Python»

Список (*list*) в «Python» – это упорядоченные изменяемые коллекции объектов произвольных типов. Чтобы работать со списками, их первоначально необходимо создать, что может быть реализовано одним из следующих способов:

1) посредством обработки любого объекта встроенной функцией

list():

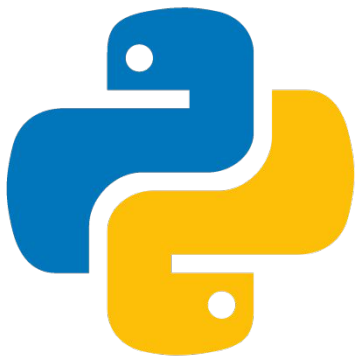
```
>>> list('информатика')  
['и', 'н', 'ф', 'о', 'р', 'м', 'а', 'т', 'и', 'к', 'а']
```

2) при помощи литерала, список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего:

```
>>> s=[] #пустой список  
>>> l=['и', 'н', ['форматик'], 'а']  
>>> s  
[]  
>>> l  
['и', 'н', ['форматик'], 'а']
```

3) с применением генератора списков (генераторы списков очень похожи на цикл for):

```
>>> s=[s*2 for s in 'курс']  
>>> s  
['кк', 'уу', 'рр', 'сс']
```

Работа с типом данных «список» в «Python»

Методы, применяемые при работе со списками:

1) `append()` – служит для добавления элемента в конец списка:

```
>>> a=[2, 5, 1, 4]
>>> a.append(8)
>>> a
[2, 5, 1, 4, 8]
```

2) `extend(L)` – дублирует список:

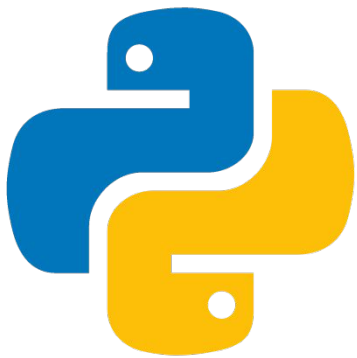
```
>>> a=[2, 5, 1, 4]
>>> a.extend(a)
>>> a
[2, 5, 1, 4, 2, 5, 1, 4]
```

3) `insert(i, x)` – вставляет на `i`-ый элемент значение `x`:

```
>>> a=[2, 5, 1, 4]
>>> a.insert(0, 9)
>>> a
[9, 2, 5, 1, 4]
```

4) `remove(x)` – удаляет первый элемент в списке, имеющий значение `x`, если указанного элемента не существует, то выводится сообщение об ошибке:

```
>>> a=[2, 5, 1, 4]
>>> a.remove(1)
>>> a
[2, 5, 4]
```



Работа с типом данных «список» в «Python»

Методы, применяемые при работе со списками:

5) `pop(i)` – удаляет *i*-ый элемент и выводит его, если индекс не указан, удаляется последний элемент:

```
>>> a=[2, 5, 1, 4]
```

```
>>> a.pop(0)
```

```
2
```

```
>>> a
```

```
[5, 1, 4]
```

```
>>> a.pop()
```

```
4
```

```
>>> a
```

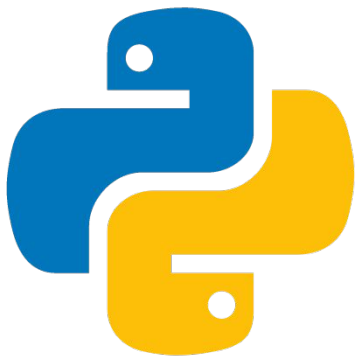
```
[5, 1]
```

6) `index(x)` – выводит порядковый номер первого элемента со значением *x*:

```
>>> a=[2, 5, 5, 1, 4, 5]
```

```
>>> a.index(5)
```

```
1
```



Работа с типом данных «список» в «Python»

Методы, применяемые при работе со списками:

7) count(x) – выводит количество элементов со значением *x*:

```
>>> a=[2, 5, 5, 1, 4, 5]
```

```
>>> a.count(5)
```

```
3
```

8) sort() – сортирует список по возрастанию:

```
>>> a=[2, 5, 5, 1, 4, 5]
```

```
>>> a.sort()
```

```
>>> a
```

```
[1, 2, 4, 5, 5, 5]
```

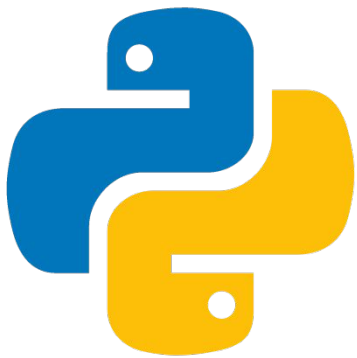
9) sort(reverse=True) – сортирует список по убыванию:

```
>>> a=[2, 5, 5, 1, 4, 5]
```

```
>>> a.sort(reverse=True)
```

```
>>> a
```

```
[5, 5, 5, 4, 2, 1]
```



Работа с типом данных «список» в «Python»

Методы, применяемые при работе со списками:

10) reverse() – позволяет перевернуть список:

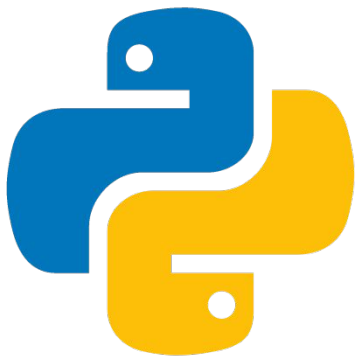
```
>>> a=[2, 5, 5, 1, 4, 5]
>>> a.reverse()
>>> a
[5, 4, 1, 5, 5, 2]
```

11) copy() – создает поверхностную копию списка:

```
>>> a=[2, 5, 5, 1, 4, 5]
>>> a.copy()
[2, 5, 5, 1, 4, 5]
```

12) clear() – очищает список:

```
>>> a=[2, 5, 5, 1, 4, 5]
>>> a.clear()
>>> a
[]
```



Работа с типом данных «кортеж» в «Python»

Преимущества кортежей (*tuple*) над строками заключается в том, что они обладают следующими преимуществами:

1. Кортежи нельзя изменять;
2. Кортежи имеют меньший размер.

Кортеж можно создать следующими способами:

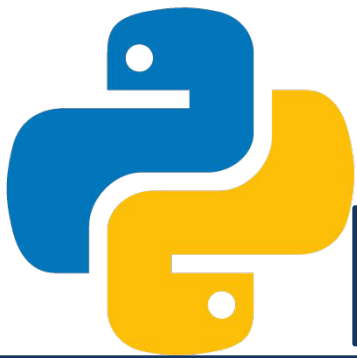
1) применяя встроенную функцию *tuple()*:

```
>>> a=tuple() #создаем пустой кортеж  
>>> a  
()
```

2) с помощью литерала кортежа:

```
>>> a=() #создаем пустой кортеж  
>>> a  
()  
>>> a=('информатика',)  
>>> a  
('информатика',)
```

Все операции, выполняемые со списками, при условии, что они не изменяют список, свойственны и кортежам



Работа с типом данных «диапазон» в «Python»

Диапазон (*range*), являясь универсальной функцией, позволяет создавать список, содержащий арифметическую прогрессию.

При этом функция `range()` может содержать от одного до трех аргументов: **старт, стоп и шаг**, аргументами должны быть как положительные, так и отрицательные числа, но обязательно целые. Все указывать необязательно, так как старт и шаг по умолчанию имеют значения 0 и 1 соответственно. Если задать только один аргумент, то задается конец диапазона – стоп.

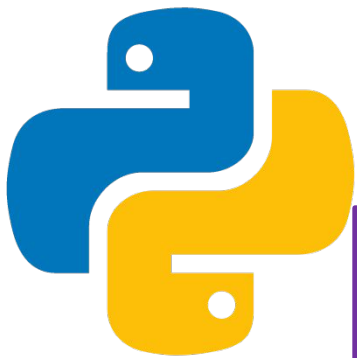
Для создания диапазона необходимо поступать одним из следующих способов (в зависимости от приложения):

```
>>> list(range(4, 28, 5))  
[4, 9, 14, 19, 24]
```

```
>>> range(4, 28, 5)  
[4, 9, 14, 19, 24]
```

Для списков, полученных посредством применения *range()* возможно применение функции *sum()*, которая вычисляет сумму элементов полученной прогрессии:

```
>>> sum(range(4, 28, 5))  
70
```

Работа со строковыми данными в «Python»

Строковые данные заключаются либо в одинарные, либо в двойные кавычки, при этом, если строка заключена в одинарные кавычки, то в ней не допускаются символы одинарных кавычек, и наоборот – если строка заключена в двойные кавычки, в ней не допускаются символы двойных кавычек.

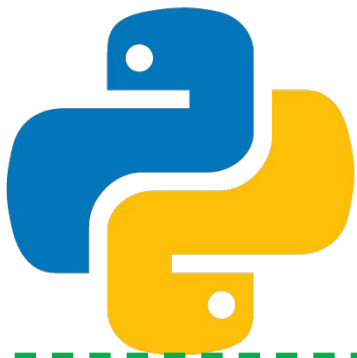
Примеры работы с данными строкового типа

1. Дублирования одинарных или двойных кавычек внутри данных, в этом случае перед повторяющимся символом нужно поставить знак «\», например:

```
>>> 'первая программа \'123'  
"первая программа '123"
```

2. Перевод строки осуществляется посредством комбинации символов \n:

```
>>> print ('первая\nпрограмма')  
первая  
программа
```



Работа со строковыми данными в «Python»

Примеры работы с данными строкового типа

3. К каждому символу строки можно обращаться как к элементу списка, при этом достаточно указать его индекс в квадратных скобках, следует помнить, что нумерация начинается с нуля:

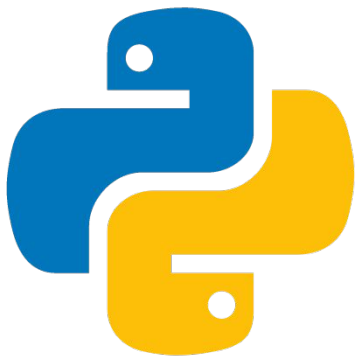
```
>>> s='мур'  
>>> s[0], s[1], s[2]  
('м', 'у', 'р')
```

4. Длину строки можно вычислить, применяя функцию `len()`, например:

```
>>> s='мур'  
>>> len(s)  
3
```

5. Существует возможность применения оператора конкатенации (является аналогичным оператору арифметического сложения). Таким образом, можно к концу одной строки присоединять другую строку, например:

```
>>> s='вы'  
>>> p='ручка'  
>>> s+p  
'выручка'
```



Работа со строковыми данными в «Python»

Примеры работы с данными строкового типа

6. Повторение строки (n-кратное):

```
>>> s='дисциплина'
```

```
>>> n=5
```

```
>>> s*n
```

```
'дисциплинадисциплинадисциплинадисциплинадисциплина'
```

7. Выбор из строки элемента с указанным порядковым номером, результатом является символ, при этом, если указывается порядковый номер больше нуля, то отсчет ведется с начала строки, если меньше – то с конца, в этом случае последний символ строки имеет порядковый номер равный «-1»:

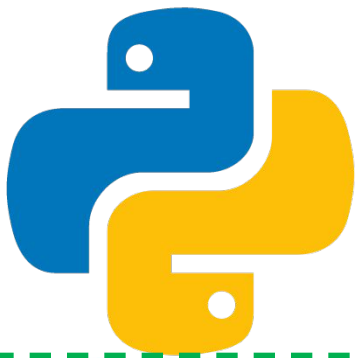
```
>>> s='польза'
```

```
>>> s[2]
```

```
'л'
```

```
>>> s[-3]
```

```
'б'
```



Работа со строковыми данными в «Python»

Примеры работы с данными строкового типа

8. Выделение подстроки, содержащей символы первоначальной строки с номерами от i до j и с шагом k . Если шаг не указывается, то символы идут подряд – это равносильно тому, что шаг равен 1, например:

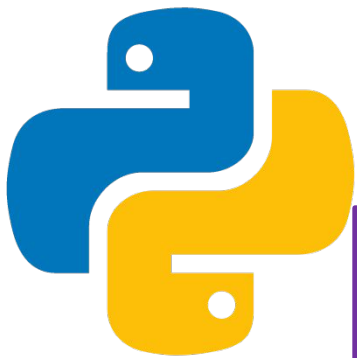
```
>>> s='дерево'  
>>> s[2:5]  
'pee'  
>>> s='дерево'  
>>> s[2:5:2]  
'pe'
```

9. Определение и вывод символа с наименьшим значением (кодом – номером в кодовой таблице):

```
>>> s='арка'  
>>> min(s)  
'a'
```

10. Определение и вывод символа с наибольшим значением (кодом – номером в кодовой таблице):

```
>>> s='арка'  
>>> max(s)  
'р'
```

Работа со строковыми данными в «Python»

Строки в «Python» обладают методами. Рассмотрим основные методы, предварительно оговорив, что строка, к которой применяются методы, называется s :

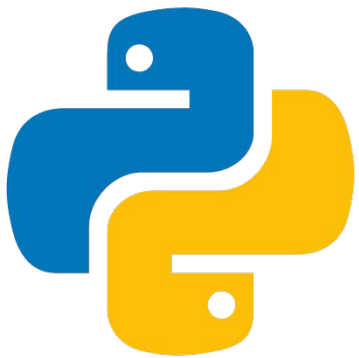
Основные методы при работе со строковыми данными:

1) `center(n)` – возвращение строки s , дополненной пробелами справа и слева до ширины в n символов, при этом исходная строка не изменяется, если $n \leq \text{len}(s)$ – пробелы не добавляются:

```
>>> s='информатика'  
>>> s.center(20)  
'   информатика   '
```

2) `ljust(n)` – выравнивание строки s по левому краю (дополняется пробелами справа) в пространстве, ширина которого составляет n символов, если $n < \text{len}(s)$ – пробелы не добавляются:

```
>>> s='информатика'  
>>> s.ljust(20)  
'информатика      '
```



Работа со строковыми данными в «Python»

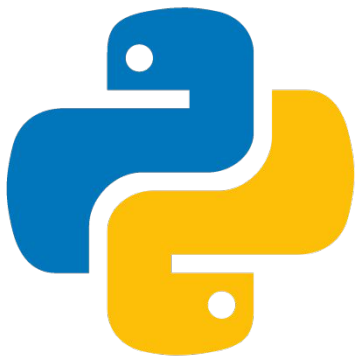
Основные методы при работе со строковыми данными:

3) `rjust(n)` – выравнивание строки `s` по правому краю (дополняется пробелами слева) в пространстве, ширина которого составляет `n` символов, если `n < len(s)` – пробелы не добавляются:

```
>>> s='информатика'  
>>> s.rjust(20)  
'      информатика'
```

4) `count(s1,i,j)` – определяет количество вхождений подстроки `s1` в строку `s`, результатом является число, здесь `i` и `j` – это позиции начала и конца поиска соответственно:

```
>>> s='химия'  
>>> s.count('u')  
2  
>>> s='химия'  
>>> s.count('я',1,3)  
0
```

Работа со строковыми данными в «Python»

Основные методы при работе со строковыми данными:

5) `find(s1,i,j)` – определяет позицию первого (считая слева) вхождения подстроки `s1` в строку `s`, результатом является число, здесь `i` и `j` – это позиции начала и конца поиска соответственно, в данном случае они необязательны:

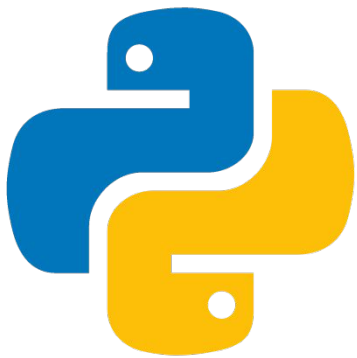
```
>>> s='химия'  
>>> s.find('я')  
4
```

6) `rfind(s1,i,j)` – определяет позицию последнего (считая слева) вхождения подстроки `s1` в строку `s`, результатом является число, здесь `i` и `j` – это позиции начала и конца поиска соответственно, в данном случае они также необязательны:

```
>>> s='химия'  
>>> s.rfind('x')  
0
```

7) `strip()` – создает строку, в которой удалены пробелы в начале и в конце (при их наличии):

```
>>> s='   корабль   '  
>>> s.strip()  
'корабль'
```



Работа со строковыми данными в «Python»

Основные методы при работе со строковыми данными:

8) `lstrip()` – создает строку, в которой удалены пробелы в начале (при их наличии):

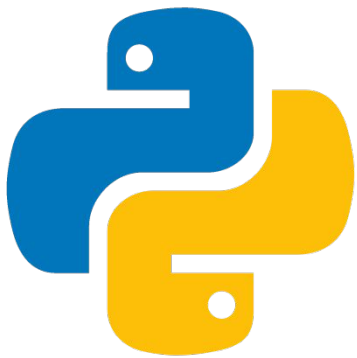
```
>>> s='   корабль '
>>> s.lstrip()
'корабль'
```

9) `rstrip()` – создает строку, в которой удалены пробелы в конце (при их наличии):

```
>>> s='   корабль '
>>> s.rstrip()
'   корабль'
```

10) `replace(s1,s2,n)` – создает строку, в которой фрагмент (подстрока) `s1` исходной строки заменяется на фрагмент `s2`, при этом необязательный аргумент `n` указывается количество замен (если требуется заменить не все фрагменты):

```
s='информатика'
>>> s.replace('a','z',1)
'информзтика'
```



Работа со строковыми данными в «Python»

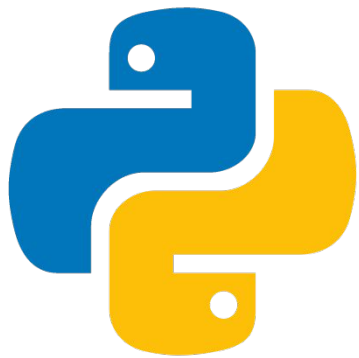
Основные методы при работе со строковыми данными:

11) `capitalize()` – создает строку, в которой первая буква исходной строки становится заглавной (приписной), а все остальные становятся маленькими (строчными):

```
>>> s='информатика'  
>>> s.capitalize()  
'Информатика'
```

12) `swapcase()` – создает строку, в которой прописные буквы заменяются на строчные, и наоборот – строчные на прописные:

```
>>> s='информатика'  
>>> s.swapcase()  
'ИНФОРМАТИКА'
```



Работа со строковыми данными в «Python»

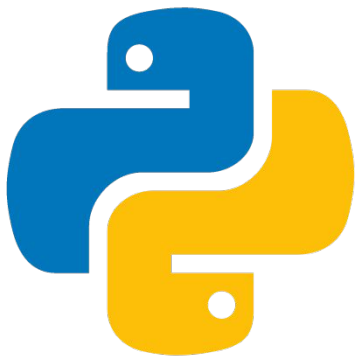
Основные методы при работе со строковыми данными:

13) `upper()` – создает строку, в которой все буквы исходной строки становятся заглавными (прописными):

```
>>> s='информатика'  
>>> s.upper()  
'ИНФОРМАТИКА'
```

14) `lower()` – создает строку, в которой все буквы исходной строки становятся маленькими (строчными):

```
>>> s='ИНФОРМАТИКА'  
>>> s.lower()  
'информатика'
```

Работа с множествами в «Python»

Множеством называется неупорядоченная последовательность уникальных элементов. В языке программирования «Python» множество объявляется при помощи функции `set()`:

```
>>> t=set()  
>>> t  
set()
```

Функция `set()` также позволяет осуществлять преобразование таких элементов последовательности во множество, как:

1) строки:

```
>>> set('string')  
{'r', 's', 'g', 'i', 't', 'n'}
```

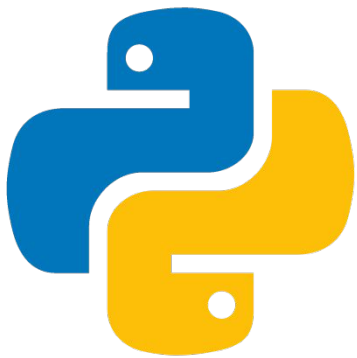
2) списка:

```
>>> set([1, 2, 3])  
{1, 2, 3}
```

3) кортежа:

```
>>> set((1, 2, 3, 4))  
{1, 2, 3, 4}
```

Следует отметить, что при преобразовании остаются только уникальные элементы.



Работа с множествами в «Python»

Преобразовать элементы множества позволяет цикл **for**:

```
>>> for i in set([1, 2]): print(i)  
1  
2
```

Получить количество элементов множества возможно при помощи функции `len()`:

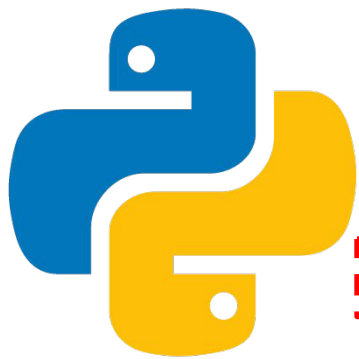
```
>>> len(set([1, 2, 3]))  
3
```

Язык программирования «Python» поддерживает генераторы множеств, Синтаксис которых схож с генераторами списков, отличие заключается лишь в том, что выражение заключается в фигурные скобки, а не в квадратные. Поскольку результатом является множество, то все повторяющиеся элементы удаляются:

```
>>> {x for x in [2, 3, 5, 2, 4, 6]}  
{2, 3, 4, 5, 6}
```

Рассмотрим пример создания множества, содержащего только уникальные четные элементы из исходного списка элементов:

```
>>> {x for x in [2, 3, 5, 2, 4, 6] if x%2==0}  
{2, 4, 6}
```

Работа с множествами в «Python»

Для работы с множествами предназначены следующие функции:

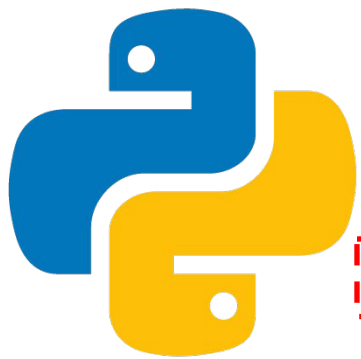
1. Объединение двух множеств посредством применения функции `union()`:

```
>>> t=set([1, 2, 3])
>>> t.union(set([5, 6, 8]), t|set([5, 6, 8]))
({1, 2, 3, 5, 6, 8}, {1, 2, 3, 5, 6, 8})
```

2. Добавление элементов одного множества к элементам другого множества с помощью функции `update()`:

```
>>> t=set([1, 2, 3])
>>> t.update(set([5, 6, 8]))
>>> t
{1, 2, 3, 5, 6, 8}
```

При этом, если элемент уже содержится в первом множестве, то он не будет повторно добавлен.



Работа с множествами в «Python»

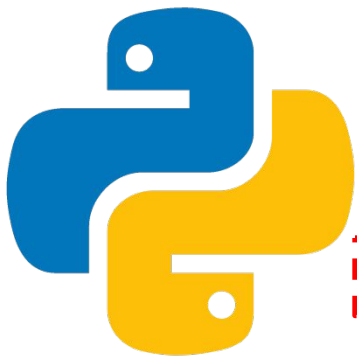
Для работы с множествами предназначены следующие функции:

3. Вычисление разницы между множествами (при этом выводятся элементы первого множества, которые отличаются от элементов второго):

```
>>> set([2, 8, 5]) - set([2, 3, 9])  
{8, 5}
```

4. Удаление элементов из первого множества, которые существуют и в первом и во втором множестве посредством применения функции `difference_update()`:

```
>>> t = set([1, 3, 5])  
>>> t.difference_update(set([1, 4, 5]))  
>>> t  
{3}
```



Работа с множествами в «Python»

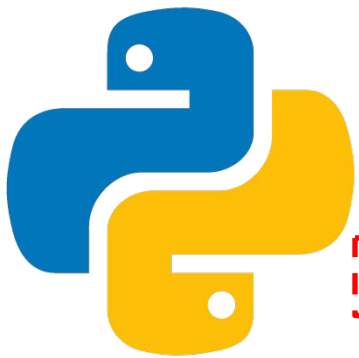
Для работы с множествами предназначены следующие функции:

5. Получение элементов, существующих и в первом и во втором множествах с помощью функции *intersection()* или символа «&»:

```
>>> t=set([2, 5, 8])
>>> t.intersection(set([2, 3, 9]))
{2}
>>> set([2, 5, 8])& set([2, 3, 9])
{2}
```

6. Формирование первого множества из элементов, которые существуют в обоих множествах с помощью функции *intersection_update()* или символа «&»:

```
>>> t=set([2, 5, 8])
>>> t.intersection_update(set([2, 3, 9]))
>>> t
{2}
>>> t=set([2, 5, 8])
>>> t&=set([2, 3, 9])
>>> t
{2}
```

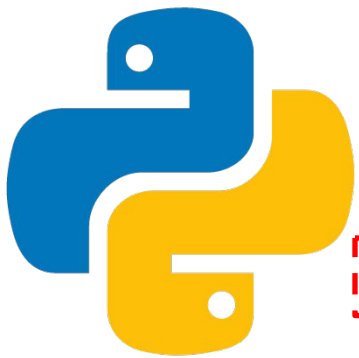


Работа с множествами в «Python»

Для работы с множествами предназначены следующие функции:

7. Возвращение всех элементов обоих множеств, за исключением тех, которые присутствуют и в первом и во втором множествах, для этого применяют функцию *symmetric_difference()*, при этом, если оба множества содержат одинаковые элементы, то в результате будут сформированы два пустых множества:

```
>>> t=set([2, 5, 8])
>>> t^set([2, 3, 9]), t.symmetric_difference(set([2, 3, 9]))
({3, 5, 8, 9}, {3, 5, 8, 9})
>>> t=set([2, 5, 8])
>>> t^set([2, 5, 8]), t.symmetric_difference(set([2, 5, 8]))
(set(), set())
```

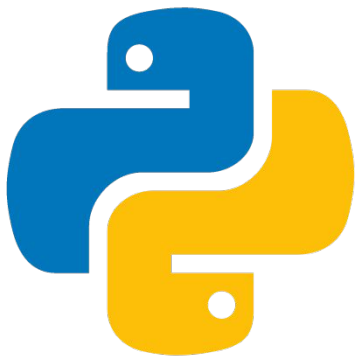



Работа с множествами в «Python»

Для работы с множествами предназначены следующие функции:

8. Составление первого множества из всех элементов обоих множеств, за исключением повторяющихся элементов, с помощью функции `symmetric_difference_update()` или символа «^»:

```
>>> t=set([2, 5, 8])
>>> t.symmetric_difference_update(set([2, 3, 9]))
>>> t
{3, 5, 8, 9}
>>> t=set([2, 5, 8])
>>> t^=set([2, 3, 9])
>>> t
{3, 5, 8, 9}
```

Работа с множествами в «Python»

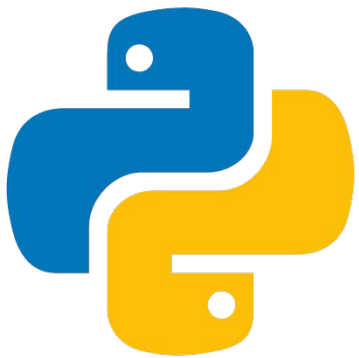
Также при работе с множествами используют операторы сравнения, такие как:

1) оператор «*in*», позволяющий проверять наличие элемента во множестве:

```
>>> t=set([1, 5, 9])  
>>> 8 in t, 1 in t  
(False, True)
```

2) оператор «*not in*», позволяющий проверять отсутствие элемента во множестве:

```
>>> t=set([1, 5, 9])  
>>> 8 not in t, 1 not in t  
(True, False)
```



Работа с множествами в «Python»

Также при работе с множествами используют операторы сравнения, такие как:

3) оператор «`==`», позволяющий выполнять проверку на равенство:

```
>>> set([1, 5, 9]) == set([1, 5, 9])
```

```
True
```

```
>>> set([1, 5, 9]) == set([2, 5, 9])
```

```
False
```

4) оператор «`<=`» или функция `issubset()` – выполняют проверку на предмет вхождения всех элементов первого множества во второе множество:

```
>>> t = set([1, 5, 9])
```

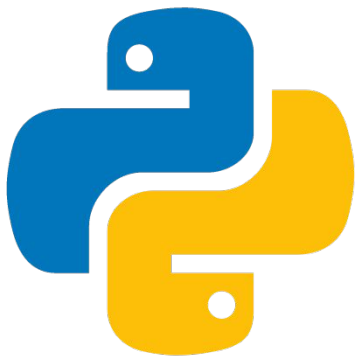
```
>>> t <= set([1, 5]), t <= set([1, 5, 9])
```

```
(False, True)
```

```
>>> t = set([1, 5, 9])
```

```
>>> t.issubset(set([1, 5]))
```

```
False
```



Работа с множествами в «Python»

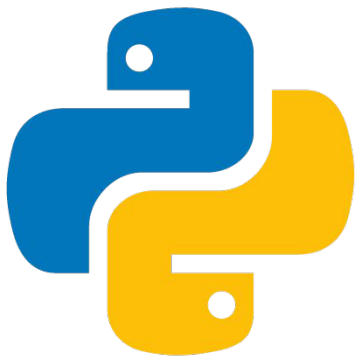
Также при работе с множествами используют операторы сравнения, такие как:

5) оператор «<» проверяет все ли элементы первого множества входят во второе множество, при условии, что первое и второе множества не равны между собой:

```
>>> t=set([1, 5, 9])  
>>> t<set([1, 5, 9]), t<set([2, 5, 9])  
(False, False)
```

6) оператор «>=» или функция `issuperset()` – выполняют проверку на предмет вхождения всех элементов второго множества в первое множество:

```
>>> t=set([1, 5, 9])  
>>> t>=set([1, 5]), t>=set([1, 5, 9])  
(True, True)  
>>> t=set([1, 5, 9])  
>>> t.issuperset(set([1, 5]))  
True
```



Работа с множествами в «Python»

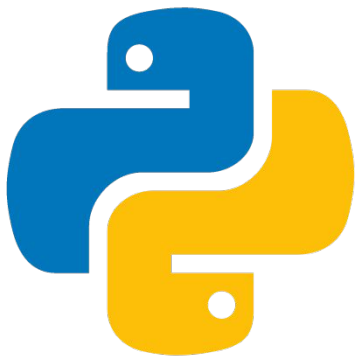
Также при работе с множествами используют операторы сравнения, такие как:

7) оператор «>» проверяет все ли элементы второго множества входят в первое множество, при условии, что первое и второе множества не равны между собой:

```
>>> t=set([1, 5, 9])  
>>> t>set([1, 5, 9]), t>set([2, 5, 9])  
(False, False)
```

8) функция `isdisjoint()` проверяют, являются ли два множества полностью разными, т.е. не содержат ни одного одинакового элемента:

```
>>> t=set([1, 5, 9])  
>>> t.isdisjoint([2, 1, 9])  
False
```

Работа с множествами в «Python»

Методы, применяемые при работе с множествами:

1. Создание копии множества с помощью функции `copy()`:

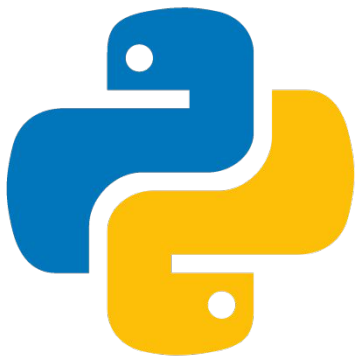
```
>>> t=set([1, 5, 9])
>>> c=t.copy()
>>> c
{1, 5, 9}
```

2. Добавление элемента во множество посредством функции `add()`:

```
>>> t=set([1, 5, 9])
>>> t.add(4)
>>> t
{1, 4, 5, 9}
```

3. Удаление элемента из множества через применение функции `remove()`, если указанный элемент отсутствует во множестве, то выдается сообщение об ошибке:

```
>>> t=set([1, 5, 9])
>>> t.remove(9)
>>> t
{1, 5}
```

Работа с множествами в «Python»

Методы, применяемые при работе с множествами:

4. Удаление элемента из множества через применение функции `discard()`, если указанный элемент отсутствует во множестве, то никаких изменений не выполняется:

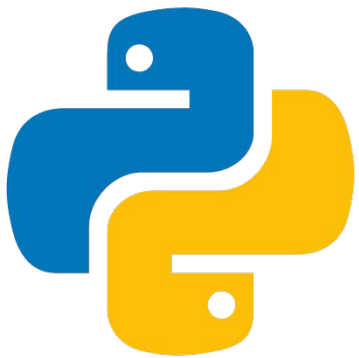
```
>>> t=set([1, 5, 9])
>>> t.discard(4)
>>> t
{1, 5, 9}
```

5. Удаление произвольного элемента из множества и его возвращение, что реализуется с помощью функции `pop()`, при этом, если начальное множество является пустым, ты выдается сообщение об ошибке:

```
>>> t=set([1, 5, 9])
>>> t.pop()
1
```

6. Удаление всех элементов множества, что достигается путем применения функции `clear()`:

```
>>> t=set([1, 5, 9])
>>> t.clear()
>>> t
set()
```



Работа с множествами в «Python»

Язык программирования «Python» поддерживает еще и неизменяемые множества, которые задаются с помощью функции `frozenset ()`:

```
>>> t=frozenset()  
>>> t  
frozenset()
```

Применение функции `frozenset()` позволяет выполнять преобразование таких элементов последовательности во множество, как:

1) строки:

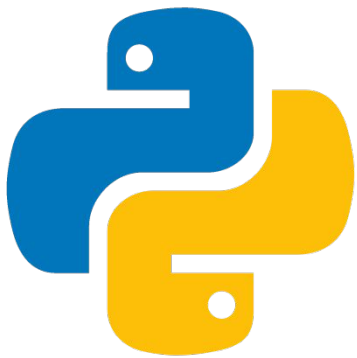
```
>>> frozenset ('май')  
frozenset({'a', 'й', 'м'})
```

2) списка:

```
>>> frozenset ([1, 3, 5])  
frozenset({1, 3, 5})
```

3) кортежа:

```
>>> frozenset ((1, 3, 5))  
frozenset({1, 3, 5})
```



Применение типа данных «словарь» в «Python»

Словарь представляет собой особый список, элементы которого имеют заданные пользователем индексы, называемые ключами. В роли индексов могут выступать данные любого типа, не допускающие изменений – числа, строки и кортежи, состоящие из чисел и строк, нельзя использовать списки. Словарь представляется как неупорядоченное множество пар ключ: значение, помещенное в фигурные скобки, с требованием уникальности ключей в пределах одного словаря.

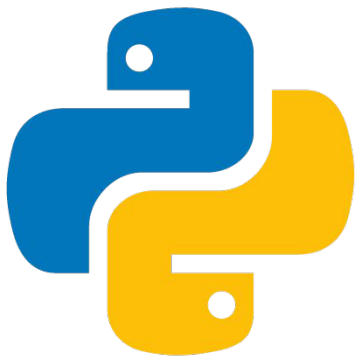
Создать словарь можно одним из следующих способов:

1) с помощью литерала:

```
>>> d={'color':3456, 'size':5}
>>> d
{'color': 3456, 'size': 5}
>>> d={} #создание пустого словаря
>>> d
{}
```

2) при помощи функции `dict()`:

```
>>> d = dict(color='3456', size='5')
>>> d
{'color': '3456', 'size': '5'}
>>> d = dict([('color', 3456), ('size', 5)])
>>> d
{'color': 3456, 'size': 5}
```



Применение типа данных «словарь» в «Python»

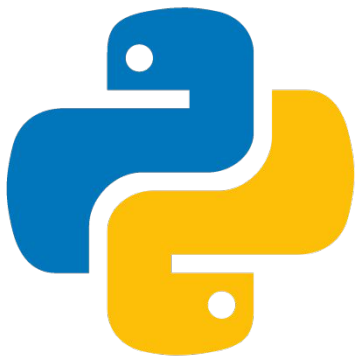
Создать словарь можно одним из следующих способов:

3) посредством применения метода *fromkeys()*:

```
>>> d = dict.fromkeys(['color', 'size'])
>>> d
{'color': None, 'size': None}
>>> d = dict.fromkeys(['color', 'size'], 20)
>>> d
{'color': 20, 'size': 20}
```

4) с помощью генераторов словарей:

```
>>> d={a:a**3 for a in range(6)}
>>> d
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

Применение типа данных «словарь» в «Python»

Методы, применяемые при работе со словарем:

1. Удаление пары *ключ: значение* посредством применения функции *del*:

```
>>> d={'color':3456, 'size':5}
>>> d
{'color': 3456, 'size': 5}
>>> del d ['color']
>>> d
{'size': 5}
```

2. Добавление пары *ключ: значение*:

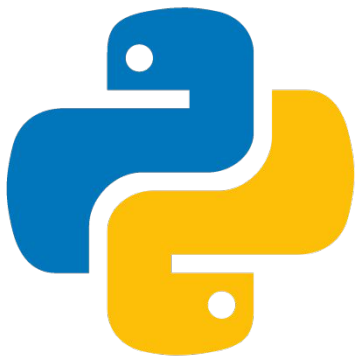
```
>>> d={'color':3456, 'size':5}
>>> d['long']=4
>>> print(d)
{'color': 3456, 'size': 5, 'long': 4}
```

3. Проверка наличия ключа в словаре выполняется при помощи оператора *in*:

```
>>> d={'color':3456, 'size':5}
>>> 'color' in d
True
>>> 'long' in d
False
```

4. Доступ к элементу словаря осуществляется следующим образом:

```
>>> d={'color':3456, 'size':5}
>>> d['color']
345
```

Применение типа данных «словарь» в «Python»

Методы, применяемые к типу данных «словарь» позволяют осуществлять следующие действия:

1. Очищение словаря, при помощи функции `dict.clear()`:

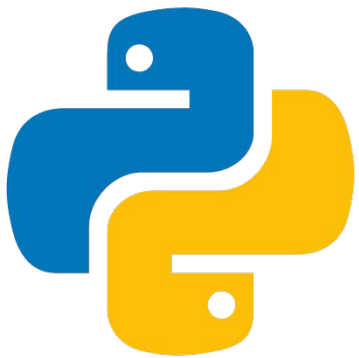
```
>>> d={'color':3456, 'size':5}
>>> d
{'color': 3456, 'size': 5}
>>> dict.clear(d)
>>> d
{}
```

2. Возвращение копии словаря, через применение функции `dict.copy()`:

```
>>> d={'color':3456, 'size':5}
>>> dict.copy(d)
{'color': 3456, 'size': 5}
```

3. Возвращение ключей в словаре при помощи функции `dict.keys()`:

```
>>> d={'color':3456, 'size':5}
>>> dict.keys(d)
dict_keys(['color', 'size'])
```



Применение типа данных «словарь» в «Python»

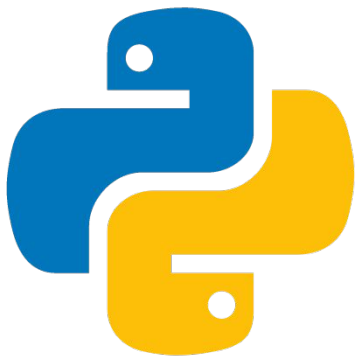
Методы, применяемые к типу данных «словарь» позволяют осуществлять следующие действия:

4. Возвращение пары ключ: значение через функцию dict.items():

```
>>> d={'color':3456, 'size':5}
>>> dict.items(d)
dict_items([('color', 3456), ('size', 5)])
```

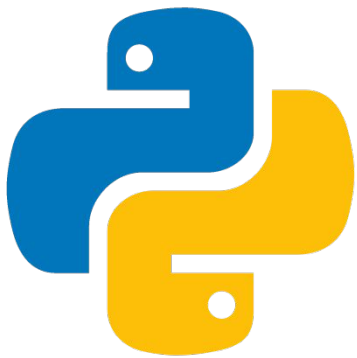
5. Возвращение значения в словаре при помощи функции dict.values():

```
>>> d={'color':3456, 'size':5}
>>> dict.values(d)
dict_values([3456, 5])
```



Операторы в «Python»

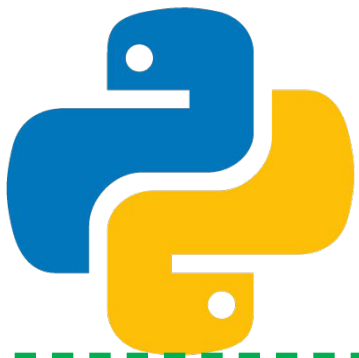
Логические операторы		
Оператор	Описание	Пример
<i>and</i>	Логический оператор «И», условие будет истинным, если оба операнда истинны	True and True равно True; True and False равно False; False and True равно False; False and False равно False
<i>or</i>	Логический оператор «ИЛИ», если хотя бы один из операндов истинный, то и все выражение будет истинным	True or True равно True; True or False равно True; False or True равно True; False or False равно False
<i>not</i>	Логический оператор «НЕ», изменяет логическое значение операнда на противоположное	not True равно False; not False равно True



Операторы в «Python»

Операторы членства	
Оператор	Описание
<i>in</i>	выводит истину, если элемент присутствует в последовательности, иначе выводит ложь
<i>not in</i>	<u>выводит истину если элемента нет в последовательности</u>

Операторы тождественности		
Оператор	Описание	Пример
<i>is</i>	выводит истину, если оба операнда указывают на один объект	$x \text{ is } y$ вернет истину, если $id(x) = id(y)$
<i>is not</i>	<u>выводит ложь если оба операнда указывают на один объект</u>	$x \text{ is not } y$ вернет истину если $id(x) \neq id(y)$



Преобразование типов данных в «Python»

1. Преобразование целых чисел в числа с плавающей точкой реализуемое с помощью функции `float()`:

```
>>> float(57)
57.0
```

Также можно использовать переменные, при этом сначала нужно объявить переменную, а затем вывести как вещественное число:

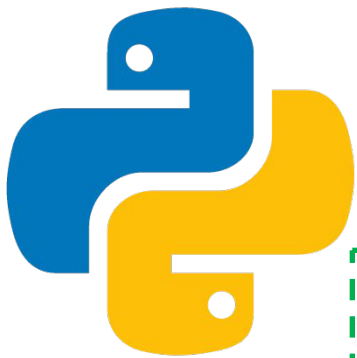
```
>>> z=57
>>> print(float(z))
57.0
```

2. Преобразование чисел с плавающей точкой в целые числа достигается через применение функции `int()`:

```
>>> int(25.8)
25
```

В приведенном примере показано, что функция `int()` не округляет числа, а просто отбрасывает знаки после запятой. Данная функция также может работать с переменными:

```
>>> a=25.8
>>> print(int(a))
25
```

Преобразование типов данных в «Python»

3. Преобразование чисел (как целых, так и вещественных) в строки, которое может быть достигнуто с помощью функции `str()`:

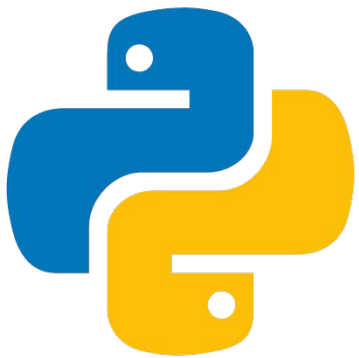
```
>>> str(5)
'5'
```

Кавычки означают, что 5 является строкой, а не числом. Функция `str()` также позволяет применять переменные для преобразования чисел в строковый тип данных:

```
>>> z=34.8
>>> print(str(z))
34.8
```

Здесь необходимо отметить, что «Python» не позволяет соединять строки с числами, однако, для этого сначала следует перевести числовой тип данных в строковый:

```
>>> a="в мае"
>>> b=25
>>> print( "" + a + " температура может достигать " + str(b) +
" градусов")
в мае температура может достигать 25 градусов
```

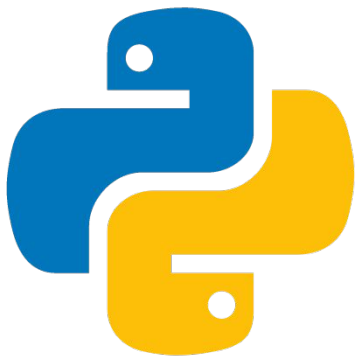


Преобразование типов данных в «Python»

4. Преобразование строк в числовой тип данных, выполняемое с применением функций `int()` и `float()`, при этом, если в строке отсутствуют десятичные знаки, то лучше выполнять преобразование в целое число:

```
>>> int('25')
25
>>> float('24.4')
24.4
```

Такое преобразование, аналогично рассмотренным ранее, позволяет работать с переменными.



Преобразование типов данных в «Python»

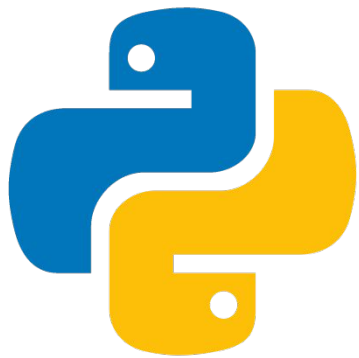
5. Преобразование списка в кортеж, осуществляемое с помощью функции `tuple()`:

```
>>> print(tuple(['май', 'июнь', 'июль']))  
('май', 'июнь', 'июль')
```

Также существует возможность преобразования в кортеж строковых данных:

```
>>> print(tuple('лето'))  
('л', 'е', 'т', 'о')
```

Преобразование числового типа в кортеж не допускается.

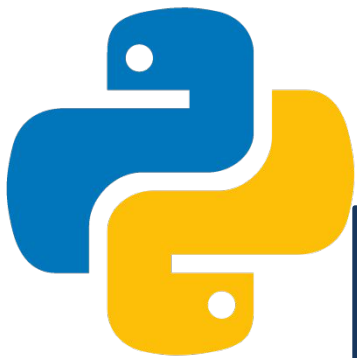


Преобразование типов данных в «Python»

6. Преобразование кортежа в список, выполняемое с применением функции `list()` такое преобразование выполняют чаще всего для того, чтобы сделать кортеж изменяемым:

```
>>> print(list('лемо'))  
['л', 'е', 'м', 'о']
```

Рассматриваемое преобразование позволяет работать с переменными, следовательно, необходимо уделять особое внимание скобкам, поскольку функции `print()` и `list()` используют круглые скобки.



Генерация случайных значений в языке «Python»

Генерировать случайные значения в языке программирования «Python» позволяет модуль *random*. Прежде чем применять данный модуль, следует выполнить его подключение с помощью инструкции: *import random*

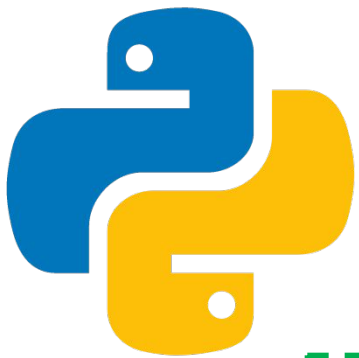
Основные возможности модуля *random*:

1) возвращение псевдослучайного числа от 0.0 до 1.0 посредством функции *random()*:

```
>>> import random  
>>> random.random()  
0.8961146431329136
```

2) настройка генератора случайных чисел на новую последовательность при помощи функции *seed()*:

```
>>> random.seed()  
>>> random.random()  
0.4562937717818225
```

Генерация случайных значений в языке «Python»

Основные возможности модуля random:

3) возвращение псевдослучайного вещественного числа из указанного диапазона через применение функции `uniform(a, b)`, в которой a – начало диапазона, b – конец:

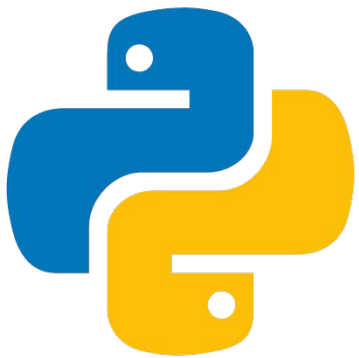
```
>>> random.uniform(5, 15)  
14.162487558142743
```

4) возвращение псевдослучайного целого числа из указанного диапазона с помощью функции `randint(a, b)`, в которой a – начало диапазона, b – конец:

```
>>> random.randint(5, 15)  
11
```

5) возвращение случайного элемента из числовой последовательности, что позволяет осуществить функция `randrange(a, b, c)`, в которой a – начало диапазона, b – конец диапазона, c – шаг:

```
>>> random.randrange(10, 24, 3)  
13
```



Генерация случайных значений в языке «Python»

Основные возможности модуля random:

6) возвращение случайного элемента из заданной последовательности (строка, список, кортеж), что может быть реализовано при помощи функции `choice()`:

```
>>> random.choice('май')  
'й'
```

7) перемешивание элементов списка случайным образом, через применение функции `shuffle()`:

```
>>> s=['л', 'е', 'м', 'о']  
>>> random.shuffle(s)  
>>> s  
['е', 'м', 'о', 'л']
```

8) возвращение списка из указанного количества элементов, которые будут выбраны случайным образом из заданной последовательности, что может быть достигнуто применением функции `sample(a, b)`, где `a` – последовательность элементов, `b` – количество элементов:

```
>>> s=['л', 'е', 'м', 'о']  
>>> random.sample(s, 3)  
['о', 'м', 'е']
```