**Типы данных языка python. Операции над ними**

В этой главе рассматриваются основные типы данных языка Python и операции над ними.

**Целые числа. Тип int**

В языке Python для представления целых чисел используется тип int. Литералы целых чисел по умолчанию записываются в десятичной системе счисления. При работе с целыми числами используются математические операторы. Пусть даны два целых числа X и Y, тогда:

* X + Y – складывает числа X и Y;
* X - Y – вычитает число Y из X;
* X \* Y – умножает числа X и Y;
* X / Y – делит число X на Y – результатом является значение типа float (число с плавающей точкой);
* X // Y – делит число X на Y, при этом результат округлен до ближайшего целого числа в менышую сторону;
* X % Y – возвращает остаток от деления X на Y;
* X \*\* Y – возводит X в степень Y;
* - X – изменяет знак числа X, если X не равен 0;

Для работы с целыми числами в языке Python предусмотрены и встроенные функции, которые являются аналогами математических функций:

* abs(X) – возвращает модуль числа X;
* pow(X,Y,[Z]) – альтернативен записи (X\*\*Y)%Z;

Для операторов (+,-,/,//,%,\*\*) существует альтернативная укороченная запись, имеющая общий вид: x op= y, которая раскрывается в x = x op y. Например x = x \* y, можно записать как: x\*=y.

a = 4  
a = a \* 2  
print(a)

Вывод на консоль:

8

a = 4  
a \*= 2  
print(a)

Вывод на консоль:

8

**Логический тип. Bool**

В Python логический тип представлен типом данных bool. Переменная типа bool может принимать два возможных значения: False (ложь) и True (истина). В языке Python имеются три логических оператора:

* X and Y – результатом является True, тогда и только тогда, когда X и Y – являются True;
* X or Y – результатом является False, тогда и только тогда, когда X и Y – являются False;
* not X – возвращает противоположное X значение;

В Python разрешено рассматривать любое значение как истинное или ложное. Например, в логических выражения число 0 представляет False, а все остальное как True.

print(bool(2))  
print(bool(1))  
print(bool(0))

Вывод на консоль:

True

True

False

Логические значения могут трактоваться как числа, т.е. True – 1, а False – 0. Значение типа int можно привести к типу bool, с помощью функции bool().

print(int(True))  
print(int(False))

Вывод на консоль:

1

0

**Числа с плавающей точкой. Тип float**

В языке Python для представления десятичных дробей используется тип float. Такие дроби в программировании называются как «Числа с плавающей точкой». Все числовые операторы и функции типа int применимы для типа float. Для типа float существует дополнительная встроенная функция round(), которая округляет число с плавающей точкой до заданной точности. Число типа int можно привести к типу float с помощью функции float(), а число типа float привести к типу int с помощью функции int(), при этом дробная часть будет отброшена. При выполнении операции с числами типа float, возможна ситуация, когда полученное значение - NaN (не число) или бесконечность.

**Строки. Тип str**

Строки в Python представляют из себя набор символов, заключённый в одинарные (' ') или двойные (" ") кавычки, при этом строки “Hello World” и ‘Hello World’ – это одно и то же. Применив кавычки одного типа в роли ограничителей, автоматически запрещается их использование внутри строки. Строки в языке Python представлены типом данных str. Любой ранее рассмотренный тип данных можно привести к типу str с помощью функции str(). Строки можно привести к типу int или float с помощью соответствующих функции int() и float(). Однако, на такое приведение накладываются дополнительные ограничения:

* Строку можно привести к типу int, тогда и только тогда, когда строка содержит числовые литералы.
* Строку можно привести к типу float, тогда и только тогда, когда строка содержит числовые литералы и один разделитель десятичной части.

В строках для представления специальных (например, непечатаемых) символов используются escape-последовательности, состоящие из обратной косой черты, за которой следует один или несколько символов. Например, \n – символ перевода строки. Доступные escape-последовательности:

|  |  |
| --- | --- |
| **\b** | Backspace |
| **\f** | Перевод страницы |
| **\n** | Новая строка |
| **\r** | Возврат каретки |
| **\t** | Горизонтальная табуляция |
| **\v** | Вертикальная табуляция |
| **\'** | Одиночная кавычка |
| **\"** | Двойная кавычка |
| **\\** | Обратная косая черта |
| **\?** | Литерал вопросительного знака |

Строки поддерживают обычные операторы сравнения <,<=,==,!=,>,>=. Эти операторы выполняют побайтовое сравнение строк. Для строк определены операторы +, \* и in. Оператор + возвращает склеенную строку.

Например, “Hello” +” world” вернет “Hello world”:

print("Hello" + " world")

Вывод на консоль:

Hello world

Умножение целого числа на строку вернет, строку содержащую размноженную копию исходной строки. Например, “Hello” \* 3, вернет “HelloHelloHello”:

print("Hello" \* 3)

Вывод на консоль:

HelloHelloHello

Оператор in cчитается истиной (true), если находит переменную в заданной последовательности, и ложью (false) в противном случае. Например: ‘e’ in ‘Hello’ – вернет True, а ‘w’ in ‘Hello’ – вернет False.

s = 'Hello'  
print('e' in s)  
print('w' in s)

Вывод на консоль:

True

False

Для обращения к отдельным элементам строки используется оператор []. Например “Hello”[1] вернет “e”, но с помощью оператора [] нельзя изменять значение элемента, потому что строки в Python являются неизменяемыми.

s = 'Hello'  
print(s[1])

Вывод на консоль:

e

С помощью оператора [] можно получить не только один элемент, а сразу несколько подряд идущих элементов. Для этого требуется в операторе [] указать диапазон, при этом последний элемент диапазона не будет включен в последовательность. Например, “Hello”[0:2] вернет “He”. Если в срезе опущен первый символ, значит, он равен нулю; если опущен последний символ – он равен длине строки. Например, “Hello” [0:2] == “Hello” [:2] == “He”, “Hello” [3:] == “lo”.

s = 'Hello'  
print(s[0:2])  
print(s[0:2])  
print(s[3:])

Вывод на консоль:

He

He

lo

Для типа str в языке Python определено большое число встроенных функции, рассмотрим наиболее популярные из них. Пусть s = “Мама мыла раму”, тогда:

* find() – находит подстроку в строке – возвращает позицию вхождения строки, либо -1. Пример: s.find(‘мыла’) вернет 5.
* split() – разбивает строку на последовательности по указанному разделителю. Разделитель по умолчанию – пробел.  
  Пример: s.split() вернет список последовательностей [“Мама”,”мыла”,”раму”]
* join() – обратная к функции split(). Объединяет через указанный разделитель. Разделитель по умолчанию – пробел.  
  Пример: s.split().join(“!”) вернет строку “Мама!мыла!раму”.
* ……..

Так как строки являются последовательностями, то с помощью встроенной функции len() можно получить количество символов в строке. Например, len(“Hello”) вернет 5.

**Приоритет операторов.**

|  |  |
| --- | --- |
| Оператор | Описание |
| **\*\*** | Возведение в степень |
| **\* / % //** | Умножение, деление, деление по модулю, целочисленное деление. |
| **+ -** | Сложение и вычитание. |
| **>> <<** | Побитовый сдвиг вправо и побитовый сдвиг влево. |
| **<= < > >=** | Операторы сравнения |
| **<> == !=** | Операторы равенства |
| **= %= /= //= -= += \*= \*\*=** | Операторы присваивания |
| **is is not** | Тождественные операторы |
| **in not in** | Операторы членства |
| **not or and** | Логические операторы |

Структуры данных

Очень часто в программировании необходимо хранить набор значений одного или различных типов, с возможностью обращаться к этим значениям. Для этого используются коллекции.

Коллекции поддерживают функцию len() возвращающую количество элементов:

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

3

и оператор in (True – если элемент содержится в коллекции, иначе – False):

>> 1 in [1, 2, 3]

True

>> 5 in [1, 2, 3]

False

Пустая коллекция интерпретируется как ложное условие (False), в то время как коллекция, состоящая хотя бы из одного элемента, интерпретируется как истинное (True):

>> bool([])

False

>> bool([1, 2, 3])

True

# Последовательности

Последовательности – это коллекции, к которой можно обращаться по индексу, т. е. у каждого элемента коллекции есть свой порядковый номер - индекс. В Python коллекции могут содержать элементы различных типов данных (int, bool, string и т. д.), в том числе другие коллекции.

Обращение по индексу производится с помощью оператора фигурные скобки ([]). В языке Python первый элемент в последовательности будет иметь индекс 0.

>> l = [1, 2, 3]

>> l[0]

1

>> l[1]

2

Ранее рассмотренные нами строки тоже являются последовательностями, элементами которых являются символы. Операторы, которые мы определили для строк (операторы сравнения (<, <=, ==, !=, >, >=) и +, \*) справедливы для всех последовательностей.

Мы рассмотрим два фундаментальных типа последовательностей – кортежи и списки.

## КОРТЕЖИ

Кортеж – это неизменяемая последовательность элементов. Когда мы рассматривали строки, мы уже использовали понятие «неизменяемый», это означает, что после того, как мы создали объект, его уже невозможно изменить (т. е. вставить, изменить или удалить какой-либо элемент).

Для создания кортежа можно использовать конструктор tuple() или перечислить элементы через запятую в круглых скобках:

# Создание пустого кортежа

t1 = tuple()  
t2 = ()

# Создание кортежа c элементами

t1 = tuple('мама', 1, True)  
t2 = ('мама', 1, True)

Если в кортеже 1 элемент, то при создании кортежа после него необходимо указать запятую:

# Создание кортежа из одного элемента

t = ('м',)

## СПИски

Список – это изменяемая последовательность элементов.

Для создания листа можно использовать конструктор list() или перечислить элементы через запятую в квадратных скобках:

# Создание пустого списка

l1 = list()  
l2 = []

# Создание списка c элементами

l1 = list('мама', 1, True)  
l2 = ['мама', 1, True]

Так как списки изменяемы, то для них определены следующие методы, непосредственно их изменяющие:

* l.append(x) – добавляет элемент x в конец списка;
* l.insert(i, x) – вставляет элемент x на позицию i;
* l.remove(x) – удаляет первый элемент из списка, имеющий значение x;
* l.pop(i) – удаляет i-ый элемент списка и возвращает его;
* l.index(i) – возвращает i-ый элемент списка;
* l.sort() – сортирует список.

Преобразовать одну последовательность к другой очень просто. Например, если мы хотим получить отсортированный кортеж, можно просто преобразовать его в список, с помощью конструктора list(), затем отсортировать с помощью метода sort(), и преобразовать обратно в кортеж, с помощью tuple():

>> t = (2, 1, 3)

>> t

(2, 1, 3)

>> l = list(t)

>> l

[2, 1, 3]

>> l.sort()

>> l

[1, 2, 3]

>> t = tuple(l)

>> t

(1, 2, 3)

# Словари и множества

## Словари

Словарь – неупорядоченная коллекция пар «ключ-значение». Понятие «неупорядоченность» означает, что к элементу в коллекции нельзя обратиться по индексу, т. к. элементы не расположены по порядку, один за другим. Доступ к значениям в словаре производится по «ключу». Это сравнимо с поиском по толковому словарю, где «ключ» — это слово, а «значение» — определение этого слова.

В качестве ключа можно использовать любой изученный нами тип данных, в качестве значения – что угодно, даже другой словарь.

Для создания словаря можно использовать конструктор dict() или перечислить пары «ключ-значение» через запятую в фигурных скобках:

# Создание пустого словаря

d = dict()  
d = {}

# Различные способы создания одного и того же словаря

d = dict({'name': 'Вася', 'age': 12})  
d = dict(name='Вася', age=12)  
d = dict([('name', 'Вася'), ('age', 12)])  
d = {'name': 'Вася', 'age': 12}

Для доступа к элементам используются фигурные скобки, в которых указывается ключ:

>> d = {'м': 1}

>> d['м']

1

Однако, если обратиться к несуществующему ключу (например, d['а']), вызовется исключение KeyError и работы программы завершится. Для безопасного доступа к словарю используется функция get():

>> d.get('а', 2)

2

Ключи в словаре являются уникальными, т. е. по одному и тому же ключу не могут храниться разные значения, поэтому если в словарь добавляется пара «ключ-значение» с ключом, который уже существует, то в результате происходит замена старого значения новым:

>> d = {'м':1}

>> d['м']

1

>> d['м'] = 2

>> d['м']

2

Некоторые методы словарей:

* d.get(key[, default]) – возвращает значение ключа или default если ключа нет в словаре;
* d.items() – возвращает пары «ключ-значение» словаря;
* d.keys() – возвращает ключи словаря;
* d.values() – возвращает значения словаря;
* d.pop(key[, default]) – удаляет ключ и возвращает значение или default.

## множества

Множество – это неупорядоченная коллекция элементов. Множество хранит в себе элементы также, как и математическое множество. Все элементы множества должны быть уникальны.

Для создания множества можно использовать конструктор set() или перечислить элементы через запятую в круглых фигурных скобках:

# Создание пустого множества

s1 = set()

# Создание множества c элементами

s1 = set('мама') # {'а', 'м'}  
s2 = {'м', 'а', 'м', 'а'} # {'а', 'м'}

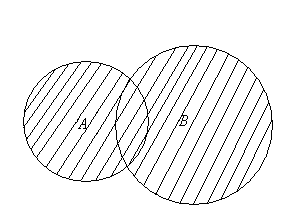
Как показано в последнем примере, мы можем преобразовать в множество последовательность. Таким образом, множество может использоваться для контроля уникальности элементов в последовательности. Множество можно привести обратно к последовательности с помощью конструкторов.

Множества поддерживают стандартные математические операции на множествами:

* Объединение:

>> set('мама') | set('папа')

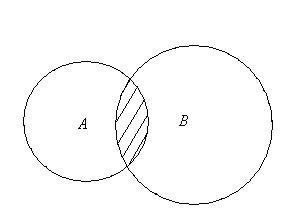
{'а', 'п', 'м'}

;

* Пересечение:

>> set('мама') & set('папа')

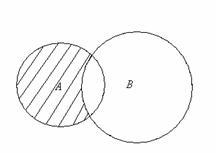
{'а'}

;

* Разность:

>> set('мама') - set('папа')

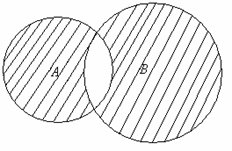
{'м'}

;

* Исключающее ИЛИ

>> set('мама') ^ set('папа')

{'п', 'м'}

.

Некоторые методы множеств:

* s.add(x) - добавляет элемент в множество;
* s.remove(x) - удаляет элемент из множества.