Типы данных

АСДП

Понятие типа данных и переменной

Тип данных (англ. Data type) - характеристика, определяющая:

- множество допустимых значений, которые могут принимать данные, принадлежащие к этому типу (например, объект типа Целое число может принимать только целочисленные значения в определенном диапазоне);
- набор операций, которые можно осуществлять над данными, принадлежащими к этому типу (например, объекты типа Целое число умеют складываться, умножаться и т.д.).

- Все типы в Python являются объектами (в отличие, например, от C++). Объект в Python - это абстракция над данными: любые данные здесь представлены объектами.
- При создании объекта вызывается специальная функция – конструктор (специальный блок инструкций, вызываемый при создании объекта).
- Переменная (англ. Variable) - это идентификатор, который указывает на определенную область памяти, где хранятся произвольные данные - созданный объект (значение переменной).

Примечание: Для имен переменных используется змеиный_регистр (англ. snake_case): например, my_variable. Стиль написания составных слов, при котором несколько слов разделяются символом подчеркивания (_), и не имеют пробелов в записи, причём каждое слово обычно пишется с маленькой буквы.

Примечание: Переменным необходимо давать информативные имена, по которым можно было бы понять, с какими данными она связана - это чрезвычайно облегчает дальнейшее чтение и изменение кода.

Например, переменную, хранящую данные о скорости можно назвать speed, а не sk; значение баланса телефона клиента - balance, а не b и т.д. «Привычные» со школы короткие имена следует использовать там, где они либо подходят по смыслу (например, a, b, c в роли коэффициентов квадратного уравнения), либо используются временно (например, счетчик i в циклической конструкции).

Классификация типов данных

В Python встроенные типы данных подразделяются на 2 группы:

1. Скалярные (неделимые).
    - Числа (целое, вещественное).
    - Логический тип.
    - NoneType.
2. Структурированные (составные) / коллекции.
    - Последовательности: строка, список, кортеж, числовой диапазон.
    - Множества.
    - Отображения: словарь.

Кроме того, все объекты в Python относятся к одной из 2-х категорий:

1. Мутирующие (англ. Mutable): содержимое объекта можно изменить после создания (например, список);

2. Немутирующие (англ. Immutable): содержимое объекта нельзя изменить после создания (например, строка или число).

Примечание: Также часто используется терминология «изменяемые» и «неизменяемые» типы соответственно.

Основным преимуществом немутирующих типов является гарантия неизменяемости с момента создания: каждый использующий участок кода имеет дело с копией объекта и не может его каким-либо образом изменить.

Этот же принцип формирует основной недостаток немутирующих типов: большее количество потребляемой памяти на дополнительное копирование объектов при необходимости внесения изменений.

Оператор присваивания

Для связывания (и при необходимости предварительного создания) объекта и переменной используется оператор присваивания =.

Присваивание выполняется «справа налево» и подразумевает шаги:

- если справа от оператора находится литерал (например, строка или число) в операнд слева записывается ссылка, которая указывает на объект в памяти, хранящий значение литерала:

```
a = 100  # Создание объекта 100 и запись ссылки на него в переменную 'a'
```

- если справа находится ссылка на объект, в левый операнд записывается ссылка, указывающая на тот же самый объект, на который ссылается правый операнд;

```
a = 100
b = a  # В переменную 'b' копируется ссылка из 'a' -
       # они будут указывать на один и тот же объект
```

Переменная лишь указывает на данные - хранит ссылку, а не сами данные. В виду того, что копирования данных при этом не происходит, операция присваивания выполняется с высокой скоростью.

```
# Выражение присваивания в Python может быть записано несколькими способами в
# зависимости от содержимого левой части (l-значение) и правой части (r-значение).

# 1. Простое присваивание использует одно l-значение и одно r-значение.
#
# Объект 5 целого типа связывается с переменной 'a'
>>> a = 5
>>> a
5

# 2. Сокращенная запись присваивания часто применяется когда нужно обновить
#    значение переменной по отношению к текущему значению.
#    Сокращенная запись образуется для всех операторов одинаково, например
#    '*=' для умножения и т.д.
#
# Увеличиваем значение связанного целого объекта 'a' на 1, эквивалентно a = a + 1
>>> a += 1
>>> a
6
```

```
# 3. Также возможно параллельное присваивание, где выражение присваивания
#    содержит больше одного l- и r-значений.
#
# Связывание значений 1, "aaa", 3 с переменными 'x', 'y', 'z' соответственно
>>> x, y, z = 1, "aaa", 3
>>> x, y, z
(1, 'aaa', 3)

# Возможно также не совпадения количества значения l- и r-значений,
# данный случай рекомендуется рассмотреть самостоятельно.

# 4. Оператор присваивание выполняется справа налево, поэтому также можно
#    образовывать цепочки присваивания.
#
# 0 связывается с переменной 'y', а затем и с переменной 'x'
>>> x = y = 0
>>> x, y
(0, 0)
```

Инициализация переменной перед использованием

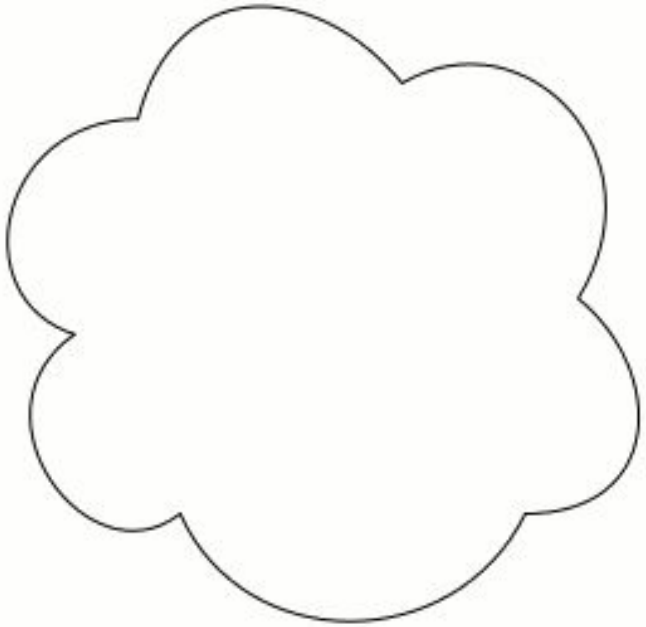
Переменная должна быть проинициализирована (ссылаться на данные) перед использованием в выражении. Например, код `a = b + 2` или `b += 1`, вызовет ошибку, если идентификатор `b` не был предварительно определен.

Управление памятью и сборщик мусора

- Создание объекта любого типа подразумевает выделение памяти для размещения данных об этом объекте. Когда объект больше не нужен - его необходимо удалить, очистив занимаемую память. Python - язык с встроенным менеджером управления памятью и выполняет данные операции автоматически за счет наличия сборщика мусора (англ. Garbage Collection, GC).
- Сборка мусора - технология, позволяющая, с одной стороны, упростить программирование, избавив программиста от необходимости вручную удалять объекты, созданные в динамической памяти, с другой - устранить ошибки, вызванные неправильным ручным управлением памятью. Алгоритм, используемый сборщиком мусора называется подсчетом ссылок (англ. Reference Counting). Python хранит журнал ссылок на каждый объект и автоматически уничтожает объект, как только на него больше нет ссылок.

```
pi = 3.14
radius = 10
area = pi * radius**2

radius = 15
```



Примерная схема работы сборщик мусора

- Время между созданием и уничтожением объекта - его жизненный цикл.
- Объекты, которые имеют на протяжении своего жизненного цикла одно неменяющееся и характеризующее их значение, а также умеют сравниваться, называются хешируемыми. К хешируемым объектам относятся все немутирующие типы данных, а также пользовательские объекты.

Скалярные типы

- В Python существует 2 категории чисел: целые и вещественны

Целые числа в Python представлены типом int.

Литералы целых чисел по умолчанию записываются в десятичн
но при желании можно использовать и другие

```
>>> 15      # десятичное число
15

>>> 0b1111  # двоичное число
15

>>> 0o17    # восьмеричное число
15

>>> 0xF     # шестнадцатиричное число
15
```

Python предоставляет три типа значений с плавающей точкой:

- float (двойная точность). Числа типа float записываются с десятичной точкой или в экспоненциальной форме записи.
- complex (комплексные числа вида 3.5 + 5j);
- decimal.Decimal (большая точность, по умолчанию 28 знаков после запятой).

```
>>> -2.5          # Отрицательное вещественное число
-2.5

>>> 8e-4          # Экспоненциальная форма записи
0.0008
```

Для чисел с плавающей точкой существует ряд нюансов:

- в машинном представлении такие хранятся как двоичные числа. Это означает, что одни дробные значения могут быть представлены точно (такие как 0.5), а другие - только приблизительно (такие как 0.1 и 0.2, например, их сумма будет равна не 0.3, а 0.30000000000000004);
- для представления используется фиксированное число битов, поэтому существует ограничение на количество цифр в представлении таких чисел.

В связи с этим числа типа float не могут надежно сравниваться на равенство значений, т.к. имеют ограниченную точность. Проблема потери точности - это не проблема, свойственная только языку Python, а особенность компьютерного представления чисел.

```
>>> 0.1 + 0.2        # Проблема потери точности актуальна для вещественных чисел
0.30000000000000004
```

Операции над числами

Для арифметических операций тип результата операции определяется типом аргументов. Если тип результата явно не предусмотрен при вычислении (например, округление до целых подразумевает получение результата типа int), действуют следующие правила:

- float: если хотя бы один аргумент имеет тип float;
- int: если все аргументы имеют тип int.

Скалярные типы

- Логический тип представлен типом bool и позволяет хранить 2 значения:

  True (Истина / Да / 1);

  False (Ложь / Нет / 0).

- NoneType

В Python существует специальное значение None типа NoneType, обозначающее нейтральное или «нулевое» поведение. Присвоение такого значения ничем не отличается от других: a = None, обозначая, что идентификатор a задан, но ни с чем не связан.

Наиболее часто используется для защитного программирования - «если что-то не None, можно продолжать работу программы».

Коллекции

- Используя скалярные типы, можно столкнуться с проблемой - что делать, если необходимо хранить и обрабатывать набор таких значений. Для этого в Python предназначены специальные типы - коллекции.

Коллекции — это группа типов данных, которые содержат в себе другие данные и поддерживают:

- проверку на вхождения элементов in и not in (True/False);
- определение размера len();
- возможность выполнения итераций (перемещения по элементам последовательности) - из-за этого коллекции также называются итерируемыми объектами

Среди коллекций выделяют 3 группы:

- последовательности: строка, список, кортеж, числовой диапазон;
- множества;
- отображения: словарь.

Последовательности

- Последовательность - это упорядоченная коллекция, поддерживающая индексированный доступ к элементам.
- Некоторые последовательности в Python в отличие от традиционных массивов (например, в Паскале или Си) могут хранить элементы различного типа (в том числе и коллекции различных видов).

Последовательности. Общие операции

Некоторые операции справедливы для всех последовательностей (за исключением случаев, где они не возможны исходя из определения типа).

Обозначения:

- s и t: последовательности одного типа;
- n, k, start, end, step: целые числа;
- x: произвольный объект, отвечающий критериям соответствующего вызова функции.

Последовательности. Общие операции

- Длина

len(s)

Функция len() возвращает длину (количество элементов в последовательности) s.

- Конкатенация («склеивание»)

s + t

Возвращает новый объект - «склейку» s и t.

- Дублирование

s * n

n * s

Возвращает последовательность, повторяющуюся n раз.

Последовательности. Общие операции

Индексация и срезы

Получить доступ к отдельному элементу или группе элементов последовательности возможно с помощью оператора []. Индексацию (получение отдельного элемента) можно считать частным случаем получения среза (слайсинга).

Оператор получения среза имеет три формы записи:

s[start]

s[start:end]

s[start:end:step]

- s[start]: индексация (с 0);
- s[start:end]: срез [start; end);
- s[start:end:step]: срез [start; end) с шагом step.

В ряде случаев целочисленные параметры start, end и step могут быть опущены. Элемент с индексом end не включается в результат при взятии срезов.

Последовательности. Общие операции

- Минимальное и максимальное значения

min(s)

max(s)

Возвращает минимальный и максимальный элементы последовательности s соответственно.

- Проверка на вхождение

x in s

Возвращает True, если x входит в последовательность s и False в противном случае.

- Индекс (положение) элемента

s.index(x[, start[, end]]) --> int

Возвращает первое вхождение элемента x в последовательность s (между индексами start и end, если они заданы).

Последовательности. Общие операции

- Количество повторений

s.count(x)

Возвращает количество вхождений элементов x в последовательность s.

- Сортировка

sorted(s, key=None, reverse=False)

Возвращает отсортированный объект в виде списка. Исходный объект при этом не изменяется.

Параметры

- key – функция сортировки (по умолчанию не учитывается, сортировка осуществляется поэлементно);
- reverse – если равен True, сортировка осуществляется в обратном порядке.

Последовательности. Строки

- Строка (str) - это упорядоченная неизменяемая последовательность символов Юникода.
- Литералы строк создаются с использованием кавычек или апострофов, при этом важно, чтобы с обоих концов литерала использовались кавычки одного и того же типа. Также можно использовать строки в тройных кавычках, то есть строки, которые начинаются и заканчиваются тремя символами кавычки (либо тремя кавычками, либо тремя апострофами).
- Важным для строкового типа является понятие кодировки символов, что в частности, влияет на правила сравнения строк. По умолчанию Python хранит строки в кодировке UTF-8.
- Если в строке необходимо использовать специальные символы (например, перенос или одноименные кавычки), можно воспользоваться механизмом экранирования символов, для чего используется специальный символ \.

Последовательность	Значение
`\\`	Обратный слеш (`\`)
`\'`	Апостроф (`'`)
`\"`	Кавычка (`"`)
`\n`	Символ «Перевод строки»
`\t`	Символ «Табуляция»

Последовательности. Строки. Операции над строковым типом

```
# 1. Арифметические операции
#    Тип результата операции определяется типом аргументов

# s1 + s2
# Соединяет строки s1 и s2
>>> "Py" + "thon"
'Python'
# или просто написать рядом
>>> "Py" "thon"
'Python'

# s1 * n
# Составляет строку из n повторений строки s1
>>> "па" * 2
'папа'
```

Последовательности. Строки. Операции над строковым типом

```
# 2. Равенство и сравнение
#    Результат логического типа
#
#    Операции сравнения выполняются посимвольно слева направо с учетом кодировки.
#    Ознакомиться с таблицей символов Юникода рекомендуется на ресурсе
#    http://unicode-table.com/.

# s1 == s2, s1 != s2
# Проверка строк на равенство/неравенство
>>> "текст1" == "текст2"
False
>>> "текст1" != "текст2"
True

# x > y, x < y, x >= y, x <= y
# Больше, меньше, больше или равно, меньше или равно
>>> "текст1" > "текст2"
False
>>> "текст1" <= "текст2"
True

# Возможно составление цепочек сравнений
>>> "текст1" < "текст12" <= "текст2"
True

# s1 in s2
# Проверка вхождения строки s1 в s2
>>> "p" in "Python"
False
```

Последовательности. Строки. Индексация и срезы

```
# Для индексации и получения срезов удобно пользоваться обозначениями ниже
#
# Индексация
#  +---+---+---+---+---+---+
#  | P | y | t | h | o | n |
#  +---+---+---+---+---+---+
#    0   1   2   3   4   5
#   -6  -5  -4  -3  -2  -1
#
# Срезы
#  +---+---+---+---+---+---+
#  | P | y | t | h | o | n |
#  +---+---+---+---+---+---+
#  0   1   2   3   4   5   6
# -6  -5  -4  -3  -2  -1

>>> s = "Python"

>>> s[0]
'P'
>>> s[3]
'h'

>>> s[-1]  # Последний символ
'n'

>>> s[0:2]  # Срез включает первые 2 символа
'Py'
>>> s[2:-1]  # С 3 по предпоследний символ
'tho'
>>> s[0:-1:2]  # С 1 по предпоследний символ через 2
'Pto'

# Параметры [start] и [end] могут быть опущены
# В таком случае срез берется с начала и до конца строки соответственно
>>> s[:3]  # Первые 3 символа
'Pyt'
>>> s[3:]  # С 3-го символа до конца
'hon'
>>>
```

Последовательности. Строки. Характерные операции

Строки поддерживают все общие операции для последовательностей и имеют ряд дополнительных методов.

- chr(i)

Возвращает символ № i из таблицы Unicode.

- ord(c)

Возвращает номер символа c из таблицы Unicode.

Последовательности. Строки. Характерные операции

Пусть s - строка, на которой вызывается метод.

- upper()

Возвращает копию строки s в верхнем регистре.

- lower()

Возвращает копию строки s в нижнем регистре.

- capitalize()

Возвращает копию строки с первым символом в верхнем регистре.

- title()

Возвращает копию строки, в которой первые символы каждого слова преобразованы в верхний регистр, а все остальные - в нижний регистр.

Последовательности. Строки. Характерные операции

- count(t[, start[, end]])

Возвращает число вхождений строки t в строку s (или в срез s[start:end]).

- find(t[, start[, end]])

Возвращает позицию самого первого (крайнего слева) вхождения подстроки t в строку s (или в срез s[start:end]); если подстрока t не найдена, возвращается -1.

- index(t[, start[, end]])

Аналогично str.find(), но генерируется исключение ValueError, если подстрока не найдена.

- replace(old, new[, count])

Возвращает копию строки s, в которой каждое (но не более count, если этот аргумент определен) вхождение подстроки old замещается подстрокой new.

- split(sep=None, maxsplit=- 1)

Возвращает список строк, разбитых по строке sep.

- join(seq)

Возвращает строку-«склейку» элементов seq, используя s в качестве разделителя.

Использование строковых методов

```
>>> s = "ЭТО просТо ТеКст"

>>> ord(s[0])
1069
>>> chr(1069)
'Э'

>>> s.upper(), s.lower(), s.title(), s.capitalize()
('ЭТО ПРОСТО ТЕКСТ', 'это просто текст', 'Это Просто Текст', 'Это просто текст')

>>> s.find("Т")
1

>>> s.replace("Т", "т")
'ЭтО просто теКст'

>>> lst = s.split()
>>> lst
['ЭТО', 'просТо', 'ТеКст']

>>> "-".join(lst)
'ЭТО-просТо-ТеКст'
```

Последовательности. Список

- Список (list) - это упорядоченная изменяемая последовательность элементов.

Особенности:

- может содержать элементы разного типа;
- поддерживает операторы сравнения: при этом сравнивание производится поэлементно (и рекурсивно, при наличии вложенных элементов).

```
# 1. Пустой список создается с помощью пустых квадратных скобок или функции list()
>>> []
[]
>>> list()
[]

# 2.  Инициализировать список элементами можно одним из следующих способов:
>>> [1, "text", 2.0]
[1, 'text', 2.0]
>>> list("text")
['t', 'e', 'x', 't']
```

Последовательности. Список. Характерные операции

Списки поддерживают все общие операции для последовательностей, и имеют ряд дополнительных методов.

Пусть lst - список, на котором вызывается метод.

- append(x)

Добавляет элемент x в конец списка lst.

- extend(m)
- lst += m

Добавляет в конец списка lst все элементы коллекции m.

- insert(i, x)

Вставляет элемент x в список lst в позицию i.

Последовательности. Список. Характерные операции

- remove(x)

Удаляет из списка lst первый элемент со значением x.

- pop([i])

Возвращает последний или i-й элемент, удаляя его из последовательности.

- clear()

Удаляет из списка lst все элементы (очищает список).

- sort(lst, key=None, reverse=None)

Выполняет сортировку списка lst. Отличается от функции sorted() тем, что сортирует исходный объект, а не возвращает новый.

Параметры

- key – функция сортировки (по умолчанию не учитывается, сортировка осуществляется поэлементно);
- reverse – если равен True, сортировка осуществляется в обратном порядке.

- reverse()
- Переворачивает элементы списка lst.
- del lst[i[:j]]

Удаляет из списка lst элемент с индексом i (или несколько элементов, если задан индекс j).

```
# 1. Методы списка

>>> a = [8, 7, 5.5, 1000, 3.50, 200]

>>> a[0] = 7
>>> a
[7, 7, 5.5, 1000, 3.50, 200]

>>> a.index(7)
0

>>> a.count(7)
2

>>> a.insert(2, 1000)
>>> a
[7, 7, 1000, 5.5, 1000, 3.5, 200]

>>> a.append(5.5)
>>> a
[7, 7, 1000, 5.5, 1000, 3.5, 200, 5.5]

>>> a += [0, 0]
>>> a
[7, 7, 1000, 5.5, 1000, 3.5, 200, 5.5, 0, 0]

>>> b = a.pop()
>>> b
0
>>> a
[7, 7, 1000, 5.5, 1000, 3.5, 200, 5.5, 0]
```

```
>>> a.sort()
>>> a
[0, 3.5, 5.5, 5.5, 7, 7, 200, 1000, 1000]

>>> a.remove(1000)
>>> a
[0, 3.5, 5.5, 5.5, 7, 7, 200, 1000]

>>> del a[2:4]
>>> a
[0, 3.5, 7, 7, 200, 1000]

>>> a.reverse()
>>> a
[1000, 200, 7, 7, 3.5, 0]

# 2. Сравнения и равенство
# Простое сравнение
>>> a = [1, 5, 10]
>>> b = [1, 5, 10]
>>> a == b
True

>>> b[0] = 5
>>> b
[5, 5, 10]
>>> a < b
True

# Вложенное сравнение
>>> a[0] = [3, "aaa"]
>>> b[0] = [3, "bb"]
>>> a, b
([[3, 'aaa'], 5, 10], [[3, 'bb'], 5, 10])
>>> a < b
True
```

Последовательности. Кортеж

- Кортеж (tuple) - это упорядоченная неизменяемая последовательность элементов.
- Особенности: умеет все, что умеет список, за исключением операций, приводящих к изменению кортежа. Применяется в случаях, когда известно, что последовательность не будет меняться после создания.

Последовательности. Кортеж

```
# 1. Пустой кортеж создается с помощью пустых круглых скобок или функции tuple()
>>> ()
()
>>> tuple()
()

# 2. Инициализировать кортеж элементами можно одним из следующих способов:
>>> 1,
(1,)
>>> 1, 2, "text"
(1, 2, 'text')
>>> s = tuple("text")
>>> s
('t', 'e', 'x', 't')
>>>

# 3. Т.к. структура является неизменяемой, изменение содержимого запрещено
>>> s[0] = "n"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Кортежи поддерживают все операции, общие для последовательностей.

Последовательности. Числовой диапазон

Числовой диапазон (range) - это упорядоченная неизменяемая последовательность элементов - целых чисел.

- range(stop)
- range(start, stop[, step])

Конструктор класса range.

Параметры

- start (int) – начальное значение (по умолчанию 0);
- stop (int) – конечное значение (не включается в результат);
- step (int) – шаг изменения (по умолчанию 1, может быть отрицательным).

```
# 10 чисел (от 0 до 9), начиная с 0 с шагом 1
>>> tuple(range(10))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# 10 чисел (от 1 до 10), начиная с 1 с шагом 1
>>> tuple(range(1, 11))
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# Числа от 0 до 19 с шагом 5
>>> tuple(range(0, 20, 5))
(0, 5, 10, 15)

# Числа от 0 до 20 с шагом 3
>>> tuple(range(0, 20, 3))
(0, 3, 6, 9, 12, 15, 18)

# Числа от 0 до -9 с шагом -1
>>> tuple(range(0, -10, -1))
(0, -1, -2, -3, -4, -5, -6, -7, -8, -9)

# Следующие 2 объекта range не содержат чисел (нет чисел от 0 до -1 с шагом 1)
>>> tuple(range(0))
()
>>> tuple(range(1, 0))
()

>>> tuple(range(1, 0, -1))
(1,)
```

Создание числового диапазона

Числовые диапазоны поддерживают те же операции, что и кортежи.

Множества

Множество - это неупорядоченная коллекция уникальных элементов.

В Python существует 2 класса для работы с множествами:

- set (изменяемое множество);
- frozenset (неизменяемое множество).

Элементы множества должны быть хешируемы. Например, списки и словари – это изменяемые объекты, которые не могут быть элементами множеств. Большинство неизменяемых типов в Python (int, float, str, bool, и т.д.) – хешируемые. Неизменяемые коллекции, например tuple, являются хешируемыми, если хешируемы все их элементы.

Наиболее часто множества используются для эффективной проверки на вхождение, удаления повторяющихся элементов а также выполнения математических операций, характерных для математических множеств (пересечение, объединение и др.)

Множества

- Оба типа обладают различиями, схожими с различиями между списком и кортежем.

```
# 1. Пустое множество создается с помощью функции set()
>>> set()
set()

# 2. Инициализировать множество элементами можно, используя:
#    - фигурные скобки с перечислением элементов;
#    - функцию set(), передав в качестве аргумента любой итерируемый объект.
#    Следует обратить внимание, что т.к. множество - неупорядоченный набор данных,
#    при выводе порядок его элементов может быть произвольным.
>>> {"a", "b", "c"}
{'c', 'b', 'a'}
>>> set([1, 2, 3, 4, 5])
{1, 2, 3, 4, 5}
```

Множества. Общие операции

Пусть st - множество, на котором вызывается метод.

- add(elem)

Добавляем элемент elem в множество st.

- remove(elem)

Удаляет элемент elem из множества st. Если элемент не находится в множестве, возникает ошибка.

- discard(elem)

Удаляет элемент elem из множества st, если он присутствует в множестве.

- pop()

Удаляет произвольный элемент из множества st и возвращает в качестве результата.

- clear()

Удаляет все элементы из множества.

Множества. Математические операции

Множества поддерживают математические операции, характерные для множеств (пересечение, объединение и др.).

Пусть st - множество, на котором вызывается метод.

union(other, ...)

st | other | ...

Возвращает новое множество - объединение множеств st и other.

intersection(other, ...)

st & other & ...

Возвращает новое множество - пересечение множеств st и other.

difference(other, ...)

st - other - ...

Возвращает новое множество - разность множеств st и other.

isdisjoint(other)

Возвращает True если st не содержит общий элементов с other.

issubset(other)

st <= other

Возвращает True если все элементы st содержатся в other.

Множества. Математические операции

st < other

Аналогично st <= other, но множества не должны полностью совпадать.

issuperset(other)

st >= other

Возвращает True если все элементы other содержатся в st.

st > other

Аналогично st >= other, но множества не должны полностью совпадать.

update(other, ...)

st |= other | ...

Добавляет элементы из other в st.

```
>>> a = {2, 4, 6, 8, 10}
>>> b = set(range(11))

>>> a
{8, 10, 2, 4, 6}
>>> b
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>>

>>> b.remove(0)
>>> b
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

>>> a.add(12)
>>> a
{2, 4, 6, 8, 10, 12}

>>> a & b
{8, 2, 10, 4, 6}

>>> a | b
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}

>>> a - b
{12}

>>> b - a
{1, 9, 3, 5, 7}
```

Пример работы с множеством

Отображения

- Отображение - это неупорядоченная коллекция пар элементов «ключ-значение». В разных языках синонимом отображений являются термины словарь, хеш-таблица или ассоциативный массив.
- Отображения в Python представлены единственным типом dict (словарь), в котором в качестве ключа может выступать любой хешируемый объект, а в качестве значения - произвольный объект.
- Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется словарем или ассоциативным массивом. Соответствующая структура данных в языке Питон называется dict.

```
# 1. Пустой словарь создается с помощью {} или функции dict()
>>> {}
{}
>>> dict()
{}

# 2. Инициализировать словарь элементами, используя:
#    - фигурные скобки с перечислением элементов в виде 'ключ: значение';
#    - функцию dict(), передав набор пар 'ключ: значение'.
#    Следует обратить внимание, что т.к. множество - неупорядоченный набор данных,
#    при выводе порядок его элементов может быть произвольным.
>>> {"one": 1, "two": 2, "three": 3}
{'two': 2, 'one': 1, 'three': 3}
>>> dict(one=1, two=2, three=3)
{'two': 2, 'one': 1, 'three': 3}
```

Словарь. Операции

Пусть d - словарь, на котором вызывается метод.

- d[key]

Возвращает значение словаря для ключа key. Если ключ не существует, возникает ошибка.

- get(key[, default])

Возвращает значение словаря для ключа key. Если ключ не существует возвращается значение default или None.

- d[key] = value

Устанавливает значение словаря по ключу key. Если ключ не существует, он создается.

Словарь. Операции

- items()

Возвращает набор пар «ключ-значение» для словаря d.

- keys()

Возвращает набор ключей для словаря d.

- values()

Возвращает набор значений для словаря d.

- clear()

Удаляет из словаря все элементы.

- del d[key]

Удаляет пару «ключ-значение» на основании ключа key.

Словарь. Операции

```
>>> phonebook = {"Петров Петр": "+79102222222"}
>>> phonebook["Иванов Сергей"] = "+79101111111"
>>> phonebook
{'Иванов Сергей': '+79101111111', 'Петров Петр': '+79102222222'}

>>> phonebook["Петров Петр"]
'+79102222222'

# Обновили номер телефона
>>> phonebook["Петров Петр"] = "+79103333333"
>>> phonebook
{'Иванов Сергей': '+79101111111', 'Петров Петр': '+79103333333'}

>>> "Васильев Василий" in phonebook
False

>>> phonebook.get("Васильев Василий", "Номер не найден")
'Номер не найден'

>>> phonebook.keys()
dict_keys(['Иванов Сергей', 'Петров Петр'])
>>> phonebook.values()
dict_values(['+79101111111', '+79103333333'])
```

Общие функции

Все объекты независимо от типа поддерживают ряд общих функций.

- help([object])

Отображает справку для object.

- type(object)

Возвращает тип object.

Проверка типов

- Для того, чтобы проверить, какой тип имеет тот или иной объект можно воспользоваться функциями type() или isinstance() - использование последней для проверки типа более предпочтительно.

```
>>> a = 5
>>> type(a) is int
True
>>> isinstance(a, int)
True
# isinstance может принимать вторым параметром кортеж проверяемых типов
>>> isinstance(a, (int, float))
True
```

Взаимное преобразование

- Все типы поддерживают взаимное преобразование (где оно имеет смысл, например, преобразование списка в кортеж, но не списка в целое число и т.п.), для чего используется конструктор типа с параметром - объектом, который нужно преобразовать.

```
# 1. Преобразование в строку
#    Строковое представление имеют практически все рассмотренные классы
>>> str(True)
'True'
>>> str(5)
'5'
>>> str(10.43)
'10.43'
>>> str([1, 2, 3, 4, 5])
'[1, 2, 3, 4, 5]'

# 2. Преобразование в целое число
>>> int(10.43)  # int отсекает дробную часть
10
>>> int("5")
5
>>> int(True)
1

# 3. Преобразование в вещественное число
>>> float(5)
5.0
>>> float("10.43")
10.43

# 4. Преобразование в логический тип
#    Всегда возвращает False, для:
#    - None, False;
#    - нулевых чисел;
#    - пустых последовательностей и отображений.
#    - ...
>>> bool(None), bool(0), bool(0.0), bool(""), bool({})
(False, False, False, False, False)
>>> bool(5), bool({1: "первый"})
(True, True)
```

```
# 5. Преобразования последовательностей
>>> tuple([1, 2, 3])
(1, 2, 3)

>>> d = dict(one=1, two=2, three=2)
>>> list(d.keys())  # Получаем список ключей
['one', 'three', 'two']
>>> set(d.values())  # И множество значений
{1, 2}
>>>
```

Приоритет операций

- Операции над объектами выполняются в определенном порядке:

1. `**`.
2. `-x`, `+x`.
3. `*`, `/`, `//`, `%`.
4. `+`, `-`.
5. `<`, `<=`, `>`, `>=`, `!=`, `==`.
6. `is`, `is not`.
7. `in`, `not in`.
8. `not`, `and`, `or`.

- Изменение порядка можно производить за счет использования скобок, например: (5 + 2) * 3 или (3 * 2)**3.

Поверхностное и глубокое копирование

- Оператор присваивания копирует ссылку на объект, создавая т.н. поверхностную копию. В ряде случае необходимо создать полную копию объекта (глубокую копию), например, для мутирующих коллекций, чтобы после изменять новую коллекцию без изменения оригинала.

```
# 1. Поверхностная и глубокая копии
>>> x = [53, 68, ["A", "B", "C"]]

>>> x1 = x  # Поверхностная копия (через присваивание)
>>> x2 = x[:]  # Глубокая копия (создается при срезе)
>>> x3 = x.copy()  # Глубокая копия (через метод copy())
>>>
>>> id(x), id(x1), id(x2), id(x3)
(4813768, 4813768, 4813848, 4813808)
>>> x1 is x, x2 is x, x3 is x
(True, False, False)

# 2. Присваивание копирует ссылки на объекты, создавая объекты при необходимости
#    Проверить можно с помощью функции id()
>>> a = 5
>>> b = a
>>> a, b
(5, 5)
>>> id(a), id(b)
(1431495600, 1431495600)

>>> c = 5
>>> id(a), id(c)
(1431495600, 1431495600)
```

```
# При изменении значения 'a', Python не изменяет объект 5
# (оставляя его "как есть", т.к. знает, что он используется другими),
# а создает новый, меняя ссылку у 'a', при этом прочие объекты продолжают ссылаться
>>> a = 10
>>> id(a), id(c)
(1431495680, 1431495600)

# Но с мутирующими типами (например, списком) Python поступает по-другому
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> c = a
>>> id(a), id(b), id(c)
(30431712, 30447736, 30431712)

# При изменении мутирующего типа "изменяются" и указывающие на него объекты -
# т.к. они хранят ссылку на тот же объект!
>>> a[0] = 5
>>> a, b, c
([5, 2, 3], [1, 2, 3], [5, 2, 3])
```

Константы

- В Python не существует привычного для, например, Си или Паскаля понятия константы. Вместо этого, значение, которое подразумевается как константа, обозначается заглавными буквами (MEMORY_MAX = 1024), визуально предупреждая, что данное значение менять не следует.

Сортировка

Функция sorted() позволяет получить отсортированный объект в виде списка.

```
# 1. Простой список
>>> lst = [1, 8, 2, 5, 0, 3]
>>> sorted(lst)
[0, 1, 2, 3, 5, 8]
>>> sorted(lst, reverse=True)
[8, 5, 3, 2, 1, 0]

# 2. Словарь
# Для словаря sorted() возвращает отсортированный список ключей
>>> phones = {'Иван': '+74951111111',  'Сергей': '+74951111113',  'Кирилл': '+74951111112'}
>>> sorted(phones)
['Иван', 'Кирилл', 'Сергей']

# 3. Сложная сортировка

# Список кортежей: номер элемента, обозначение, название
>>> elements = [(1, "H", "Водород"), (8, "O", "Кислород"), (53, "I", "Йод")]

# Используя параметр key, sorted() позволяет сортировать список по
# необходимой части коллекции
# Сортировка по наименованию (элемент с индексом 2 в кортеже)
>>> sorted(elements, key=lambda item: item[2])
[(1, 'H', 'Водород'), (53, 'I', 'Йод'), (8, 'O', 'Кислород')]

# Список чисел
>>> nums = [123, 100, 1001, 234, 515]

# Сортировка по последней цифре числа по убыванию
>>> sorted(nums, key=lambda item: item % 10, reverse=True)
[515, 234, 123, 1001, 100]
```