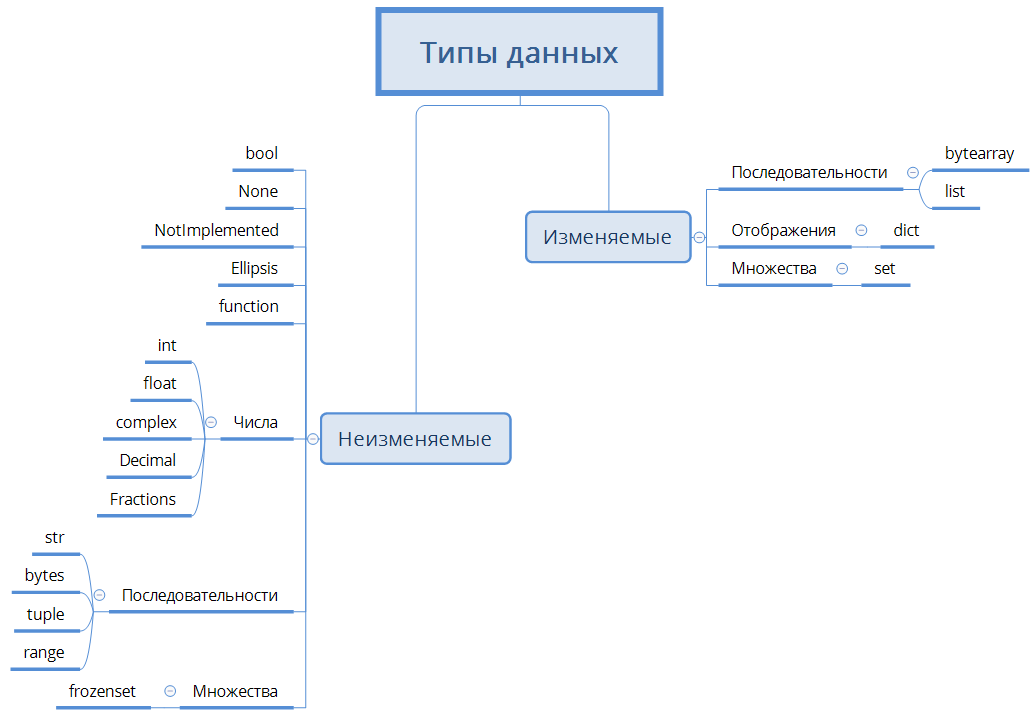
# Введение

Изменяемые и неизменяемые типы данных



Переменные в Python не требуют объявления типа переменной (так как Python – язык с динамической типизацией) и являются ссылками на область памяти. Правила именования переменных:

* имя переменной может состоять только из букв, цифр и знака подчёркивания;
* имя не может начинаться с цифры;
* имя не может содержать специальных символов @, $, %.

# Урок №1.

1. Hello World!

*# Первая программа  
#  
# Если в Python 2 скобки можно не использовать,  
# то в Python3 они обязательны.  
# Если их не указать, то будет вызвана синтаксическая ошибка.  
# Функция print выводит все элементы, разделяя их значением sep,  
# и завершает вывод значением end.  
# end по умолчанию использует перевод строки \n  
  
# Вывод: Hello, World!*print(**'Hello, World!'**)  
*# Вывод: Hello, World!*print(**'Hello,'**, **'World!'**, sep=**' '**) *# Разделитель пробел  
# Вывод: Hello,World!*print(**'Hello,'**, **'World!'**, sep=**''**) *# Разделителя нет  
# Вывод: Hello, World!*print(**'Hello, World'**, end=**'!\n'**) *# Окончание вывода с ! и переходом на новую строчку  
# Вывод: Hello, World!!*print(**'Hello,'**, **'World!'**, sep=**' '**, end=**'!'**) *# Разделитель пробел, окончание вывода с !  
  
# Для печати спец. символов нужно использоать экранирование \*print()  
*# Вывод: \t*print(**'\\t'**)  
*# Вывод: \n*print(**'\\n'**)  
*# Вывод: \*print(**'\\'**)

2. Объявление и разница строк

*# Строки  
# Одинарные и двойные кавычки равны между собой*my\_string\_o = **'Hello, World!'**my\_string\_d = **"Hello, World!"***# Вывод: Hello, World!*print(my\_string\_o)  
*# Вывод: Hello, World!*print(my\_string\_d)  
  
*# Тройные кавычки сохраняют форматирование*my\_string\_to = **''' Hello, World!'''**my\_string\_td = **"""  
 Hello, World!"""***# Вывод: Hello, World!*print(my\_string\_to)  
*# Вывод: ----------*print(**'-'** \* 10) *# Напечатать символ 10 раз  
# Вывод:   
# Hello, World!*print(my\_string\_td)  
  
*# Форматирование без тройных кавычек*my\_string\_of = **"\t\tHello, World!"** *# Повторение форматирования на строчке 9  
# Вывод: Hello, World!*print(my\_string\_of)  
*# Вывод: ----------*print(**'-'** \* 10) *# Напечатать символ 10 раз*my\_string\_df = **"\n\tHello, World!"** *# Повторение форматирования на строчке 10  
# Вывод:   
# Hello, World!*print(my\_string\_df)  
  
*# Использование кавычек внутри строки  
# Можно использовать одинарные кавычки внутри двойных и наоборот  
# Вывод: "Hello, World!"*print(**'"Hello, World!"'**)  
*# Вывод: 'Hello, World!'*print(**"'Hello, World!'"**)  
*# Либо использовать экранирование \  
# Вывод: 'Hello, World!'*print(**'\'Hello, World!\''**)  
  
*# Операции над строками  
# Вывод: HelloWorld*print(**'Hello'** + **'World'**)  
*# Вывод: HelloHelloHelloWorldWorld*print(**'Hello'** \* 3 + **'World'** \* 2)

3. Форматирование строк

*# Форматирование строк  
# Существует несколько способов форматирования строк*print(**'% / Python 2'**)  
*# Старый способ / Python 2  
# Раньше форматирование строк происходило при помощи оператора %*hello = **"Hi, my name is %s"** % **"Jane"***# Вывод: Hi, my name is Jane*print(hello)  
  
hello = **"Hi, my name is %s %s"** % (**"Jane"**, **"Doe"**)  
*# Вывод: Hi, my name is Jane Doe*print(hello)  
  
print(**'Format'**)  
*# Использование метода format  
# Подстановка по порядку*names = **"{}, {} and {}"**.format(**"John"**, **"Bob"**, **"Jane"**)  
*# Вывод: John, Bob and Jane*print(names)  
  
*# Подстановка по позиционному (positional) аргументу*names = **"{2}, {0} and {1}"**.format(**"John"**, **"Bob"**, **"Jane"**)  
*# Вывод: Jane, John and Bob*print(names)  
  
*# Подстановка по аргументу ключевому (keyword) слову*names = **"{a}, {c} and {b}"**.format(a=**"John"**, b=**"Bob"**, c=**"Jane"**)  
*# Вывод: John, Jane and Bob*print(names)  
  
print(**'f-строки'**)  
*# Использование f-строк*names = **f"{'John'}, {'Bob'} and {'Jane'}"***# Вывод: John, Bob and Jane*print(names)  
  
john = **'John'**bob = **'Bob'**jane = **'Jane'**names = **f'{**jane**}, {**bob**} and {**john**}'***# Вывод: Jane, Bob and John*print(names)  
  
print(**'-'** \* 10)  
  
*# Примеры форматирования с format и f-строками  
# Вывод числа в разных форматах  
# Вывод: int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010*print(**"int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}"**.format(42))  
*# Вывод: int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010*print(**f"int: {**42**:d}; hex: {**42**:#x}; oct: {**42**:#o}; bin: {**42**:#b}"**)  
  
*# Выравнивание строк  
# Вывод: Cat*print(**"{:15}"**.format(**'Cat'**)) *# Лево  
# Вывод: Cat*print(**"{:>15}"**.format(**'Cat'**)) *# Право  
# Вывод: Cat*print(**"{:^15}"**.format(**'Cat'**)) *# Центр  
# Вывод: \*\*\*\*\*\*Cat\*\*\*\*\*\**print(**"{:\*^15}"**.format(**'Cat'**)) *# Центр с заполнением символа \*  
  
# Вывод: Cat*print(**f"{'Cat':15}"**)  
*# Вывод: Cat*print(**f"{'Cat':>15}"**)  
*# Вывод: Cat*print(**f"{'Cat':^15}"**)  
*# Вывод: \*\*\*\*\*\*Cat\*\*\*\*\*\**print(**f"{'Cat':\*^15}"**)  
  
*# Выравнивание чисел  
# Целочисленные  
# Вывод: 12*print(**"{:5d}"**.format(12)) *# Отступ с шириной  
# Вывод: 12345*print(**"{:2d}"**.format(12345)) *# Отступ не работает, если число больше ширины  
# Дробные  
# Вывод: 012.2457*print(**"{:08.4f}"**.format(12.2456845)) *# Отступ заполненный нулями и ограничение тремя знаками после запятой  
# Вывод: 12.246*print(**"{:2.3f}"**.format(12.2456845)) *# Отступ заполненный нулями и ограничение тремя знаками после запятой*

4. Срезы

*# Срезы строк  
# Можно использовать только целочисленные значения 0, 1, 2 ...  
# Нельзя 0,5; 2.3 ...*str = **'Hello, Python!'***#  
# Печать первого символа*print(**f'str[0] = {**str[0]**}'**) *# Вывод: H  
# Печать последнего символа*print(**f'str[-1] = {**str[-1]**}'**) *# Вывод: !  
  
# TypeError: string indices must be integers*print(**f'str[0.5] = {**str[0.5]**}'**)  
  
*# IndexError: string index out of range. Выход за пределы строки*print(**f'str[15] = {**str[15]**}'**)  
*# IndexError: string index out of range. Выход за пределы строки*print(**f'str[-15] = {**str[-15]**}'**)  
  
*# Стрез со второго элемента по четвертый  
# Отсчет начинается с 0*print(**f'str[1:5] = {**str[1:5]**}'**) *# Вывод: str[1:5] = ello  
  
# Срез с шестого по второй элемент*print(**f'str[5:-2] = {**str[5:-2]**}'**) *# Вывод: str[5:-2] = , Pytho  
  
# Отрицательный срез*print(**f'str[-6:-2] = {**str[-6:-2]**}'**) *# Вывод: str[-6:-2] = ytho*

5. Изменение строк

*# Изменение строк  
# Строки являются неизменяемым типом данных  
# Нельзя изменить символы, нельзя удалять символы*str = **'Hello, World!'**str[0] = **'Q'  
del** str[3]

6. Ввод данных с клавиатуры

*# Ввод данных с клавиатуры  
# Данные возвращаются в строковом формате  
# Функция input позволяет вывод сообщения без вызова функции print*name = input(**"Name?\n"**)  
print(**"Hello"**, name)  
print(**"Hello "** + name)  
print(**f"Hello {**name**}"**)  
print(**"Hello {}"**.format(name))  
print(**"Hello %s"** % name)  
  
*# Функция type позволяет узнать тип переменной (имени)  
# <class 'str'>*print(type(name))

7. Приведение типов. Числа.

*# Конвертация str в int*number = input(**'Введите число: '**) *# 9*print(type(number)) *# Вывод: <class 'str'>*int\_n = int(number)  
print(int\_n) *# Вывод: 9*print(type(int\_n)) *# Вывод: <class 'int'>  
  
# Тип данных int  
# Числа в Python сразу могут в длинную математику*print(7 + 8) *# Сложение. Вывод: 15*print(7 - 8) *# Вычитание. Вывод: -1*print(7 \* 8) *# Умножение. Вывод: 56  
# Деление типов int всегда возвращает float, даже если деление нацело*print(7 / 7) *# Деление. Вывод: 1.0*print(8 // 7) *# Деление, остаток отсекается. Вывод: 1*print(8 % 7) *# Остаток от деления. Вывод: 1*print(8 \*\* 8) *# Возведение в степень. Вывод: 16777216  
  
# Тип данных Float*print(7.5 + 3.8) *# Сложение. Вывод: 11.3*print(7.5 - 3.8) *# Вычитание. Вывод: 3.7*print(7.5 \* 3.8) *# Умножение. Вывод: 28.5*print(7.5 / 7.5) *# Деление. Вывод: 1.0*print(3.8 // 7.5) *# Деление, остаток отсекается. Вывод: 0.0*print(3.8 % 7.5) *# Остаток от деления. Вывод: 3.8*print(3.8 \*\* 7.5) *# Возведение в степень. Вывод: 22303.703067314564  
  
# Конвертация int <-> float*print(int(2.6)) *# Вывод: 2. Дробная часть отсекается*print(float(2)) *# Вывод: 2.0  
  
# tbc round*

8. Bool

*# Bool  
# Логический тип представлен двумя постоянными значениями False и True.  
# Значения используются для представления истинности.  
# Можно представить, как False - 0, True - 1  
# Подтип int*print(bool(0)) *# Вывод: False*print(bool(1)) *# Вывод: True  
  
# Сравнение  
# В данном примере возвращается True/False в зависимости от равенства, if использовать необязательно*print(**True** == 0) *# Вывод: False*print(**False** == 1) *# Вывод: False  
  
# Можно сравнивать с другими числами*print(**True** > 0) *# Вывод: True*print(**False** <= 0) *# Вывод: True*print(**True** > 2) *# Вывод: True*print(**False** <= 10) *# Вывод: True  
  
# Bool с другими значениями*print(bool(**''**)) *# Вывод: False*print(bool(**'string'**)) *# Вывод: True*print(bool(10)) *# Вывод: True*print(bool(-10)) *# Вывод: True*print(bool(5.5)) *# Вывод: True*print(bool(-9.8)) *# Вывод: True*

# Урок №2

9. Условия

*# Условная инструкция if-elif-else (оператор ветвления)  
# Любое число, не равное 0, или непустой объект - истина.  
# Числа, равные 0, пустые объекты и значение None - ложь  
# Операции сравнения применяются к структурам данных рекурсивно  
# Операции сравнения возвращают True или False  
# Логические операторы and и or возвращают истинный или ложный объект-операнд  
  
# Простая программа на понимание условий. Если введенное число == 5, то выводится надпись "Вы угадали число!"*print(**'Угадай число'**)  
a = int(input(**'Введите число: '**))  
*# 5 - искомое число***if** a == 5:  
 print(**'Вы угадали число!'**)  
**else**:  
 print(**'Введите другое число!'**)  
  
*# Обновленная версия с несколькими числами*print(**'Угадай число'**)  
a = int(input(**'Введите число: '**))  
*# 5, 55 - искомые число***if** a == 5:  
 print(**'Вы угадали число!'**)  
**elif** a == 55:  
 print(**'Вы угадали число!'**)  
**else**:  
 print(**'Введите другое число!'**)  
  
*# Использование или.*print(**'Угадай число'**)  
a = int(input(**'Введите число: '**))  
*# 5, 55 - искомые число***if** a == 5 **or** a == 55:  
 print(**'Вы угадали число!'**)  
**else**:  
 print(**'Введите другое число!'**)  
  
*# Использование и.*print(**'Угадай число'**)  
a = int(input(**'Введите число: '**))  
b = int(input(**'Введите число: '**))  
*# 5, 55 - искомые число***if** a == 5 **and** b == 55:  
 print(**'Вы угадали число!'**)  
**else**:  
 print(**'Введите другое число!'**)  
  
*# Что будет если пользователь введет не число, а букву  
# ValueError: invalid literal for int() with base 10: 'f'*print(**'Угадай число'**)  
a = input(**'Введите число: '**)  
b = input(**'Введите число: '**)  
**if not** a.isdigit() **or** b.isdigit():  
 print(**'Введите число, а не строчку!'**)  
 exit()  
*# 5, 55 - искомые число***if** int(a) == 5 **and** int(b) == 55:  
 print(**'Вы угадали число!'**)  
**else**:  
 print(**'Введите другое число!'**)  
  
*# или*print(**'Угадай число'**)  
a = input(**'Введите число: '**)  
b = input(**'Введите число: '**)  
**if** a.isdigit() **and** b.isdigit():  
 *# 5, 55 - искомые число* **if** int(a) == 5 **and** int(b) == 55:  
 print(**'Вы угадали число!'**)  
 **else**:  
 print(**'Введите другое число!'**)  
**else**:  
 print(**'Введите число, а не строчку!'**)

10. Цикл for

*# Цикл For  
# для <каждого элемента> в <последовательности>:  
# <выполняем тело цикла>***for** index **in** range(10):  
 print(index) *# Вывод: 0 1 2 3 4 5 6 7 8 9*print()  
*# Итерация по списку  
# C-подобная*nums = [1, 2, 3, 4, 5, 6, 7]  
*# Функция len позволяет узнать размер (длину) списка***for** i **in** range(len(nums)):  
 *# Происходит обращение к списку по индексу* print(nums[i])  
print()  
*# Range  
# Функция range генерирует список от 0 до конца и проходит по нему  
# Синтаксис range(начало, конец, шаг)  
# Шаг может быть положительным, отрицательным, но не 0  
# Положительный***for** index **in** range(0, 10, 2):  
 print(index, end=**''**) *# Вывод: 0 2 4 6 8*print()  
*# Отрицательный  
# Для этого необходимо указать конец > начала***for** index **in** range(20, 10):  
 print(index) *# Вывод: 20 18 16 14 12*print()  
**for** index **in** range(20, 10, -2):  
 print(index) *# Вывод: 18 16 14 12*print()  
*# В Python сущетсвует возможность итерироваться по элементам напрямую***for** num **in** nums:  
 print(num)  
print()  
n = 3  
print(n) *# Вывод: 3  
# Временная переменная n в теле цикла перезаписывает внешнюю переменную n,  
# тк циклы не создают новую область видимости переменных***for** n **in** nums:  
 print(n) *# Вывод: 1 2 3 4 5 6 7*print(n) *# Вывод: 7*print()  
*# Break  
# В Python выражение break дает возможность выйти из цикла при условии.***for** num **in** nums:  
 **if** num == 4:  
 print(**'Выход из цикла'**)  
 **break** print(num),  
print()  
*# Continue  
# В Python выражение continue дает возможность пойти дальше по циклу при условии.***for** num **in** nums:  
 **if** num == 4:  
 print(**'Пропускаем 4'**)  
 **continue** print(num)  
print()  
**for** num **in** nums:  
 **if** num == 3:  
 print(**'Пропускаем 3'**)  
 **continue  
 elif** num == 5:  
 print(**'Выход из цикла'**)  
 **break** print(num)  
  
print()  
*# pass***for** num **in** nums:  
 **if** num == 2:  
 print(**'Ничего не делаем\nНе пропускаем 2 и не выходим из цикла'**)  
 **pass  
 elif** num == 3:  
 print(**'Пропускаем 3'**)  
 **continue  
 elif** num == 5:  
 print(**'Выход из цикла'**)  
 **break** print(num)  
  
*# For ... else  
# For может содержать блок else, который выполнится, только если цикл завершится полностью (без применения break)*print()  
*# Выведется надпись Конец цикла***for** num **in** nums:  
 **if** num == 2:  
 print(**'Ничего не делаем\nНе пропускаем 2 и не выходим из цикла'**)  
 **pass  
 elif** num == 3:  
 print(**'Пропускаем 3'**)  
 **continue  
else**:  
 print(**'Конец цикла'**)  
  
print()  
*# Не выведется надпись Конец цикла, потому что цикл был прерван конструкцией break***for** num **in** nums:  
 **if** num == 2:  
 print(**'Ничего не делаем\nНе пропускаем 2 и не выходим из цикла'**)  
 **pass  
 elif** num == 3:  
 print(**'Пропускаем 3'**)  
 **continue  
 elif** num == 5:  
 print(**'Выход из цикла'**)  
 **break  
else**:  
 print(**'Конец цикла'**)

11. Цикл while

*# Цикл while  
# пока <условаие истинно>:  
# <выполняем тело цикла>*n = 10  
**while** n >= 1:  
 print(n) *# Вывод: 10 9 8 7 6 5 4 3 2 1* n -= 1  
  
print()  
*# Цикл While может содержать блок else, который выполнится, только если цикл завершится полностью (без применения break)*counter = 0  
*# Выведется надпись Конец цикла***while** counter <= 3:  
 print(counter)  
 counter += 1  
**else**:  
 print(**'Конец цикла'**)  
  
print()  
*# Обнуляем счетчик так в нем осталось значение от предыдущего цикла*counter = 0  
*# Не выведется надпись Конец цикла***while** counter <= 3:  
 print(counter)  
 **if** counter == 2:  
 **break** counter += 1  
**else**:  
 print(**'Конец цикла'**)  
  
print()  
*# Бесконечный цикл  
# Раскомменитровать для работоспособности  
# while True:  
# print('Бесконечный цикл')*

# Урок №3

12. Списки

*# Списки  
# Список в Python представлят из себя последовательность элементов с возможностью хранить различные типы данных  
# в отличии от массивов  
# Реализация схожа с динамическим массивом  
  
# Объявление пустого списка списка*mylist = []  
mylist = list()  
print(mylist) *# []  
# Список интов*mylist = [1, 2, 3, 4, 5] *# [1, 2, 3, 4, 5]*print(mylist)  
*# Список с различными типами данных*mylist = [1, **'Hello'**, 3.5] *# [1, 'Hello', 3.5]*print(mylist)  
  
*# Вложенный список*mylist = [**'World'**, [1, 2, 3], 4, [5.6, 7.8], [**'H'**], **'ello'**]  
print(mylist) *# ['World', [1, 2, 3], 4, [5.6, 7.8], ['H'], 'ello']  
  
# Индексация и обращение к элементам  
# В [] необходимо указать индекс элемента в формате целого числа  
# При указании индекса большего, чем длина списка будет ошибка выход за пределы списка  
# Индексация начинается с 0  
# 0 1 2 3 4*mylist = [**'h'**, **'e'**, **'l'**, **'l'**, **'o'**]  
print(mylist[0]) *# h*print(mylist[2]) *# l*print(mylist[-1]) *# o  
# print(mylist[1.5]) # TypeError: list indices must be integers or slices, not float  
# print(mylist[99]) # IndexError: list index out of range  
# print(mylist[-1]) # IndexError: list index out of range  
  
# Индексация и обращение к элементам вложенного списка*mylist = [**'World'**, [1, 2, 3], 4, [5.6, 7.8], [**'H'**], **'ello'**]  
print(mylist[0]) *# World*print(mylist[0][1]) *# o*print(mylist[1][1]) *# 2*print(mylist[-3][0]) *# 5.6  
  
# Элементы указанные без запятой, будут считаться одним элементом*mylist = [**'H'**, **'e' 'l' 'l' 'o' 'W'**, **'o'**, **'r' 'l' 'd'**, **'!'**]  
print(mylist) *# ['H', 'elloW', 'o', 'rld', '!']  
  
# Срезы списков*mylist = [**'H'**, **'e'**, **'l'**, **'l'**, **'o'**, **'W'**, **'o'**, **'r'**, **'l'**, **'d'**, **'!'**]  
print(mylist) *# ['H', 'e', 'l', 'l', 'W', 'o', 'r', 'l', 'd', '!']*print(mylist[2:5]) *# ['l', 'l', 'o']  
# От начала до 5 индекса*print(mylist[:5]) *# ['H', 'e', 'l', 'l', 'o']  
# С 2 индекса до конца*print(mylist[2:]) *# ['l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']  
# Полный срез (Копирование списка, об этом в 1\_list\_advanced)*print(mylist[:]) *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']  
# Отрицательный срез*print(mylist[-5:-2]) *# ['o', 'r', 'l']  
# Отрицательный срез до начала до -5 индекса*print(mylist[:-5]) *# ['H', 'e', 'l', 'l', 'o', 'W']  
# Отрицательный срез с -3 индекса до конца*print(mylist[-3:]) *# ['l', 'd', '!']  
  
# Изменение значений  
# Список четных чисел*mylist = [2, 4, 6, 8]  
print(mylist) *# [2, 4, 6, 8]  
# Изменение одного элемента по индексу 0*mylist[0] = 1  
print(mylist) *# [1, 4, 6, 8]  
# Вставка нескольких значений*mylist = [2, 4, 6, 8]  
print(mylist) *# [2, 4, 6, 8]*mylist[0] = [1, 3, 5]  
print(mylist) *# [[1, 3, 5], 4, 6, 8]  
# Для вставки нескольких значений на 0 индекс нужно использовать срезы*mylist = [2, 4, 6, 8]  
mylist[0:0] = [1, 3, 5] *# Изменение значений с 0-0 индекс, значения не перетираются*print(mylist) *# [1, 3, 5, 2, 4, 6, 8]  
  
# Ошибка, нужно передавать только последовательность, например [1, 2, 3]  
# mylist[2:2] = 1 # TypeError: can only assign an iterable  
  
# Изменение значений с 1-4 индекс, значения перетираются*mylist = [2, 4, 6, 8]  
mylist[1:4] = [1, 3, 5]  
print(mylist) *# [2, 1, 3, 5]  
  
# Склейка списков*mylist = [1, 2, 3] + [4, 5, 6]  
print(mylist) *# [1, 2, 3, 4, 5, 6]  
  
# Склеивать можно только одинаковые типы данных  
# mylist = (1, 2, 3) + [4, 5, 6] # TypeError: can only concatenate tuple (not "list") to tuple  
  
  
# Методы списков  
# Список нечетных чисел*mylist = [1, 3, 5]  
print(mylist) *# [1, 3, 5]  
# Метод append добавляет указанный элемент в конец списка*mylist.append(7)  
print(mylist) *# [1, 3, 5, 7]  
# Добавление нескольких значений используя append*mylist = [1, 3, 5]  
print(mylist) *# [1, 3, 5]  
# append добавляет элемент как он есть*mylist.append([7, 9, 11])  
print(mylist) *# [1, 3, 5, [7, 9, 11]]  
  
# Добавление нескольких значений  
# Для добавления множества элементов необходимо использовать метод extend*mylist = [1, 3, 5]  
print(mylist) *# [1, 3, 5]*mylist.extend([7, 9, 11])  
*# mylist.extend(7) # extend ожидает на вход последовательность, поэтому передача одного элемента будет ошбикой*print(mylist) *# [1, 3, 5, 7, 9, 11]  
  
# Вставка одного элемента на определенную позицию методом insert  
# Элемент стоявший на указанной позиции, и все следующие смещаются вправо*mylist = [1, 2, 3, 4]  
*# insert(Индекс, элемент)*mylist.insert(3, 5)  
print(mylist) *# [1, 2, 3, 5, 4]*mylist.insert(-1, 7)  
print(mylist) *# [1, 2, 3, 5, 7, 4]  
  
# Подсчет количества элементов в списке*mylist = [3, 9, 7, 1, 1, 3, 4, 5, 6, 6, 0]  
*# count(Значение)*print(mylist.count(0)) *# 1*print(mylist.count(1)) *# 2*print(mylist.count(6)) *# 2  
  
# Узнать индекс элемента  
# Вернется индекс первого попавшегося элемента*mylist = [3, 9, 7, 1, 1, 3, 4, 5, 6, 6, 0]  
*# index(Значение)*print(mylist.index(0)) *# 10*print(mylist.index(1)) *# 3*print(mylist.index(6)) *# 8  
  
# Создание списка из строки  
# sep - разделитель, по умлочанию пробел указывать его не нужно*mylist = **'H e l l o W o r l d !'**.split() *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*print(mylist)  
mylist = **'H.e.l.l.o.W.o.r.l.d.!'**.split(sep=**'.'**) *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*print(mylist)  
*# maxsplit - максимальное количество разделений*mylist = **'H e l l o W o r l d !'**.split(maxsplit=3) *# ['H', 'e', 'l', 'l o W o r l d !']*print(mylist)  
  
*# Удаление значений из списка методом remove (без возврата значения)  
# remove(элемент) - удаляет первый попавшиеся элемент*mylist = **'H e l l o W o r l d !'**.split() *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*mylist.remove(**'l'**)  
print(mylist) *# ['H', 'e', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']  
  
# Задание - удалить все птовторяющиеся элементы  
  
# Удаление значений из списка методом pop (возврата значения)  
# pop(индекс) - удаляет элемент по индексу*mylist = **'H e l l o W o r l d !'**.split() *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*mylist.remove(**'l'**)  
print(mylist) *# ['H', 'e', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*mylist = **'H e l l o W o r l d !'**.split() *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']*mylist.pop() *# по умолчанию удаляет последний == mylist.pop(-1)*print(mylist) *# ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']*mylist.pop(4)  
print(mylist) *# ['H', 'e', 'l', 'l', 'W', 'o', 'r', 'l', 'd']  
# Возврат значения означает, что можно перехватить удаленное значение*elem = mylist.pop(3)  
print(mylist) *# ['H', 'e', 'l', 'W', 'o', 'r', 'l', 'd']*print(elem) *# l  
  
# Очистка списка*mylist = **'H e l l o W o r l d !'**.split()  
mylist.clear()  
print(mylist) *# []  
  
# Повторение*print([**'Hello'**] \* 3) *# ['Hello', 'Hello', 'Hello']*

12.1 Сортировка списков

*# Сортировка списков  
  
# Сортировка методом sort() (без возврата значений, с текущим объектом)  
# Этот метод берет список и сортирует его. На выходе мы получаем тот же список, только отсортированный.*mylist = [99, 0, 25, 8, 4]  
mylist.sort()  
print(mylist) *# [0, 4, 8, 25, 99]  
  
# Метод sort() может принимать два необязательных аргумента: key и reverse.  
# Значением key выступает функция, которая будет вызываться для каждого элемента в списке.*names = [**"Jessica"**, **"Ben"**, **"Carl"**, **"Jackie"**, **"Wendy"**]  
print(names) *# ['Jessica', 'Ben', 'Carl', 'Jackie', 'Wendy']  
# Таким образом, key=len укажет отсортировать список по длине имен, от наименьшего к наибольшему.*names.sort(key=len)  
print(names) *# ['Ben', 'Carl', 'Wendy', 'Jackie', 'Jessica']  
# reverse - вернуть отсортированный список в обратном порядке*names = [**"Jessica"**, **"Ben"**, **"Carl"**, **"Jackie"**, **"Wendy"**]  
print(names) *# ['Jessica', 'Ben', 'Carl', 'Jackie', 'Wendy']*names.sort(key=len, reverse=**True**)  
print(names) *# ['Jessica', 'Jackie', 'Wendy', 'Carl', 'Ben']  
  
# Сортировка функцией sorted() (с возвратом значений, создание нового объекта)  
# sorted() принимает любые итерируемые объекты(списки, строки, кортежи и т.д.),  
# тогда как метод sort() работает только со списками.*mylist = [99, 0, 25, 8, 4]  
mylist\_sorted = sorted(mylist)  
print(mylist) *# [99, 0, 25, 8, 4]*print(mylist\_sorted) *# [0, 4, 8, 25, 99]  
  
# sorted() тоже может принимать два необязательных аргумента: key и reverse.*names = [**"Jessica"**, **"Ben"**, **"Carl"**, **"Jackie"**, **"Wendy"**]  
names\_sorted = sorted(names, key=len)  
print(names\_sorted) *# ['Ben', 'Carl', 'Wendy', 'Jackie', 'Jessica']*names = [**"Jessica"**, **"Ben"**, **"Carl"**, **"Jackie"**, **"Wendy"**]  
names\_sorted = sorted(names, key=len, reverse=**True**)  
print(names\_sorted) *# ['Jessica', 'Jackie', 'Wendy', 'Carl', 'Ben']  
  
# sorted() создает новый объект, независимый от старого*mylist\_1 = [4, 9, 8, 6]  
mylist\_2 = sorted(mylist\_1)  
mylist\_1[0] = 1  
mylist\_2[0] = 2  
print(mylist\_1) *# [1, 9, 8, 6]*print(mylist\_2) *# [2, 6, 8, 9]*

12.2. Списки продолжение

*# Изменение значений или пример работы ссылок в Python*mylist\_1 = [**'Hello'**, [1, 3, 5]]  
mylist\_2 = mylist\_1  
print(**f"Список 1: {**mylist\_1**}"**)  
print(**f"Список 2: {**mylist\_2**}"**)  
mylist\_1[0] = **'World'**print(**f"Список 1: {**mylist\_1**}"**)  
print(**f"Список 2: {**mylist\_2**}"**)  
print()  
*# Python создает ссылку для авторого списка на тот же самый объект на который указывает ссылка первого списка  
# Это происходит потому, что интерпретатор языка оптимизирует память и не создает нового объекта для списка 2  
# Решение довольно простое, сделать полный срез списка, тк срезы это создание нового объекта с определенной длиной*mylist\_1 = [**'Hello'**, [1, 3, 5]]  
mylist\_2 = mylist\_1[:]  
print(**f"Список 1: {**mylist\_1**}"**)  
print(**f"Список 2: {**mylist\_2**}"**)  
mylist\_1[0] = **'!'**print(**f"Список 1: {**mylist\_1**}"**)  
print(**f"Список 2: {**mylist\_2**}"**)  
  
  
*# Продвинутые срезы mylist[START:STOP:STEP]*mylist = [**'H'**, **'e'**, **'l'**, **'l'**, **'o'**, **'W'**, **'o'**, **'r'**, **'l'**, **'d'**, **'!'**]  
print(mylist) *# ['H', 'e', 'l', 'l', 'W', 'o', 'r', 'l', 'd', '!']*print(mylist[2:5:2]) *# ['l', 'o']*print(mylist[::3]) *# ['H', 'l', 'o', 'd']  
  
  
# Списковые включения, генераторы списков или list comprehensions  
# Пример генерации списка с кубами чисел*pow\_ = []  
**for** x **in** range(10):  
 pow\_.append(x \*\* 2)  
print(pow\_) *# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
  
# Пример работы генератора списков*pow\_list = [x \*\* 2 **for** x **in** range(10)]  
print(pow\_list) *# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
  
# Пример работы генератора списков if*pow\_list = [x \*\* 2 **for** x **in** range(10) **if** x % 2 == 0]  
print(pow\_list) *# [0, 4, 16, 36, 64]  
  
# Пример работы генератора словарей в списке*pow\_list = [{x: x \*\* 2} **for** x **in** range(10)]  
print(pow\_list) *# [{0: 0}, {1: 1}, {2: 4}, {3: 9}, {4: 16}, {5: 25}, {6: 36}, {7: 49}, {8: 64}, {9: 81}]*

13. Ключевое слово in

# Урок №4

14. Кортежи

*# Кортежи  
# Кортежи (tuple) — это также упорядоченные наборы элементов.  
# От списков их отличает одно: кортежи неизменяемы.  
# То есть, после объявления кортежа изменить его элементы мы не сможем.  
# Кортежи намного быстрее списков. Для получения доступа к элементам кортежа мы можем использовать индексы и срезы.  
  
# Объявление пустого кортежа*mytuple = ()  
mytuple = tuple()  
print(mytuple) *# ()  
  
# Кортеж интов*mytuple = (1, 2, 3, 4, 5) *# (1, 2, 3, 4, 5)*print(mytuple)  
  
*# Кортеж с различными типами данных*mytuple = (1, **'Hello'**, 3.5) *# (1, 'Hello', 3.5)*print(mytuple)  
  
*# При создании кортежа необязательно использовать скобки*mytuple = 1, **'Hello'**, 3.5 *# (1, 'Hello', 3.5)*print(mytuple)  
  
*# Особенность создания котежа с одним элементом!*mytuple = (**'hello'**)  
print(type(mytuple)) *# <class 'str'>*mytuple = (**'hello'**,)  
print(type(mytuple)) *# <class 'tuple'>  
# Без скобок*mytuple = **'hello'**print(type(mytuple)) *# <class 'str'>*mytuple = **'hello'**,  
print(type(mytuple)) *# <class 'tuple'>  
  
# Вложенный кортеж*mytuple = (1, 2, (3, 4), 5) *# (1, 2, (3, 4), 5)*print(mytuple)  
  
*# Распаковка кортежа*mytuple = 3, 4.6, **'world'**a, b, c = mytuple  
print(a, b, c) *# 3 4.6 world  
  
# "Запаковка" кортежа*a = **'a'**b = **'b'**c = **'c'**mytuple = a, b, c  
print(mytuple)  
  
*# Индексация и обращение к элементам*mytuple = (**'h'**, **'e'**, **'l'**, **'l'**, **'o'**)  
print(mytuple[0]) *# h*print(mytuple[2]) *# l*print(mytuple[-1]) *# o  
# print(mytuple[1.5]) # TypeError: tuple indices must be integers or slices, not float  
# print(mytuple[99]) # IndexError: tuple index out of range  
# print(mytuple[-1]) # IndexError: tuple index out of range  
  
# Индексация и обращение к элементам вложенного списка*mytuple = (**'World'**, (1, 2, 3), 4, [5.6, 7.8], (**'H'**), **'ello'**)  
print(mytuple[0]) *# World*print(mytuple[0][1]) *# o*print(mytuple[1][1]) *# 2*print(mytuple[-3][0]) *# 5.6  
  
# Элементы указанные без запятой, будут считаться одним элементом*mytuple = (**'H'**, **'e' 'l' 'l' 'o' 'W'**, **'o'**, **'r' 'l' 'd'**, **'!'**)  
print(mytuple) *# ('H', 'elloW', 'o', 'rld', '!')*print(**'-'**\*80)  
*# Срезы кортежей*mytuple = (**'H'**, **'e'**, **'l'**, **'l'**, **'o'**, **'W'**, **'o'**, **'r'**, **'l'**, **'d'**, **'!'**)  
print(mytuple) *# ('H', 'e', 'l', 'l', 'W', 'o', 'r', 'l', 'd', '!')*print(mytuple[2:5]) *# ('l', 'l', 'o')  
# От начала до 5 индекса*print(mytuple[:5]) *# ('H', 'e', 'l', 'l', 'o')  
# С 2 индекса до конца*print(mytuple[2:]) *# ('l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!')  
# Полный срез (Копирование кортежа, поведение как у списков)*print(mytuple[:]) *# ('H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!')  
# Отрицательный срез*print(mytuple[-5:-2]) *# ('o', 'r', 'l')  
# Отрицательный срез до начала до -5 индекса*print(mytuple[:-5]) *# ('H', 'e', 'l', 'l', 'o', 'W')  
# Отрицательный срез с -3 индекса до конца*print(mytuple[-3:]) *# ('l', 'd', '!')  
  
# Изменение значений  
# Кортеж четных чисел*mytuple = (2, 4, 6, 8)  
print(mytuple) *# [2, 4, 6, 8]  
# Изменение значений в кортеже невозможна  
# mytuple[0] = 1 # TypeError: 'tuple' object does not support item assignment  
  
# Изменение значений списка в кортеже возможна*mytuple = (2, [4, 6], 8)  
mytuple[1][0] = 1  
print(mytuple) *# (2, [1, 6], 8)  
  
# Склейка кортежей*mytuple = (1, 2, 3) + (4, 5, 6)  
print(mytuple) *# (1, 2, 3, 4, 5, 6)  
  
# Склеивать можно только одинаковые типы данных  
# mytuple = (1, 2, 3) + [4, 5, 6] # TypeError: can only concatenate tuple (not "list") to tuple  
  
# Методы кортежей*mytuple = (**'h'**, **'e'**, **'l'**, **'l'**, **'o'**)  
print(mytuple.count(**'l'**)) *# 2*print(mytuple.index(**'e'**)) *# 1  
  
# In*mytuple = (**'h'**, **'e'**, **'l'**, **'l'**, **'o'**)  
print(**'l' in** mytuple) *# True*print(**'a' in** mytuple) *# False  
  
# Повторение*print((**'Hello'**,) \* 3) *# ('Hello', 'Hello', 'Hello')  
  
# Удаление*mytuple = (**'h'**, **'e'**, **'l'**, **'l'**, **'o'**)  
print(sorted(mytuple))  
*# Удаление элемента в кортеже невозможна  
# del mytuple[0] # TypeError: 'tuple' object doesn't support item deletion  
  
# Удаление кортежа***del** mytuple  
*# print(mytuple) # NameError: name 'mytuple' is not defined*

14.1 Кортежи продолжение

*# Продвинутые срезы mytuple[START:STOP:STEP]*mytuple = (**'H'**, **'e'**, **'l'**, **'l'**, **'o'**, **'W'**, **'o'**, **'r'**, **'l'**, **'d'**, **'!'**)  
print(mytuple) *# ('H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!')*print(mytuple[2:5:2]) *# ('l', 'o')*print(mytuple[::3]) *# ('H', 'l', 'o', 'd')  
  
# Кортежные включения, генераторы кортежей или tuple comprehensions  
# Пример генерации списка с кубами чисел  
# Кортеж является неизменяемым типом данных, поэтом данный формат записи не подойдет для создания кортежа*pow\_ = []  
**for** x **in** range(10):  
 pow\_.append(x \*\* 2)  
print(pow\_) *# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
  
# Использование tuple перед генератором обязательно для получения кортежа  
# pow\_list = (x \*\* 2 for x in range(10)) - вернет объект генератора, а не кортежа  
# <generator object <genexpr> at 0x000002F7C20C9AC0>  
# Пример работы генератора кортежей*pow\_list = tuple(x \*\* 2 **for** x **in** range(10))  
print(pow\_list) *# (0, 1, 4, 9, 16, 25, 36, 49, 64, 81)  
  
# Пример работы генератора кортежей if*pow\_list =tuple(x \*\* 2 **for** x **in** range(10) **if** x % 2 == 0)  
print(pow\_list) *# (0, 4, 16, 36, 64)  
  
# Пример работы генератора словарей в кортеже*pow\_list = tuple({x: x \*\* 2} **for** x **in** range(10))  
print(pow\_list) *# ({0: 0}, {1: 1}, {2: 4}, {3: 9}, {4: 16}, {5: 25}, {6: 36}, {7: 49}, {8: 64}, {9: 81})*

15. Множества

*# Множества  
# В Python множества очень похожи на списки, за исключением того факта, что их элементы неизменяемы.  
# Это означает, что вы не можете изменять элементы уже объявленного множества.  
# Но добавлять и удалять элементы из множества можно. (Создается новое множество взамен старого)  
# Другими словами, множество – это изменяемая неупорядоченная группа элементов, в которой сами элементы неизменяемы.  
# Другой особенностью множества является то, что оно может включать в себя элементы разных типов.  
# Это означает, что у вас в одном множестве могут быть числа, строки и и даже кортежи!  
# Самая важная особенность множеств, то что они не хранят дубликаты  
# Хэши элементов рассчиваются при создании множества!  
  
# Объявление пустого множества  
# Использование set необходимо для создания можества*myset = set()  
print(type(myset)) *# <class 'set'>  
  
# Использование пустых {} скобок приведет к созданию словаря, а не множества*myset = {}  
print(type(myset)) *# <class 'dict'>  
  
# Множество интов*myset = {1, 2, 3, 4, 5} *# {1, 2, 3, 4, 5}*print(myset)  
  
*# Множество с различными типами данных*myset = {1, **'Hello'**, 3.5} *# {1, 'Hello', 3.5}*print(myset)  
  
*# Множество с одинаковыми элементами*myset = {1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5} *# {1, 2, 3, 4, 5}*print(myset)  
  
*# Независимо от способа создания множества каждый его элемент должен быть неизменяемым объектом.  
# Поэтому, при добавлении списока в множество, произойдет ошибка, ведь списки изменяемы.  
# myset = {1, '', ['H', 'e', 'l', 'lo'], 3.5} # TypeError: unhashable type: 'list'  
  
# при добавлении множества в множество, произойдет ошибка, ведь множества изменяемы.  
# myset = {'World', {1, 2, 3}, 4, {5.6, 7.8}, {'H'}, 'ello'} # TypeError: unhashable type: 'set'  
  
# Вложенное множество с кортежем*myset = {1, **''**, (**'H'**, **'e'**, **'l'**, **'lo'**), 3.5} *# {'', 1, 3.5, ('H', 'e', 'l', 'lo')}*print(myset)  
  
*# Индексация и обращение к элементам  
# Множества не поддерживают индексацию, тк каждый раз элементы расположены в разном порядке*myset = {**'h'**, **'e'**, **'l'**, **'l'**, **'o'**}  
print(myset)  
*# print(myset[0]) # TypeError: 'set' object is not subscriptable  
  
# Индексация и обращение к элементам вложенного списка, смотрите выше на 43 строчку*myset = {**'World'**, (1, 2, 3), 4, (5.6, 7.8), (**'H'**), **'ello'**}  
print(myset)  
*# print(mylist[1][1]) # TypeError: 'set' object is not subscriptable  
  
# Элементы указанные без запятой, будут считаться одним элементом*myset = {**'H'**, **'e' 'l' 'l' 'o' 'W'**, **'o'**, **'r' 'l' 'd'**, **'!'**}  
print(myset) *# {'o', 'H', '!', 'elloW', 'rld'}  
  
# Множества не поддерживают срезы*myset = {**'H'**, **'e'**, **'l'**, **'l'**, **'o'**, **'W'**, **'o'**, **'r'**, **'l'**, **'d'**, **'!'**}  
print(myset)  
*# print(myset[2:5]) # TypeError: 'set' object is not subscriptable  
  
# Множества не поддерживают склейку, объединение множеств в 2\_set\_operations  
# myset = {1, 2, 3} + {4, 5, 6} # TypeError: unsupported operand type(s) for +: 'set' and 'set'  
  
  
# Методы множеств*myset = {1, 2, 3, 4} *# {1, 2, 3, 4}*print(myset)  
  
*# Добавление элемента в конец*myset.add(5) *# {1, 2, 3, 4, 5}*print(myset)  
  
*# Добавление нескольких элементов  
# update принимает последовательность элементов (кортежи, списки, строки и другие сножества)  
# Список*myset.update([6, 7, 8]) *# {1, 2, 3, 4, 5, 6, 7, 8}*print(myset)  
*# При попытке добавить в множество элемент, который уже есть в множестве, то ничего не изменится  
# Кортеж*myset.update((6, 7, 8)) *# {1, 2, 3, 4, 5, 6, 7, 8}*print(myset)  
*# Сет*myset.update({6, 7, 8}) *# {1, 2, 3, 4, 5, 6, 7, 8}*print(myset)  
*# Строки*myset.update(**'123'**) *# {1, 2, 3, 4, 5, 6, 7, 8, '3', '2', '1'}*print(myset)  
  
*# Удаление значений из множества методом pop  
# pop удаляет первый элемент из множества (случайный), но так как при каждом вызове множества  
# элементы расположены в случайном порядке, то предугадать этот элемент нельзя*myset = set(**'Hello,World!'**) *# {',', 'd', 'r', 'o', '!', 'H', 'e', 'W', 'l'}*print(myset)  
myset.pop() *# 'W'*print(myset) *# {',', '!', 'r', 'e', 'l', 'H', 'd', 'o'}  
  
# Удаление значений из множества методом remove*myset = set(**'Hello,World!'**)  
myset.remove(**'l'**) *# {'r', 'e', 'o', 'H', 'W', ',', '!', 'd'}*print(myset)  
*# При попытке удаления несуществующего элемента будет вызвано исключение KeyError  
# myset.remove('l') # KeyError: 'l'  
  
# Удаление значений из множества методом discard*myset = set(**'Hello,World!'**)  
myset.discard(**'l'**) *# {'d', 'H', 'W', '!', 'r', 'o', 'e', ','}*print(myset)  
*# При попытке удаления несуществующего элемента не будет вызвано исключение*myset.discard(**'l'**) *# {'d', 'H', 'W', '!', 'r', 'o', 'e', ','}*print(myset)  
  
*# Очистка множества*myset.clear() *# set()*print(myset)

15.1 Операции множеств

*# Операции с множествами*a = {1, 2, 3, 4, 5}  
b = {4, 5, 6, 7, 8}  
  
*# Объединение  
# Вввод: {1, 2, 3, 4, 5, 6, 7, 8}*print(a | b)  
print(a.union(b))  
print(b.union(a))  
  
*# Пересечение  
# Вввод: {4, 5}*print(a & b)  
print(a.intersection(b))  
print(b.intersection(a))  
  
*# Разница  
# Вввод: {4, 5}*print(a - b) *# {1, 2, 3}*print(a.difference(b)) *# {1, 2, 3}*print(b.difference(a)) *# {8, 6, 7}*print(b - a) *# {8, 6, 7}  
  
# Симметричная разница  
# Вввод: {1, 2, 3, 6, 7, 8}*print(a ^ b)  
print(a.symmetric\_difference(b))  
print(b.symmetric\_difference(a))  
  
*# Остальные операции  
# возвращает True, если a является подмножеством b  
# Вввод: False*print(a <= b)  
print(a.issubset(b))  
  
*# возвращает True, если a является надмножеством b  
# Вввод: False*print(a >= b)  
print(a.issuperset(b))  
  
*# возвращает True, если в множествах a и b нет общих элементов*print(a.isdisjoint(b)) *# False*a = {1, 2, 3}  
b = {4, 5, 6, 7, 8}  
print(a.isdisjoint(b)) *# True*