Для работы с наборами данных Python предоставляет такие встроенные типы как списки, кортежи и словари.

**Список**

Список (list) представляет тип данных, который хранит набор или последовательность элементов. Для создания списка в квадратных скобках ([]) через запятую перечисляются все его элементы. Во многих языках программирования есть аналогичная структура данных, которая называется массив. Например, определим список чисел:

|  |
| --- |
| numbers = [1, 2, 3, 4, 5] |

Также для создания списка можно использовать конструктор **list()**:

|  |
| --- |
| numbers1 = []  numbers2 = list() |

Оба этих определения списка аналогичны - они создают пустой список.

Конструктор list для создания списока может принимать другой список:

|  |
| --- |
| numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(numbers) |

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть второй элемент будет иметь индекс 1. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

|  |
| --- |
| numbers = [1, 2, 3, 4, 5]  print(numbers[0])   # 1  print(numbers[2])   # 3  print(numbers[-3])  # 3    numbers[0] = 125  # изменяем первый элемент списка  print(numbers[0])   # 125 |

Если необходимо создать список, в котором повторяется одно и то же значение несколько раз, то можно использовать символ звездочки \*. Например, определим список из шести пятерок:

|  |
| --- |
| numbers = [5] \* 6  # [5, 5, 5, 5, 5, 5]  print(numbers) |

Кроме того, если нам необходим последовательный список чисел, то для его создания удобно использовать функцию **range**, которая имеет три формы:

* range(end): создается набор чисел от 0 до числа end
* range(start, end): создается набор чисел от числа start до числа end
* range(start, end, step): создается набор чисел от числа start до числа end с шагом step

|  |
| --- |
| numbers = list(range(10))  print(numbers)      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(2, 10))  print(numbers)      # [2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(10, 2, -2))  print(numbers)      # [10, 8, 6, 4] |

Например, следующие два определения списка будут аналогичны, но за счет функции range мы сокращаем объем кода:

|  |
| --- |
| numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(range(1, 10)) |

Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных:

|  |
| --- |
| objects = [1, 2.6, "Hello", True] |

**Перебор элементов**

Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла for:

|  |
| --- |
| companies = ["Microsoft", "Google", "Oracle", "Apple"]  for item in companies:      print(item) |

Здесь вместо функции range мы сразу можем подставить имеющийся список companies.

Перебор с помощью цикла while:

|  |
| --- |
| companies = ["Microsoft", "Google", "Oracle", "Apple"]  i = 0  while i < len(companies):      print(companies[i])      i += 1 |

Для перебора с помощью функции **len()** получаем длину списка. С помощью счетчика i выводит по элементу, пока значение счетчика не станет равно длине списка.

**Сравнение списков**

Два списка считаются равными, если они содержат один и тот же набор элементов:

|  |
| --- |
| numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(range(1,10))  if numbers == numbers2:      print("numbers equal to numbers2")  else:      print("numbers is not equal to numbers2") |

В данном случае оба списка будут равны.

**Методы и функции по работе со списками**

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

* **append(item)**: добавляет элемент item в конец списка
* **insert(index, item)**: добавляет элемент item в список по индексу index
* **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
* **clear()**: удаление всех элементов из списка
* **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
* **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
* **count(item)**: возвращает количество вхождений элемента item в список
* **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
* **reverse()**: расставляет все элементы в списке в обратном порядке

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

* **len(list)**: возвращает длину списка
* **sorted(list, [key])**: возвращает отсортированный список
* **min(list)**: возвращает наименьший элемент списка
* **max(list)**: возвращает наибольший элемент списка

**Добавление и удаление элементов**

Для добавления элемента применяются методы append() и insert, а для удаления - методы remove(), pop() и clear().

Использование методов:

|  |
| --- |
| users = ["Tom", "Bob"]    # добавляем в конец списка  users.append("Alice")  # ["Tom", "Bob", "Alice"]  # добавляем на вторую позицию  users.insert(1, "Bill")          # ["Tom", "Bill", "Bob", "Alice"]    # получаем индекс элемента  i = users.index("Tom")  # удаляем по этому индексу  removed\_item = users.pop(i)            # ["Bill", "Bob", "Alice"]    last\_user = users[-1]  # удаляем последний элемент  users.remove(last\_user)           # ["Bill", "Bob"]    print(users)    # удаляем все элементы  users.clear() |

**Проверка наличия элемента**

Если определенный элемент не найден, то методы remove и index генерируют исключение. Чтобы избежать подобной ситуации, перед операцией с элементом можно проверять его наличие с помощью ключевого слова **in**:

|  |
| --- |
| companies = ["Microsoft", "Google", "Oracle", "Apple"]  item = "Oracle"  # элемент для удаления  if item in companies:      companies.remove(item)    print(companies) |

Выражение item in companies возвращает True, если элемент item имеется в списке companies. Поэтому конструкция if item in companies может выполнить последующий блок инструкций в зависимости от наличия элемента в списке.

**Подсчет вхождений**

Если необходимо узнать, сколько раз в списке присутствует тот или иной элемент, то можно применить метод count():

|  |
| --- |
| users = ["Tom", "Bob", "Alice", "Tom", "Bill", "Tom"]    users\_count = users.count("Tom")  print(users\_count)      # 3 |

**Сортировка**

Для сортировки по возрастанию применяется метод **sort()**:

|  |
| --- |
| users = ["Tom", "Bob", "Alice", "Sam", "Bill"]    users.sort()  print(users)      # ["Alice", "Bill", "Bob", "Sam", "Tom"] |

Если необходимо отсортировать данные в обратном порядке, то мы можем после сортировки применить метод reverse():

|  |
| --- |
| users = ["Tom", "Bob", "Alice", "Sam", "Bill"]    users.sort()  users.reverse()  print(users)      # ["Tom", "Sam", "Bob", "Bill", "Alice"] |

При сортировке фактически сравниваются два объекта, и который из них "меньше", ставится перед тем, который "больше". Понятия "больше" и "меньше" довольно условны. И если для чисел все просто - числа расставляются в порядке возрастания, то для строк и других объектов ситуация сложнее. В частности, строки оцениваются по первым символам. Если первые символы равны, оцениваются вторые символы и так далее. При чем цифровой символ считается "меньше", чем алфавитный заглавный символ, а заглавный символ считается меньше, чем строчный.

Таким образом, если в списке сочетаются строки с верхним и нижним регистром, то мы можем получить не совсем корректные результаты, так как для нас строка "bob" должна стоять до строки "Tom". И чтобы изменить стандартное поведение сортировки, мы можем передать в метод sort() в качестве параметра функцию:

|  |
| --- |
| users = ["Tom", "bob", "alice", "Sam", "Bill"]    users.sort(key=str.lower)  print(users)      # ["alice", "Bill", "bob", "Sam", "Tom"] |

Кроме метода sort мы можем использовать встроенную функцию **sorted**, которая имеет две формы:

* sorted(list): сортирует список list
* sorted(list, key): сортирует список list, применяя к элементам функцию key

|  |
| --- |
| users = ["Tom", "bob", "alice", "Sam", "Bill"]    sorted\_users = sorted(users, key=str.lower)  print(sorted\_users)      # ["alice", "Bill", "bob", "Sam", "Tom"] |

При использовании этой функции следует учитывать, что эта функция не изменяет сортируемый список, а все отсортированные элементы она помещает в новый список, который возвращается в качестве результата.

**Минимальное и максимальное значения**

Встроенный функции Python min() и max() позволяют найти минимальное и максимальное значения соответственно:

|  |
| --- |
| numbers = [9, 21, 12, 1, 3, 15, 18]  print(min(numbers))     # 1  print(max(numbers))     # 21 |

**Копирование списков**

При копировании списков следует учитывать, что списки представляют изменяемый (mutable) тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную:

|  |
| --- |
| users1 = ["Tom", "Bob", "Alice"]  users2 = users1  users2.append("Sam")  # users1 и users2 указывают на один и тот же список  print(users1)   # ["Tom", "Bob", "Alice", "Sam"]  print(users2)   # ["Tom", "Bob", "Alice", "Sam"] |

Это так называемое "поверхностное копирование" (shallow copy). И, как правило, такое поведение нежелательное. И чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить глубокое копирование (deep copy). Для этого можно использовать метод **deepcopy()**, который определен во встроенном модуле **copy**:

|  |
| --- |
| import copy    users1 = ["Tom", "Bob", "Alice"]  users2 = copy.deepcopy(users1)  users2.append("Sam")  # пееменные users1 и users2 указывают на разные списки  print(users1)   # ["Tom", "Bob", "Alice"]  print(users2)   # ["Tom", "Bob", "Alice", "Sam"] |

**Копирование части списка**

Если необходимо скопировать не весь список, а только его какую-то определенную часть, то мы можем применять специальный синтаксис. который может принимать следующие формы:

* list[:end]: через параметр end передается индекс элемента, до которого нужно копировать список
* list[start:end]: параметр start указывает на индекс элемента, начиная с которого надо скопировать элементы
* list[start:end:step]: параметр step указывает на шаг, через который будут копироваться элементы из списка. По умолчанию этот параметр равен 1.

|  |
| --- |
| users = ["Tom", "Bob", "Alice", "Sam", "Tim", "Bill"]    slice\_users1 = users[:3]   # с 0 по 3  print(slice\_users1)   # ["Tom", "Bob", "Alice"]    slice\_users2 = users[1:3]   # с 1 по 3  print(slice\_users2)   # ["Bob", "Alice"]    slice\_users3 = users[1:6:2]   # с 1 по 6 с шагом 2  print(slice\_users3)   # ["Bob", "Sam", "Bill"] |

**Соединение списков**

Для объединения списков применяется операция сложения (+):

|  |
| --- |
| users1 = ["Tom", "Bob", "Alice"]  users2 = ["Tom", "Sam", "Tim", "Bill"]  users3 = users1 + users2  print(users3)   # ["Tom", "Bob", "Alice", "Tom", "Sam", "Tim", "Bill"] |

**Списки списков**

Списки кроме стандартных данных типа строк, чисел, также могут содержать другие списки. Подобные списки можно ассоциировать с таблицами, где вложенные списки выполняют роль строк. Например:

|  |
| --- |
| users = [      ["Tom", 29],      ["Alice", 33],      ["Bob", 27]  ]    print(users[0])         # ["Tom", 29]  print(users[0][0])      # Tom  print(users[0][1])      # 29 |

Чтобы обратиться к элементу вложенного списка, необходимо использовать пару индексов: users[0][1] - обращение ко второму элементу первого вложенного списка.

Добавление, удаление и исменение общего списка, а также вложенных списков аналогично тому, как это делается с обычными (одномерными) списками:

|  |
| --- |
| users = [      ["Tom", 29],      ["Alice", 33],      ["Bob", 27]  ]    # создание вложенного списка  user = list()  user.append("Bill")  user.append(41)  # добавление вложенного списка  users.append(user)    print(users[-1])         # ["Bill", 41]    # добавление во вложенный список  users[-1].append("+79876543210")    print(users[-1])         # ["Bill", 41, "+79876543210"]    # удаление последнего элемента из вложенного списка  users[-1].pop()  print(users[-1])         # ["Bill", 41]    # удаление всего последнего вложенного списка  users.pop(-1)    # изменение первого элемента  users[0] = ["Sam", 18]  print(users)            # [ ["Sam", 18], ["Alice", 33], ["Bob", 27]] |

Перебор вложенных списков:

|  |
| --- |
| users = [      ["Tom", 29],      ["Alice", 33],      ["Bob", 27]  ]    for user in users:      for item in user:          print(item, end=" | ") |

Консольный вывод:

Tom | 29 | Alice | 33 | Bob | 27 |

Кортежи

Кортеж (tuple) представляет последовательность элементов, которая во многом похожа на список за тем исключением, что кортеж является неизменяемым (immutable) типом. Поэтому мы не можем добавлять или удалять элементы в кортеже, изменять его.

Для создания кортежа используются круглые скобки, в которые помещаются его значения, разделенные запятыми:

|  |
| --- |
| user = ("Tom", 23)  print(user) |

Также для определения кортежа мы можем просто перечислить значения через запятую без применения скобок:

|  |
| --- |
| user = "Tom", 23  print(user) |

Если вдруг кортеж состоит из одного элемента, то после единственного элемента кортежа необходимо поставить запятую:

|  |
| --- |
| user = ("Tom",) |

Для создания кортежа из списка можно передать список в функцию **tuple()**, которая возвратит кортеж:

|  |
| --- |
| users\_list = ["Tom", "Bob", "Kate"]  users\_tuple = tuple(users\_list)  print(users\_tuple)      # ("Tom", "Bob", "Kate") |

Обращение к элементам в кортеже происходит также, как и в списке по индексу. Индексация начинается также с нуля при получении элементов с начала списка и с -1 при получении элементов с конца списка:

|  |
| --- |
| users = ("Tom", "Bob", "Sam", "Kate")  print(users[0])     # Tom  print(users[2])     # Sam  print(users[-1])     # Kate    # получим часть кортежа со 2 элемента по 4  print(users[1:4])       # ("Bob", "Sam", "Kate") |

Но так как кортеж - неизменяемый тип (immutable), то мы не сможем изменить его элементы. То есть следующая запись работать не будет:

|  |
| --- |
| users[1] = "Tim" |

При необходимости мы можем разложить кортеж на отдельные переменные:

|  |
| --- |
| user = ("Tom", 22, False)  name, age, isMarried = user  print(name)             # Tom  print(age)              # 22  print(isMarried)        # False |

Особенно удобно использовать кортежи, когда необходимо возвратить из функции сразу несколько значений. Когда функция возвращает несколько значений, фактически она возвращает в кортеж:

|  |
| --- |
| def get\_user():      name = "Tom"      age = 22      is\_married = False      return name, age, is\_married      user = get\_user()  print(user[0])              # Tom  print(user[1])              # 22  print(user[2])              # False |

С помощью встроенной функции **len()** можно получить длину кортежа:

|  |
| --- |
| user = ("Tom", 22, False)  print(len(user))        # 3 |

**Перебор кортежей**

Для перебора кортежа можно использовать стандартные циклы **for** и **while**. С помощью цикла for:

|  |
| --- |
| user = ("Tom", 22, False)  for item in user:      print(item) |

С помощью цикла while:

|  |
| --- |
| user = ("Tom", 22, False)    i = 0  while i < len(user):      print(user[i])      i += 1 |

Как для списка с помощью выражения элемент in кортеж можно проверить наличие элемента в кортеже:

|  |
| --- |
| user = ("Tom", 22, False)  name = "Tom"  if name in user:      print("Пользователя зовут Tom")  else:      print("Пользователь имеет другое имя") |

**Сложные кортежи**

Один кортеж может содержать другие кортежи в виде элементов. Например:

|  |
| --- |
| countries = (      ("Germany", 80.2, (("Berlin",3.326), ("Hamburg", 1.718))),      ("France", 66, (("Paris", 2.2),("Marsel", 1.6)))  )    for country in countries:      countryName, countryPopulation, cities = country      print("\nCountry: {}  population: {}".format(countryName, countryPopulation))      for city in cities:          cityName, cityPopulation = city          print("City: {}  population: {}".format(cityName, cityPopulation)) |

Здесь кортеж countries, который представляет страны, состоит из кортежей, каждый из которых - отдельная страна.

Вложенные кортежи имеют три элемента: название страны, численность ее населения и города. Города представляют отдельный кортеж, где каждый отдельный город - это вложенный кортеж, содержащий название города и численность его населения.