Функции.

**Метод** — это функция, которая применяется к определенному объекту, используя символ точку:

объект.метод()

Посмотрим как происходит добавление элементов в массив:

# допустим, у нас есть список, содержащий первые 4 буквы латинского алфавита

letters = ['a', 'b', 'c', 'd']

# с помощью метода append() мы добавляем еще один элемент в список

letters.append('e')

**print**(letters)

# ['a', 'b', 'c', 'd', 'e']

Как же получить последний элемент списка, если его точная длина заранее неизвестна? Существует два способа. Первый — использование длины списка. Её можно узнать с помощью встроенной функции len(), которая возвращает длину любого итерируемого объекта. К ним относят строки, списки, кортежи и объекты некоторых других типов данных.

С ними подробнее познакомимся в следующих модулях, а пока что можно получать радость от использования такой функции:

**print**(len(letters))

# 5

Как и ожидалось, длина списка равна 5. Тогда доступ к последнему элементу можно получить, если уменьшить эту длину на 1:

**print**(letters[len(letters)-**1**])

# e

letters.append('f') # добавляем еще одну букву

letters.append('g') # и еще одну

**print**(letters[len(letters)-**1**])

# g

Изменение структуры списка может происходить не только путем ее увеличения (добавления новых объектов), но и удаления уже существующих. Для этого можно использовать метод **pop()**:

**print**(letters)

# ['a', 'b', 'c', 'd', 'e', 'f', 'g']

letters.pop() # вызов метода без аргументов удаляет последний элемент списка

**print**(letters)

# ['a', 'b', 'c', 'd', 'e', 'f']

# был удален последний элемент

letters.pop(**0**) # или можно удалить элемент по его индексу

**print**(letters)

# ['b', 'c', 'd', 'e', 'f']

# был удален нулевой элемент

letters.pop(**3**) # и не обязательно удалять из начала или конца списка

**print**(letters)

# ['b', 'c', 'd', 'f']

# был удален элемент с индексом 3

С помощью **срезов** можно получать сразу несколько элементов списка.

|  |  |  |
| --- | --- | --- |
| **Срез** | **Как работает?** | **Пример** |
| [:] | Возвращает элементы полностью | [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’] |
| [2:] | Возвращает элементы списка, начиная с элемента индекса 2 и до конца списка | [‘c’, ‘d’, ‘e’, ‘f’, ‘g’] |
| [:3] | Возвращает элементы списка от его начала до элемента с индексом 3, **не включая** его | [‘a’, ‘b’, ‘c’] |
| [1:4] | Объединяя предыдущие два способа можно получить элементы из середины. В данном случае начиная с индекса 1 до индекса 4, не включительно. Иными словами, элементы с индексами 1,2 и 3 | [‘b’, ‘c’, ‘d’] |
| [::2] | Задает шаг, через который извлекаются элементы | [‘a’, ‘c’, ‘e’, ‘g’] |
| [::-1] | Используя отрицательный шаг, можно развернуть массив | [‘g’, ‘f’, ‘e’, ‘d’, ‘c’, ‘b’, ‘a’] |

L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# ["б", "в", 1]

Подсказка

  неверно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# ["а", 1, 4]

Подсказка

  верно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# [1, "в", "б", "а"]

Подсказка

  верно



L = ["а", "б", "в", **1**, **2**, **3**, **4**]

**print** (L[ ??? ])

# [4, 3, 2]

Подсказка

  верно



Возможности языка позволяют выполнить определенные действия для каждого элемента списка. Такую операцию можно проделать с помощью функцию **map()**:

map(function, list)

Первый аргумент map() — функция, которую нужно применить к каждому элементу списка, а сам список — второй аргумент. Возвращаемое значение этой функции — объект map, который можно преобразовать, например, обратно в список.

Рассмотрим пример:

# имеем список с числами с плавающей точкой

L = [**3.3**, **4.4**, **5.5**, **6.6**]

# печатаем сам объект map

**print**(map(round, L)) # к каждому элементу применяем функцию округления

# <map object at 0x7fd7e86eb6a0>

# и результат его преобразования в список

**print**(list(map(round, L)))

# [3, 4, 6, 7]

L = ['3.3', '4.4', '5.5', '6.6']

**print** (list (map ( ??? , L)))

  неверно



Однако, пользуясь функциями split() и map() можно выполнить нужное преобразование:

string = input("Введите числа через пробел:")

list\_of\_strings = string.split() # список строковых представлений чисел

list\_of\_numbers = list(map(int, list\_of\_strings)) # cписок чисел

**print**(sum(list\_of\_numbers[::**3**])) # sum() вычисляет сумму элементов списка

*Напишите программу, которая на вход получает последовательность чисел, а выводит модифицированный список:*

* 1. *Первое и последнее числа последовательности должны поменяться местами.*
  2. *В конец списка нужно добавить сумму всех чисел.*

*Посмотреть ответ для самопроверки*

*# все операции - деление строки по пробелам, преобразование к числам*

*# и приведение объекта map к типу список, можно делать в одной строке*

*L = list(map(float, input().split()))*

*# обмениваем первое и последнее число*

*# с помощью множественного присваивания*

*L[****0****], L[-****1****] = L[-****1****], L[****0****]*

*# находим сумму и добавляем ее в конец списка*

*L.append(sum(L))*

***print****(L)*

*Задание 2.5.7*

*1 point possible (graded)*

*Чему будет равен последний элемент списка, полученного в результате работы алгоритма из последней задачи, если на вход подается последовательность чисел:*

***1******1******2******3******5******8******13******21******34******55***

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

***Словари***

*Использование списков открывает много возможностей, но и имеет свои ограничения. Каждому элементу списка присваивается целочисленный индекс, по которому можно обращаться к нему, модифицировать и даже удалять. С другой стороны, использование целочисленного индекса не является всегда удобным.*

*Например, если мы хотим в одной переменной хранить информацию о человеке (имя, фамилию, электронную почту, номер телефона, почтовый адрес), то нумерация является неудобной и малоинформативной. Более того, в таком случае не принципиален и порядок этих данных — главное, чтобы к ним можно было удобно обратиться.*

*Для этой цели в Python предусмотрены****словари****(dict) — упорядоченные наборы объектов, доступных по ключу. Иными словами, словарь — это совокупность пар ключ-объект.*

*Сам объект, который хранится в словаре, может быть любым. Даже другим словарём. Но на ключи есть важное ограничение: ключ может быть только объектом неизменяемых типов данных, т.е. числом, строкой или кортежем.*

*Как и в случае списков, словарь можно создать пустым, можно сразу наполнить его объектами, а можно расширять постепенно:*

*person = {} # с помощью фигурных скобок можно создать словарь*

*# словарь заполняется по принципу - ключ:объект (через двоеточие)*

*person = {'name' : 'Ivan Petrov'}*

*# в него можно также добавлять новые объекты по ключу*

*person['age'] =* ***25***

*person['email'] = 'ivan\_petrov@example.com'*

*person['phone'] = '8(800)555-35-35'*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}*

*Попытка извлечения объекта по несуществующему ключу приведет к ошибке:*

***print****(person['address'])*

*# KeyError: 'address'*

*Можно отдельно получить список ключей:*

***print****(person.keys())*

*# dict\_keys(['name', 'age', 'email', 'phone'])*

*Или список значений:*

***print****(person.values())*

*# dict\_values(['Ivan Petrov', 25, 'ivan\_petrov@example.com', '8(800)555-35-35'])*

*Из словаря аналогично спискам можно удалить объект по его ключу. Словарь является упорядоченным. В функцию pop() всегда нужно передавать ключ удаляемого объекта:*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}*

*person.pop('phone')*

***print****(person)*

*# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com'}*

*Задание 2.5.8*

*1 point possible (graded)*

*Какой из перечисленных объектов****не****может быть ключом в словаре?*

*42*



*['id', 42]*



*'id'*



*('id', 42)*



*нет ответа*

*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.9*

*1 point possible (graded)*

*Вместо знаков ??? вставьте название функции, которая удаляет объект из словаря по его ключу.*

*d.??? ('key')*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.10*

*1 point possible (graded)*

*Что выведет программа? Впишите получившуюся строку****без использования кавычек****.*

*d = {'day' :* ***22****, 'month' :* ***6****, 'year' :* ***2015****}*

***print****("||".join(d.keys()))*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.11 (Внешний источник)*

*Посмотреть ответ для самопроверки*

***Практический пример***

*Использование списков и словарей по-отдельности обеспечивает удобный и эффективный способ хранения данных, а их совместное использование открывает еще больше возможностей.*

*Рассмотрим, как может храниться база данных абитуриентов, поступающих в университет. Информация о каждом абитуриенте может храниться в виде словаря с ключами ФИО, Количество баллов, Заявление (о согласии на зачисление):*

*abit1 = {"ФИО" : 'Фадеев О.Е.', "Количество баллов" :* ***283****, "Заявление" : True}*

*abit2 = {"ФИО" : 'Дружинин И.Я.', "Количество баллов" :* ***278****, "Заявление" : False}*

*abit3 = {"ФИО" : 'Афанасьев Д.Н.', "Количество баллов" :* ***276****, "Заявление" : True}*

*abits = [abit1, abit2, abit3]*

***print****(abits)*

*# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}]*

*Этот список, по мере поступления документов, можно пополнять:*

*abit4 = {"ФИО" : 'Любимчиков А.Я.', "Количество баллов" :* ***269****, "Заявление" : True}*

*abits.append(abit4)*

***print****(abits)*

*# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}, {'ФИО': 'Любимчиков А.Я.', 'Количество баллов': 269, 'Заявление': True}]*

*Использование списка словарей с одним и тем же набором ключей позволяет обрабатывать все эти данные целиком, например, ранжировать по количеству баллов или фильтровать по наличию заявления. В следующих модулях мы научимся выполнять такие операции.*

***Уникальные элементы списка***

*Представим, что у нас есть список номеров и абонентов мобильного оператора, которым принадлежат эти номера. Всегда найдутся люди, на которых зарегистрировано несколько номеров. В таком случае в списке номеров-абонентов данные клиентов будут повторяться. Однако, например, для анализа клиентской базы может понадобиться выделить из общего списка только уникальные данные (скажем, уникальные фамилии).*

*Можно подойти к этой задаче с помощью циклов, пробегая по каждому элементу исходного списка и добавляя в новый только уникальные элементы. Уже звучит так, что для, казалось бы, такой простой операции нужно написать много кода. Не правда ли? Это правда. Руководствуясь философией Python — простое лучше, чем сложное, — хотелось бы иметь более простой способ это сделать. И он есть!*

*Для решения практических задач такого рода в Python есть ещё один изменяемый тип данных, который мы упоминали, но подробно не разбирали —****множества****(set).*

***Множество****— это неупорядоченный набор уникальных элементов. Иными словами, во множествах не могут повторяться элементы, а хранятся они в памяти компьютера в произвольном порядке.*

*Создать множество можно несколькими способами:*

*a = {'a', 'b', 'c', 'd'} # используя синтаксис { }*

*Или, что нам будет более полезно, множество можно создать из списка с помощью приведения типов:*

*L = [****1****,****1****,****2****,****3****,****2****]*

*b = set(L)*

***print****(b)*

*# {1,2,3}*

*В начале мы имели список из 5 элементов, два из которых встречались дважды. «Обернув» исходный список в множество, мы получили только уникальные элементы! И не потребовалось писать много строк кода, чтобы это сделать. Осталось только множество вернуть обратно в списковое представление, опять же используя явное приведение типов.*

*b\_list = list(b)*

***print****(b\_list)*

*# [1,2,3]*

*А для краткости все эти операции можно записать в одну строку, ведь в Python естественным образом заложена лаконичность кода.*

*c = list(set(L))*

***print****(c)*

*# [1,2,3]*

*Задание 2.5.12*

*Задание на самопроверку.*

*Напишите программу, которая на вход принимает текст и выводит количество уникальных символов.*

*Решение*

*2.5.13*

*1 point possible (graded)*

*Используя алгоритм из прошлой задачи, найдите количество уникальных символов в тексте. Скопируйте его к себе в консоль целиком.*

*The Zen of PythonBeautiful is better than ugly.Explicit is better than implicit.Simple is better than complex.Complex is better than complicated.Flat is better than nested.Sparse is better than dense.Readability counts.Special cases aren't special enough to break the rules.Although practicality beats purity.Errors should never pass silently.Unless explicitly silenced.In the face of ambiguity, refuse the temptation to guess.There should be one-- and preferably only one --obvious way to do it.Although that way may not be obvious at first unless you're Dutch.Now is better than never.Although never is often better than \*right\* now.If the implementation is hard to explain, it's a bad idea.If the implementation is easy to explain, it may be a good idea.Namespaces are one honking great idea -- let's do more of those!*

*нет ответа*



*Loading*

*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Множества в Python аналогичны математическим множествам, поэтому для них существует несколько собственных операций.*

|  |  |  |
| --- | --- | --- |
| ***Операция*** | ***Название*** | ***Смысл*** |
| *set.union(other)* | *Объединение* | *Возвращает множество, состоящее из элементов set и other.* |
| *set.intersection(other)* | *Пересечение* | *Возвращает множество, состоящее из элементов, которые встречаются и в set, и в other.* |
| *set.difference(other)* | *Разность* | *Возвращает множество элементов set, которые не встречаются в other.* |
| *set.symmetric\_difference(other)* | *Симметричная разность* | *Возвращает множество, включающее все элементы исходных множеств, которые не принадлежат обоим одновременно.* |

*Чтобы не ходить далеко за примером, вернёмся к базе абонентов мобильного оператора. Пусть у нас есть множество абонентов (для простоты — фамилии) и множество должников, а мы хотим получить множество абонентов, не имеющих долгов.*

*abons = {"Иванов", "Петров", "Васильев", "Антонов"}*

*debtors = {"Петров", "Антонов"}*

*non\_debtors = abons.difference(debtors)*

***print****(non\_debtors)*

*# {'Васильев', 'Иванов'}*

*Задание 2.5.14*

*1 point possible (graded)*

*Найдите ошибку в коде и перепишите строку с ошибкой полностью. Представленная ниже программа должна находить множество символов, которые встречаются в двух строках одновременно.*

*a = input("Введите первую строку: ")*

*b = input("Введите вторую строку: ")*

*a\_set, b\_set = set(a), set(b) # используем множественное присваивание*

*a\_and\_b = a\_set.union(b\_set)*

***print****(a\_and\_b)*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

*Задание 2.5.15*

*1 point possible (graded)*

*Напишите программу, которая на вход получает две последовательности целых чисел, а возвращает список элементов, встречающихся только в первой или во второй последовательности, но не в двух одновременно. Какую операцию над множествами вы использовали?*

*Как и в предыдущем задании, нужно подобрать подходящую функцию множества. Выглядеть это будет так: set.something(other), где something и есть выбранная вами функция. Именно её нужно ввести в поле ответа.****Обратите внимание:****необходимо указать только название функции, без скобок.*

*нет ответа*



*Отправить*

*В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».*

***Проверьте себя***

*Напишите числа в порядке возрастания через пробел, которые выведет программа из предыдущего задания, если на вход подаются две последовательности чисел:*

***1******2******3******4******5******6******7******8***

***2******4******6******8******10******12***

*Ответ*

*НазадВперёд*

person = {} # с помощью фигурных скобок можно создать словарь

# словарь заполняется по принципу - ключ:объект (через двоеточие)

person = {'name' : 'Ivan Petrov'}

# в него можно также добавлять новые объекты по ключу

person['age'] = **25**

person['email'] = 'ivan\_petrov@example.com'

person['phone'] = '8(800)555-35-35'

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}

**print**(person['address'])

# KeyError: 'address'

**print**(person.keys())

# dict\_keys(['name', 'age', 'email', 'phone'])

**print**(person.values())

# dict\_values(['Ivan Petrov', 25, 'ivan\_petrov@example.com', '8(800)555-35-35'])

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com', 'phone': '8(800)555-35-35'}

person.pop('phone')

**print**(person)

# {'name': 'Ivan Petrov', 'age': 25, 'email': 'ivan\_petrov@example.com'}

Напишите программу, которая получает на вход название книги - title, фамилию автора - author и год выпуска - year.

Полученные данные должны быть преобразованы в словарь book с ключами: title, author, year. Причем year нужно преобраовать в тип int.

title = input("Введите название книги:")  
author = input("Введите фамилию автора:")  
year = int(input("Введите год издания:"))  
  
book = {'title': title,  
 'author': author,  
 'year': year}  
  
print(book)

print("Привет работаем")  
abit1 = {"ФИО" : 'Фадеев О.Е.', "Количество баллов" : 283, "Заявление" : True}  
abit2 = {"ФИО" : 'Дружинин И.Я.', "Количество баллов" : 278, "Заявление" : False}  
abit3 = {"ФИО" : 'Афанасьев Д.Н.', "Количество баллов" : 276, "Заявление" : True}  
  
  
abits = [abit1, abit2, abit3]  
  
print(abits)  
# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}]

Этот список, по мере поступления документов, можно пополнять:

abit4 = {"ФИО" : 'Любимчиков А.Я.', "Количество баллов" : **269**, "Заявление" : True}

abits.append(abit4)

**print**(abits)

# [{'ФИО': 'Фадеев О.Е.', 'Количество баллов': 283, 'Заявление': True}, {'ФИО': 'Дружинин И.Я.', 'Количество баллов': 278, 'Заявление': False}, {'ФИО': 'Афанасьев Д.Н.', 'Количество баллов': 276, 'Заявление': True}, {'ФИО': 'Любимчиков А.Я.', 'Количество баллов': 269, 'Заявление': True}]

**множества** (set).

**Множество** — это неупорядоченный набор уникальных элементов. Иными словами, во множествах не могут повторяться элементы, а хранятся они в памяти компьютера в произвольном порядке.

a = {'a', 'b', 'c', 'd'} # используя синтаксис { }

множество можно создать из списка с помощью приведения типов:

L = [**1**,**1**,**2**,**3**,**2**]

b = set(L)

**print**(b)

# {1,2,3}

Осталось только множество вернуть обратно в списковое представление, опять же используя явное приведение типов.

b\_list = list(b)

**print**(b\_list)

# [1,2,3]

А для краткости все эти операции можно записать в одну строку, ведь в Python естественным образом заложена лаконичность кода.

c = list(set(L))

**print**(c)

# [1,2,3]

Напишите программу, которая на вход принимает текст и выводит количество уникальных символов.

text = input("Введите текст:")

unique = list(set(text))

**print**("Количество уникальных символов: ", len(unique))

text = "The Zen of PythonBeautiful is better than ugly.Explicit is better than implicit.Simple is better than complex.Complex is better than complicated.Flat is better than nested.Sparse is better than dense.Readability counts.Special cases aren't special enough to break the rules.Although practicality beats purity.Errors should never pass silently.Unless explicitly silenced.In the face of ambiguity, refuse the temptation to guess.There should be one-- and preferably only one --obvious way to do it.Although that way may not be obvious at first unless you're Dutch.Now is better than never.Although never is often better than \*right\* now.If the implementation is hard to explain, it's a bad idea.If the implementation is easy to explain, it may be a good idea.Namespaces are one honking great idea -- let's do more of those!"  
#input("Введите текст:")  
print(text)  
unique = list(set(text))  
  
print("Количество уникальных символов: ", len(unique))

Множества в *Python* аналогичны математическим множествам, поэтому для них существует несколько собственных операций.

|  |  |  |
| --- | --- | --- |
| **Операция** | **Название** | **Смысл** |
| set.union(other) | Объединение | Возвращает множество, состоящее из элементов set и other. |
| set.intersection(other) | Пересечение | Возвращает множество, состоящее из элементов, которые встречаются и в set, и в other. |
| set.difference(other) | Разность | Возвращает множество элементов set, которые не встречаются в other. |
| set.symmetric\_difference(other) | Симметричная разность | Возвращает множество, включающее все элементы исходных множеств, которые не принадлежат обоим одновременно. |

Чтобы не ходить далеко за примером, вернёмся к базе абонентов мобильного оператора. Пусть у нас есть множество абонентов (для простоты — фамилии) и множество должников, а мы хотим получить множество абонентов, не имеющих долгов.

abons = {"Иванов", "Петров", "Васильев", "Антонов"}

debtors = {"Петров", "Антонов"}

non\_debtors = abons.difference(debtors)

**print**(non\_debtors)

# {'Васильев', 'Иванов'}

--

Найдите ошибку в коде и перепишите строку с ошибкой полностью. Представленная ниже программа должна находить множество символов, которые встречаются в двух строках одновременно.

a = input("Введите первую строку: ")

b = input("Введите вторую строку: ")

a\_set, b\_set = set(a), set(b) # используем множественное присваивание

a\_and\_b = a\_set.union(b\_set)

**print**(a\_and\_b)

#a = str(123456789)  
#a = list(str(123456789))  
#print(type(a))  
#for e in str(123456789):  
# print(e)  
#r = map(int.list(a))  
print('3' in str(123456789) and '0' in str(123456789))  
print('5' in str(123456789))  
# True

N = **123**

N\_str = str(N) # преобразуем число в строку

first\_digit = int(N\_str[**0**]) # берём самый левый элемент строки и преобразуем его в число

print(first\_digit % **2** == **0**) # выводим True в случае, если цифра делится на 2, иначе False

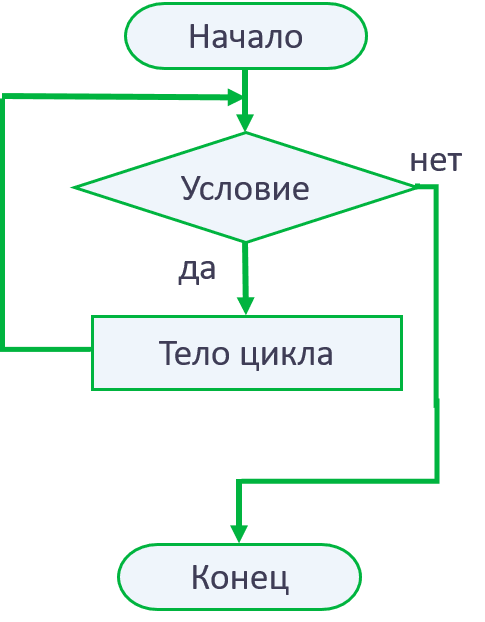
# False



С циклом **while** ситуация похожая: он также проверяет условие и затем начинает выполнение команды в теле цикла. Только команда выполняется не один раз, а **на повторе, пока условие истинно**. Как только условие становится ложным, цикл прерывается. Цикл **while** мы можем перевести на родной язык как: «Пока верно условие, делай…» .

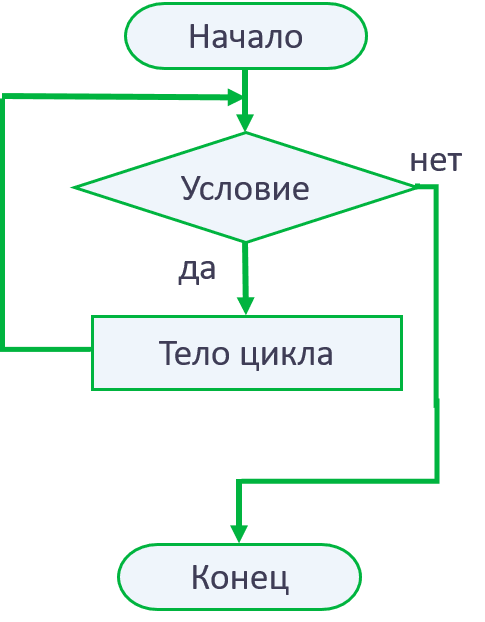
С циклом **while** ситуация похожая: он также проверяет условие и затем начинает выполнение команды в теле цикла. Только команда выполняется не один раз, а **на повторе, пока условие истинно**. Как только условие становится ложным, цикл прерывается. Цикл **while** мы можем перевести на родной язык как: «Пока верно условие, делай…» .

*Блок-схема цикла while*



Цикл **while** называют циклом с **пред**условием, потому что условие проверяется **перед** входом в тело цикла. Давайте разбирать работу цикла **while** на примере.

*Блок-схема цикла while*



Цикл **while** называют циклом с **пред**условием, потому что условие проверяется **перед** входом в тело цикла. Давайте разбирать работу цикла **while** на примере.

S = 0 *# это наша переменная-счетчик, в которой мы будем считать сумму чисел*

n = 1 *# текущее натуральное число, с которого начинаем складывать натуральные числа*

*# заводим цикл while, который будет работать пока сумма не превысит 500*

**while** S < 500: *# делай пока ...*

S += n *# увеличиваем сумму, равносильно S = S + n*

n += 1 *# так как сумма ещё не достигла нужного значения, то увеличиваем переменную счетчик*

print("Ещё считаю ...")

print("Сумма равна: ", S)

print("Количество чисел: ", n-1)

*# хорошо*

n = 1

**while** **True**: *# в данной программе это условие всегда True, цикл будет бесконечным*

print("Hello World")

n += 1

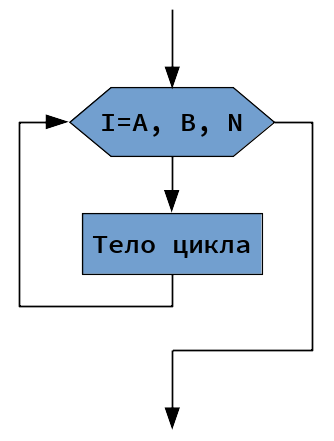
**if** n > 10: *# условие, при достижении которого цикл while будет принудительно завершен*

**break**

|  |  |
| --- | --- |
| C/C++/Java | Python |
| **for** (int i = 0; i < 10; i++):  ...  Здесь цикл будет выполняться, пока переменная **i** будет меньше 10. После этого цикл остановится. | **for** i **in** [1, 2, 3, 4, 5]:  print(i)  Цикл будет работать до тех пор, пока не закончатся все элементы в списке. |

В общем смысле цикл for работает по заранее известному количеству шагов. Вы как бы

*Блок-схема цикла for*



На этом рисунке: I – параметр (счётчик) цикла, A — начальное значение параметра, B — его конечное значение, а N — шаг цикла (приращение счётчика)



Так вот, функция **range** может работать тремя способами:

* **range(END);**
* **range(START, END);**
* **range(START, END, STEP).**

**RANGE(END)**

|  |  |
| --- | --- |
| Пример | Соответствующий диапазон |
| **range(10)** | от 0 до 9 |
| **range(1, 10)** | от 1 до 9 |
| **range(1, 10, 2)** | от 1 до 9 с шагом 2, то есть 1, 3, 5, 7, 9 |

print(list(range(5)))

*# [0, 1, 2, 3, 4]*

**RANGE(START, END)**

Во втором случае вы получите последовательность от **start** до **end-1** также с шагом в 1.

print(list(range(2, 5)))

*# [2, 3, 4]*

**RANGE(START, END, STEP)**

В третьем случае вы можете задать произвольный шаг и получить, например, все нечётные натуральные числа меньше 10:

print(list(range(1, 10, 2)))

*# [1, 3, 5, 7, 9]*

S = 0 *# заводим переменную счетчик, в которой мы будем считать сумму*

N = 5

*# заводим цикл for в котором мы будем проходить по всем числам от одного до N*

**for** i **in** range(1, N + 1): *# равносильно выражению for i in [1, 2, 3, ... , N -1, N]:*

print("Значение суммы на предыдущем шаге: ", S)

print("Текущее число: ", i)

S = S + i *# cуммируем текущее число i и перезаписываем значение суммы*

print("Значение суммы после сложения: ", S)

print("---")

print("Конец цикла")

print()

print("Ответ: сумма равна = ", S)

Перебираем последовательно элементы в нашей матрице:

matrix = [

[1, 2]

,[3, 4]

,[5, 6]

]

**for** row **in** matrix:

**for** element **in** row:

print(element, end = " ")

На вывод будут распечатаны последовательно элементы матрицы.

Если мы хотим, чтобы вывод был в более «табличном» виде, то в конце каждой строки нужно будет вставить перенос строки.

matrix = [

[1, 2]

,[3, 4]

,[5, 6]

]

**for** row **in** matrix:

**for** element **in** row:

print(element, end = " ")

print() *# этот принт относится к циклу for row in matrix:*

**ПРИМЕР**

**Условие задачи. Д**ана двумерная матрица 3x3 (двумерный массив). Необходимо определить максимум и минимум каждой строки, а также их индексы.

random\_matrix = [

[9, 2, 1],

[2, 5, 3],

[4, 8, 5]

]



В каждой строке помечен минимальный элемент. Нам нужно пройтись по всем строкам сверху вниз и по элементам в строке справа налево, определить среди них минимум в каждой строке и записать результат в отдельный список.

Обращаться к элементам матрицы мы можем так ↓

random\_matrix[i][j]

где i — номер строки, j — номер столбца.

Нам нужно найти минимумы в каждой строке и индексы этих минимумов (номера столбцов с минимальным элементом).

Создадим списки, где мы будем хранить ответ на наш вопрос:

min\_value\_rows = []

min\_index\_rows = []

Для начала напишем цикл, в котором пройдём по всем строкам матрицы:

**for** row **in** random\_matrix:

Таким образом, в переменной **row** будут строки матрицы (обычные списки) — в них и выполним поиск минимального и максимального элемента.

Далее реализуем стандартный **алгоритм поиска минимального элемента в списке**.

**КРАТКОЕ ОПИСАНИЕ АЛГОРИТМА:**

1. Создадим новую переменную, в которой будем хранить кандидата на звание «минимальный элемент».

2. Для начала пусть кандидатом будет первый элемент списка.

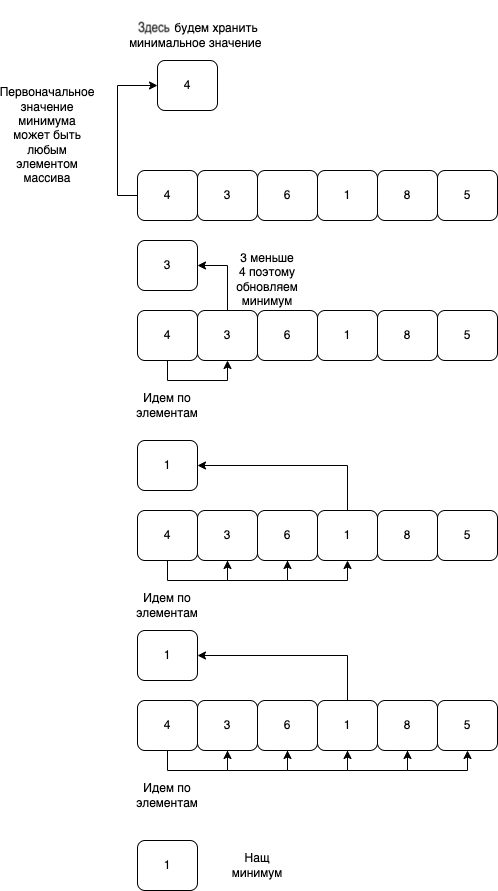
3. В цикле пройдём по всем элементам списка.

4. Сравним этот элемент списка с кандидатом.

5. Если элемент меньше кандидата на звание «минимальный элемент», то наш текущий кандидат не подходит — он не является минимальным.

6. Поэтому теперь кандидатом будет этот самый элемент.

В конце, после прохода по всему списку, наш кандидат будет меньше каждого элемента списка, а значит мы нашли минимум.



✍️ Реализуем алгоритм в коде ↓

Создадим две переменные:

min\_index = 0

min\_value = row[min\_index]

где **min\_index** — индекс кандидата (по алгоритму изначально кандидат — это первый элемент, поэтому индекс 0). **min\_value** — сам кандидат, его значение: по алгоритму это первый элемент, поэтому можно написать **row[0]**, а можно — **row[min\_index]**, потому что в переменной **min\_index** хранится 0.

Теперь в цикле пройдём по элементам строки **row**.

Так как нам требуется ещё и индекс, будем перебирать индексы элементов этой строки. Индексы списка **row** — это **0, 1 , ... , len(row)-1**.

Индексы можно перебрать так:

**for** index\_col **in** range(len(row)):

Теперь сравниваем элемент с индексом **index\_col** и нашего кандидата в соответствии с алгоритмом:

**if** row[index\_col] < min\_value:

Если он оказался меньше, то теперь кандидат — это сам текущий элемент:

min\_value = row[index\_col]

min\_index = index\_col

Поэтому значение кандидата **min\_value** — это сам элемент **row\_[index\_col]**. А индекс кандидата — это индекс элемента.

После прохода по всему списку кандидатом как раз является минимум списка **row**. Поэтому добавим этого кандидата и его индекс в наши списки с ответами:

min\_value\_rows.append(min\_value)

min\_index\_rows.append(min\_index)

Теперь, после прохода в цикле по строкам матрицы, выведем наши ответы в консоль:

print(min\_value\_rows)

print(min\_index\_rows)

Аналогично сделаем для максимума.

**Плоный код.**

random\_matrix = [

[9, 2, 1],

[2, 5, 3],

[4, 8, 5]

]

min\_value\_rows = []

min\_index\_rows = []

max\_value\_rows = []

max\_index\_rows = []

**for** row **in** random\_matrix:

min\_index = 0

min\_value = row[min\_index]

max\_index = 0

max\_value = row[max\_index]

**for** index\_col **in** range(len(row)):

**if** row[index\_col] < min\_value:

min\_value = row[index\_col]

min\_index = index\_col

**if** row[index\_col] > max\_value:

max\_value = row[index\_col]

max\_index = index\_col

min\_value\_rows.append(min\_value)

min\_index\_rows.append(min\_index)

max\_value\_rows.append(max\_value)

max\_index\_rows.append(max\_index)

print("Minimal elements:", min\_value\_rows) *# минимальные элементы*

print("Their indices:", min\_index\_rows) *# их индексы*

print("Maximal elements:", max\_value\_rows) *# максимальные элементы*

print("Their indices:", max\_index\_rows) *# их индексы*

**Test.**

random\_matrix = [  
 [9, 2, 1],  
 [2, 5, 3],  
 [4, 8, 5]  
]  
min\_value\_rows = []  
min\_index\_rows = []  
max\_value\_rows = []  
max\_index\_rows = []  
for row in random\_matrix:  
 min\_index = 0  
 min\_value = row[min\_index]  
 print(1,min\_value)  
 #print(1.1,min\_index)  
 print(2,row)  
 max\_index = 0  
 max\_value = row[max\_index]  
 #print(3,max\_index)  
 print(4,row)  
 for index\_col in range(len(row)):  
 if row[index\_col] < min\_value:  
 min\_value = row[index\_col]  
 min\_index = index\_col  
 if row[index\_col] > max\_value:  
 max\_value = row[index\_col]  
 max\_index = index\_col  
 min\_value\_rows.append(min\_value)  
 min\_index\_rows.append(min\_index)  
 max\_value\_rows.append(max\_value)  
 max\_index\_rows.append(max\_index)  
 print(5,min\_value\_rows)  
 print(6,min\_index\_rows)  
 print(7,max\_value\_rows)  
 print(8,max\_index\_rows)  
print("Minimal elements:", min\_value\_rows) # минимальные элементы  
print("Their indices:", min\_index\_rows) # их индексы  
print("Maximal elements:", max\_value\_rows) # максимальные элементы  
print("Their indices:", max\_index\_rows) # их индексы

**My test/ Макс номер в матрице**

test\_matrix = [  
 [1, 2, 3],  
 [7, -1, 2],  
 [123, 2, -1]  
]  
#random\_matrix = [  
# [9, 2, 1],  
 # [2, 5, 3],  
 # [4, 8, 5]  
#]  
min\_value\_rows = []  
min\_index\_rows = []  
max\_value\_rows = []  
max\_index\_rows = []  
#for row in random\_matrix:  
for row in test\_matrix:  
 min\_index = 0  
 min\_value = row[min\_index]  
 max\_index = 0  
 max\_value = row[max\_index]  
 for index\_col in range(len(row)):  
 if row[index\_col] < min\_value:  
 min\_value = row[index\_col]  
 min\_index = index\_col  
 if row[index\_col] > max\_value:  
 max\_value = row[index\_col]  
 max\_index = index\_col  
 min\_value\_rows.append(min\_value)  
 min\_index\_rows.append(min\_index)  
 max\_value\_rows.append(max\_value)  
 max\_index\_rows.append(max\_index)  
print("Minimal elements:", min\_value\_rows) # минимальные элементы  
print("Their indices:", min\_index\_rows) # их индексы  
print("Maximal elements:", max\_value\_rows) # максимальные элементы  
print("Their indices:", max\_index\_rows) # их индексы  
max\_number = max(max\_value\_rows)  
print("максимальные элемент =", max\_number) # максимальные элемент

Функция Python max() возвращает самый большой элемент итерабельного объекта. Ее также можно использовать для поиска максимального значения между двумя или более параметрами.

list1 = [3, 2, 8, 5, 10, 6]

max\_number = max(list1)

print("Наибольшее число:", max\_number)

list1 = ["Виктор", "Артем", "Роман"]

max\_string = max(list1, key=len)

print("Самая длинная строка:", max\_string)

В этой статье мы научимся находить максимальное значение в списке на Python. Для всестороннего понимания вопроса мы рассмотрим использование некоторых встроенных функций, простые подходы, а также небольшие реализации известных алгоритмов.

Сначала давайте вкратце рассмотрим, что такое список в Python и как найти в нем максимальное значение или просто наибольшее число.

**Список в Python**

В Python есть встроенный тип данных под названием список (list). По своей сути он сильно напоминает массив. Но в отличие от последнего данные внутри списка могут быть любого типа (необязательно одного): он может содержать целые числа, строки или значения с плавающей точкой, или даже другие списки.

Хранимые в списке данные определяются как разделенные запятыми значения, заключенные в квадратные скобки. Списки можно определять, используя любое имя переменной, а затем присваивая ей различные значения в квадратных скобках. Он является упорядоченным, изменяемым и допускает дублирование значений. Например:

КОПИРОВАТЬ

list1 = ["Виктор", "Артем", "Роман"]

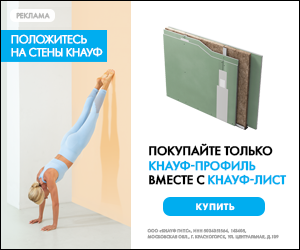
list2 = [16, 78, 32, 67]

list3 = ["яблоко", "манго", 16, "вишня", 3.4]

Далее мы рассмотрим возможные варианты кода на Python, реализующего поиск наибольшего элемента в списке, состоящем из сравниваемых элементов. В наших примерах будут использоваться следующие методы/функции:

1. Встроенная функция max()
2. Метод грубой силы (перебора)
3. Функция reduce()
4. Алгоритм Heap Queue (очередь с приоритетом)
5. Функция sort()
6. Функция sorted()
7. Метод хвостовой рекурсии

**№1 Нахождение максимального значения с помощью функции max()**

[](https://yandex.ru/an/count/WWiejI_zO5K13H00z1jQwoTwoB17T0K0LG8nvWkROm00000uh8sW0K0CI09WZ1Xe173qYjkKyzsIgfa1W07StCe6Y06MqlOIa06Un_Q5ue20W0AO0Px7zeLYi07E-uUm2BW1eAdHzINPsgm1u06wYeMg0UW1gWGHc0B6eAKOe0BQaxG1y0AQwex7181AY0ME8f05eoEm1TO8k0Ny2C05n2h81UWAq0Mxk0O2oGRk2vwxyztR5wa7PW4fOP-DaMUu1v0ou0U6mf211k08xObQw0a7z0E02WWYw0oN0k0DWe20WO20W0YO3kwN-DoxllVj5uWGp_EYdGQHH9WHnRCgeRc84W6G4W7f4gNzOBBNYu2cg1FLgDBMlwAfc1U0523W507e58m2c1QGnEkS1g395l0_q1R2Yjw-0PWNwktABwWN2RWN1C0NjjO1e1cg0x0Pk1c16l__n-anHi1oi1hotyIEmftqxKI270rYMKGwK75jP5HhLsKtg1u1i1-_0iWVtQojLhWWvvCeiY49DZaqC3CnEZata2APY2gG8fk8Af0Yr9uga2BMdYgG8iEpAf0YnRCga2B9iogm8W7L8l__V_-18uaZcfcPcPcPsJyI0F7BIAnYOuHCUJx5m5I6QYt1ywIvY0Wz6OS28R_qJhnMPwTb-GCd-Gopsr8tol2favjmm7mJH8sbQee3XnW5~1?media-test-tag=2251799813689171)

РЕКЛАМА

Это самый простой и понятный подход к поиску наибольшего элемента. Функция Python max() возвращает самый большой элемент итерабельного объекта. Ее также можно использовать для поиска максимального значения между двумя или более параметрами.

В приведенном ниже примере список передается функции max в качестве аргумента.

КОПИРОВАТЬ

list1 = [3, 2, 8, 5, 10, 6]

max\_number = max(list1)

print("Наибольшее число:", max\_number)

Наибольшее число: 10

Если элементы списка являются строками, то сначала они упорядочиваются в алфавитном порядке, а затем возвращается наибольшая строка.

КОПИРОВАТЬ

list1 = ["Виктор", "Артем", "Роман"]

max\_string = max(list1, key=len)

print("Самая длинная строка:", max\_string)

Самая длинная строка: Виктор

**№2 Поиск максимального значения перебором**

Это самая простая реализация, но она немного медленнее, чем функция max(), поскольку мы используем этот алгоритм в цикле.

В примере выше для поиска максимального значения нами была определена функция large(). Она принимает список в качестве единственного аргумента. Для сохранения найденного значения мы используем переменную max\_, которой изначально присваивается первый элемент списка. В цикле for каждый элемент сравнивается с этой переменной. Если он больше max\_, то мы сохраняем значение этого элемента в нашей переменной. После сравнения со всеми членами списка в max\_ гарантировано находится наибольший элемент.

КОПИРОВАТЬ

def large(arr):

max\_ = arr[0]

for ele in arr:

if ele > max\_:

max\_ = ele

return max\_

list1 = [1,4,5,2,6]

result = large(list1)

print(result) # вернется 6

**№3 Нахождение максимального значения с помощью функции reduce()**

В функциональных языках reduce() является важной и очень полезной функцией. В Python 3 функция reduce() перенесена в отдельный модуль стандартной библиотеки под названием functools. Это решение было принято, чтобы поощрить разработчиков использовать циклы, так как они более читабельны. Рассмотрим приведенный ниже пример использования reduce() двумя разными способами.

В этом варианте reduce() принимает два параметра. Первый — ключевое слово max, которое означает поиск максимального числа, а второй аргумент — итерабельный объект.

КОПИРОВАТЬ

from functools import reduce

list1 = [-1, 3, 7, 99, 0]

print(reduce(max, list1)) # вывод: 99

Другое решение показывает интересную конструкцию с использованием лямбда-функции. Функция reduce() принимает в качестве аргумента лямбда-функцию, а та в свою очередь получает на вход условие и список для проверки максимального значения.

КОПИРОВАТЬ

from functools import reduce

list1 = [-1, 3, 7, 99, 0]

print(reduce(lambda x, y: x if x > y else y, list1)) # -> 99

## №4 Поиск максимального значения с помощью приоритетной очереди

Heapq — очень полезный модуль для реализации минимальной очереди. Если быть более точным, он предоставляет реализацию алгоритма очереди с приоритетом на основе кучи, известного как heapq. Важным свойством такой кучи является то, что ее наименьший элемент всегда будет корневым элементом. В приведенном примере мы используем функцию heapq.nlargest() для нахождения максимального значения.

КОПИРОВАТЬ

import heapq

list1 = [-1, 3, 7, 99, 0]

print(heapq.nlargest(1, list1)) # -> [99]

Приведенный выше пример импортирует модуль heapq и принимает на вход список. Функция принимает n=1 в качестве первого аргумента, так как нам нужно найти одно максимальное значение, а вторым аргументом является наш список.

## №5 Нахождение максимального значения с помощью функции sort()

Этот метод использует функцию sort() для поиска наибольшего элемента. Он принимает на вход список значений, затем сортирует его в порядке возрастания и выводит последний элемент списка. Последним элементом в списке является list[-1].

КОПИРОВАТЬ

list1 = [10, 20, 4, 45, 99]

list1.sort()

print("Наибольшее число:", list1[-1])

Наибольшее число: 99

## №6 Нахождение максимального значения с помощью функции sorted()

Этот метод использует функцию sorted() для поиска наибольшего элемента. В качестве входных данных он принимает список значений. Затем функция sorted() сортирует список в порядке возрастания и выводит наибольшее число.

КОПИРОВАТЬ

list1=[1,4,22,41,5,2]

sorted\_list = sorted(list1)

result = sorted\_list[-1]

print(result) # -> 41

## №7 Поиск максимального значения с помощью хвостовой рекурсии

Этот метод не очень удобен, и иногда программисты считают его бесполезным. Данное решение использует рекурсию, и поэтому его довольно сложно быстро понять. Кроме того, такая программа очень медленная и требует много памяти. Это происходит потому, что в отличие от чистых функциональных языков, Python не оптимизирован для хвостовой рекурсии, что приводит к созданию множества стековых фреймов: по одному для каждого вызова функции.

КОПИРОВАТЬ

def find\_max(arr, max\_=None):

if max\_ is None:

max\_ = arr.pop()

current = arr.pop()

if current > max\_:

max\_ = current

if arr:

return find\_max(arr, max\_)

return max\_

list1=[1,2,3,4,2]

result = find\_max(list1)

print(result) # -> 4

## Заключение

В этой статье мы научились находить максимальное значение из заданного списка с помощью нескольких встроенных функций, таких как max(), sort(), reduce(), sorted() и других алгоритмов. Мы написали свои код, чтобы попробовать метод перебора, хвостовой рекурсии и алгоритма приоритетной очереди.

**Test**

a = int(input('Input your lucky number 1:'))  
b = int(input('Input your lucky number 2:'))  
  
if a == 10 and b == 5:  
 print("You re winner")  
else:  
 print('=',a,b)

a = [1, 'Hello', 5]  
for x in a:  
 print(x)

a = [1, 'Hello', 5]  
for x in a:  
 pass  
print(x)

**Тоже**

a = [1, 'Hello', 'world', 5, 6, ['my list', 5]]  
it = iter(a)  
  
try:  
 while True:  
 print(next(it))  
except StopIteration:  
 pass

**Print all**

a = [1, 'Hello', 'world', 5, 6, ['my list', 5]]  
for x in a:  
 if type(x) == list:  
 for l in x:  
 print(l)  
 else:  
 print(x)