МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. В.Ф. УТКИНА

Python. Простые списки и кортежи. Обработка одномерных массивов

Методические указания к лабораторной работе

14

Рязань 2021

УДК 004.432

Python. Простые списки и кортежи. Обработка одномерных массивов: методические указания к лабораторной работе №14 / Рязан. гос. радиотехн. ун-т.; сост.: А.В.Климухина, А.Н.Пылькин, Ю.С.Соколова, Е.С. Щенёв, М.Г. Щетинин. Рязань, 2021. – 24 с.

Рассмотрены сложные типы данных: списки и кортежи, которые позволяют проводить обработку данных, объединенных в одномерные массивы. Приведены сведения, позволяющие составлять алгоритмы и программы обработки одномерных массивов. Приведены основные приемы программирования типовых задач обработки массивов. Рассмотрены функции и методы, используемые при обработке списков.

В качестве практических заданий предлагается провести обработку одномерных массивов.

Предназначены для студентов очной и заочной формы обучения всех направлений подготовок и специальностей.

Ил.5

Простые списки, кортежи, обработка одномерных массивов.

Печатается по решению Научно-методического совета Рязанского государственного радиотехнического университета им. В.Ф. Уткина.

Рецензент: кафедра информатики, информационных технологий и защиты информации ФГБОУ ВО «Липецкий государственный педагогический университет им. П.П. Семенова-Тян-Шанского» (зав. каф., к.т.н., доц. Скуднев Д.М.).

Python. Простые списки и кортежи. Обработка одномерных массивов.

Составители: К л и м у х и н а Анастасия Витальевна

П ы л ь к и н Александр Николаевич

С о к о л о в а Юлия Сергеевна

Щ е н ё в Евгений Сергеевич

Щ е т и н и н Максим Геннадьевич

ПРОСТЫЕ СПИСКИ И КОРТЕЖИ. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

Список (list) в Python является непрерывной динамической конструкцией, которая состоит из одногипных или разнотипных элементов. Количество элементов в списке можно изменять в процессе выполнения программы. С помощью списка в программе на Python представляется массив данных, поэтому вместо названия «список» используют название «массив». Массив также можно представлять в форме кортежа (tuple), который является неизменяемым списком.

Создание списка

Список создается перечислением элементов через запятую в квадратных скобках:

```
s1 = [1, 9, 5, 2]
s2 = ['r', 's', 'r', 'e', 'u']
s3 = ['aa', 'bb', 'ccc', 1, 9]
s4 = []          # пустой список
```

Элементы списка в памяти компьютера хранятся в виде указателей на объекты, Python размещает элементы списка в памяти, а затем размещаются указатели на эти элементы, т.е. список представляет собой массив указателей (см. рисунок 14.1).

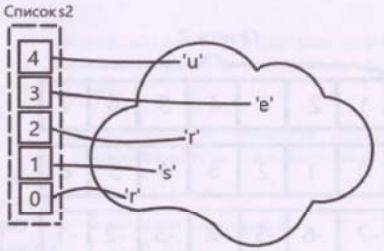


Рис. 14.1. Взаимосвязь значений и указателей списка

Очень просто создается список из одинаковых элементов:

```
u = ['k']*3      # то же самое, что ['k', 'k', 'k']
y = [0]*100      # список из ста нулей
```

Список можно генерировать с помощью использования конструкции простого цикла с заголовком (цикла *for*):

```
x1 = [k for k in range(7)]    #[0,1,2,3,4,5,6]
x2 = [k**2 for k in range(7)]# [0,1,4,9,16,25,36]
x3 = [k for k in range(7) if k % 3]  # [1,2,4,5]
import random
x4 = [random.random() for x in range(7)]
        # список из
        # 7 случайных чисел
```

Обращение к элементам списка

Список (массив) является упорядоченной совокупностью элементов, в которой место элемента определяется индексом. Поэтому доступ к нужному элементу списка осуществляется с помощью индекса. Значение индекса начинается с 0, далее следует 1,2,3 и т.д. Если указывается индекс, который не входит в диапазон, то возникает ошибка *indexError*. Можно использовать отрицательный индекс. В этом случае элементы считаются (упорядочиваются) от конца списка к началу в диапазоне от -1 до -(длина), что показано на рис. 14.2.

Рис. 14.2. Соотношение номера элемента, индекса и отрицательного индекса

Например, обозначения **z**[4] и **z**[-3] соответствуют обращению к одному и тому же элементу. Ниже приведена программа, в которой показаны практические приемы, связанные с обработкой одномерных массивов с использованием типовых средств языка Python. Вся программа разделена на отдельные части, которые отмечены соответствующими комментариями *#1, #2, ..., #11*. Результаты обработки и использования тех или иных средств приведены ниже и также разделены на отдельные части. Например, в первой части *#1* приведен пример обращения к элементу **mas[3]** и альтернативный способ обращения к этому элементу **mas[-2]**.

```
#1                объявление массива
mas = [1, 2, 3, 4]
print('# 1              объявление массива')
print(mas)
print(mas[3])   #вывод элемента массива
print(mas[-2])  #вывод элемента массива
#2                обновление элемента
mas[1] = 12
print('# 2              обновление элемента')
print(mas)      #вывод измененного массива
#3                удаление элемента
del mas[1]
print('# 3              удаление массива')
print(mas)
#4                итерация по списку
print('# 4              итерация по списку')
for x in mas:
    print(x)
#5              вывод массива в обратном порядке
print('# 5          вывод массива в обратном порядке')
for z in reversed(mas):
    print(z)
#6              принадлежность элемента массиву
print('# 6        принадлежность элемента массиву')
print(1 in mas)
print(99 in mas)
#7              объединение массивов
mas1 = [0.5, 0.6, 0.7, 0.8]
massiv = mas + mas1
print('# 7        объединение массивов')
print(massiv)
```

```
#8                определение длины массива
print('# 8          определение длины массива')
print(len(massiv))
#9                нарезка массива
print('# 9          нарезка массива')
mas2 = massiv[1:4]
print(mas2)
print(massiv[:5])
print(massiv[3:])
print(massiv[:4])
# 10               повторение элементов массива
x = [1, 2, 3, 4]
y = x*3
print('# 10         повторение элементов массива')
print(y)
# 11               встроенный конструктор
print('# 11         встроенный конструктор')
print(m)
z = list((1, 9, 5, 2))
print(z)
```

В процессе выполнения программы осуществляется ввод данных, являющихся результатами простейшей обработки одномерных массивов. Все выводимые данные так же, как и программа, разделены на части. Поэтому дополнительные комментарии касаются одноименных частей программы и результатов выполнения этой программы.

Результат выполнения вышеприведенной программы:

```
# 1               объявление массива
[1, 2, 3, 4]
4
3
# 2               обновление элемента
[1, 12, 3, 4]
# 3               удаление массива
[1, 3, 4]
# 4               итерация по списку
1
3
4
# 5         вывод массива в обратном порядке
```



```
4
3
1
# 6         принадлежность элемента массиву
True
False
# 7         объединение массивов
[1, 3, 4, 0.5, 0.6, 0.7, 0.8]
# 8         определение длины массива
7
# 9         нарезка массива
[3, 4, 0.5]
[1, 3, 4, 0.5, 0.6]
[0.5, 0.6, 0.7, 0.8]
[1, 3, 4, 0.5]
# 10        повторение элементов массива
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
# 11        встроенный конструктор
['r', 's', 'r', 'e', 'u']
[1, 9, 5, 2]

Process finished with exit code 0
```

1 Объявление массива.

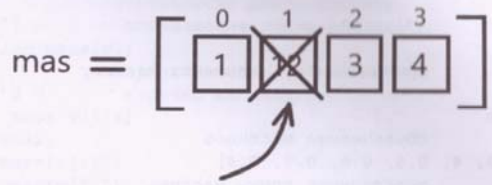
В данной части программы продемонстрирован простейший способ объявления массива (в форме списка) и показаны способы обращения к упорядоченным элементам одномерного массива

2 Обновление элемента.

Значение элемента массива очень просто удается изменить путем использования оператора присвоения.

3 Удаление элемента.

При удалении элемента следует помнить, что нумерация элементов в списке начинается с нуля.



4 Итерация по списку.

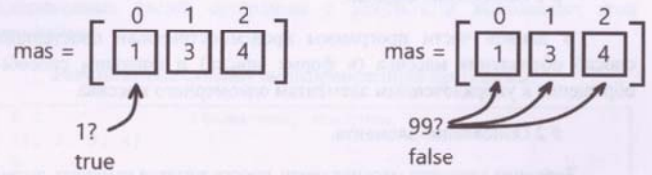
В данном участке продемонстрирован пример использования цикла *for* для перебора элементов массива.

5 Вывод массива в обратном порядке.

В дополнении к # 4 продемонстрировано использование функции *reserved* для просмотра массива в обратном порядке.

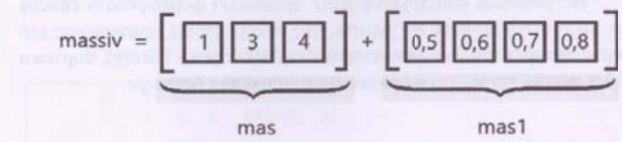
6 Принадлежность элемента массива.

Чтобы проверить наличие элемента в массиве используется функция *in*, которая возвращает значение *true(да)* или *false(нет)*.



7 Объединение массивов.

Объединение массивов (двух или более) осуществляется с помощью оператора конкатенации +.



8 Определение длины массива.

Длина полученного в результате объединения массива *massiv* округляется с помощью функции *len* и равна 7.

9 Нарезка массива.

Нарезка массива позволяет сформировать срез массива, т.е. некоторую последовательность, не изменяя исходный список. Синтаксис среза определяется в общем случае так:

итерируемая_переменная[начало: конец – 1: шаг]

Из приведенных примеров следует, что в простейшем случае требуется указать начальное значение, уменьшенное на единицу (т.к. индекс изменяется с нулевого значения). Если первый индекс начинается с нуля, то его можно опустить. Если конечный индекс равен длине списка, то его также можно опустить. Третий параметр определяет длину шага и используется достаточно редко. Следует иметь в виду, что срез представляет собой последовательность, а не новый список. В результате этого срезы используются в других итерируемых типах данных (например, в списках или кортежах). Данное обстоятельство говорит о том, что срезы можно использовать на участках #4 и #5 обсуждаемой программы.

10 Повторение элементов массива.

Если *x* является списком, то операция *x*3* равносильна конкатенации *x+x+x*, в результате чего формируется соответствующий список.

11 Встроенный конструктор.

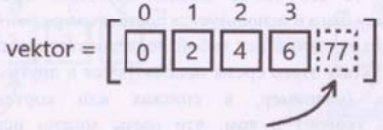
Встроенный конструктор *list()* позволяет формировать список с помощью итеративного аргумента, что обеспечивает применение его в упорядоченных типах данных: список (*list*), строки (*string*), кортежи (*tuple*) и другие типы, что и продемонстрировано в примере.

Методы для работы со списками

Список является конкретным экземпляром класса *list*, для которого в *Python* определены некоторые функции, позволяющие выполнять действия со списками. Метод – функция, которую можно использовать для объектов определенного типа (в данном случае речь идет об объектах класса *list*). Метод определен в пределах класса, а не внутри экземпляра. В *Python* используется широкий набор методов.

Добавление в список

Для добавления элемента в конец списка используется метод *append()*, который указывается совместно с именем списка через точку. Например, пусть объявлен массив *vector*, состоящий из 4-х элементов. Ниже приведена программа, которая использует метод *append()* и добавляет элемент в конец списка *vektor*.



Далее еще два раза применяем метод append() для добавления элементов 8 и 9.

```
# создаем массив из 4-ч элементов
vektor = [0, 2, 4, 6]
# добавляем в конец списка число 77
vektor.append(77)
print(vektor)
vektor.append(8)
vektor.append(9)
```



```
print(vektor)
```

В результате выполнения программы будет выведен результат:

```
[0, 2, 4, 6, 77]
[0, 2, 4, 6, 77, 8, 9]
```

Рассмотренный метод append() может быть использован для реализации алгоритма ввода значений элементов массива с клавиатуры (см. рис. 14.3). Пусть требуется ввести массив, состоящий из 4-х элементов. Первоначально формируется пустой массив, а далее с помощью цикла с заголовком вводят значения массива. Для обеспечения визуализации и принадлежности вводимого значения конкретному элементу организована подсказка.

В программе, реализующей рассматриваемый алгоритм, выполняется добавление в конец списка строки «пять», что демонстрирует динамическое изменение списка. Программа и результаты ее выполнения имеют следующий вид:

```
# ввод элементов массива с клавиатуры
n = 4
a = []
for I in range(n):
  print('a[', I, ']=', sep=' ', end=' ')
  a.append(int(input()))
print(a)
a.append('пять ')
print(a)
```

```
a[0]=1
a[1]=2
a[2]=3
a[3]=4
[1, 2, 3, 4]
[1, 2, 3, 4, 'пять']
```

Замечание. Список [1, 2, 3, 4] можно называть массивом, а список [1, 2, 3, 4, 'пять'] не является массивом, так как по определению массив – упорядоченная совокупность однотипных данных.

Рис. 14.3. Алгоритм ввода массива с клавиатуры

Добавление в список на указанную позицию

Синтаксис метода, осуществляющего добавление в список элемента на указанную позицию

list.insert(i,k), где list – список; x – добавленный элемент.

Для примера в список b на третью позицию (отсчет начинается с 0) добавим элемент 'пять'.

Приведенные ниже программа и результат ее выполнения не требуют дополнительных комментариев.

```
# добавление элемента на указанную позицию
b = [2, 3, 4, 6]
b.insert(3, 'пять')
print(b)
b.insert(5, 'семь')
print(b)
b.insert(0, 'начальный элемент')
print(b)
```

```
[2, 3, 4, 'пять', 6]
[2, 3, 4, 'пять', 6, 'семь']
['начальный элемент',2, 3, 4, 'пять', 6, 'семь']
```

Удаление элемента из списка

Синтаксис вызова данного метода имя_списка.pop(i), где i – номер позиции удаляемого элемента.

Приведем пример и результаты его выполнения, которые достаточно полно иллюстрируют метод pop. Если пример индекса не указывается, то удаляется последний элемент.

```
# удаление элемента из списка
z = [0.22, 0.44, 0.66, 0.88, 0.99]
print(z)
z.pop(1)
print(z)
z.pop(-2)
print(z)
z.pop()
print(z)
```

```
z.pop()
print(z)
```

Результат выполнения программы:

```
[0.22, 0.44, 0.66, 0.88, 0.99]
[0.22, 0.66, 0.88, 0.99]
[0.22, 0.66, 0.99]
[0.22, 0.66]
[0.22]
```

Другие методы обработки списков

Всего в Python определены 11 методов обработки списков. Три метода (append, insert, pop) рассмотрены подробно. Оставшиеся методы и их действия продемонстрированы в следующей программы:

```
# методы списков
# метод extend-объединение(сложение) списков
mas1 = [1, 3, 5, 3, 6, 3]
mas2 = [2, 3, 10, 9, 10]
print('mas1 =', mas1, ' mas2 =', mas2)
mas1.extend(mas2)   # объдиненный массив
print('       объединенный массив - ', mas1)
# метод remove-удаление элемента (первое вхождение)
mas1.remove(10)
print('удален первый элемент 10 - ', mas1)
# метод count-число конкретных элементов в списке
print('число элементов со значением "3"
=',mas1.count(3))
# метод index-первое вхождение элемента в список
print('элементы "2" и "3" на позициях',
mas1.index(2), 'и', mas1.index(3))
# метод sort-сортировка списка
mas1.sort()
print('    упорядоченный массив -', mas1)
# метод reverse-изменение порядка элементов на
обратный
mas1.reverse()
print('массив в обратном порядке - ', mas1)
# метод copy-копирование списка
mas = mas2.copy()
print('mas2 = ', mas2, 'mas = ', mas)
# метод clear-удаление всех элементов
```



```
mas2.clear()
print('все элементы массива mas2 удалены, mas2
=',mas2)
```

Результаты выполнения программы:

```
mas1 = [1, 3, 5, 3, 6, 3]
mas2 = [2, 3, 10, 9, 10]
#    объединенный массив - [1, 3, 5, 3, 6, 3, 2, 3,
 10, 9, 10]
#    удален первый элемент 10 - [1, 3, 5, 3, 6, 3, 2,
 3, 9, 10]
число элементов со значением "3" = 4
элементы "2" и "3" на позициях 6 и 1
   упорядоченный массив - [1, 2, 3, 3, 3, 3, 5, 6,
9, 10]
массив в обратном порядке - [10, 9, 6, 5, 3, 3, 3, 3,
2, 1]
mas2 = [2, 3, 10, 9, 10]  mas = [2, 3, 10, 9, 10]
все элементы массива mas2 удалены, mas2 = []
```

На рис. 14.4 приведен полный перечень методов, которые могут применяться при обработке списков. Использование функций (рассмотренных в начале настоящего раздела) позволяют упростить разработку алгоритмов и программ на Python.

append(x)	Добавление элемента x в конец списка list
clear()	Удаление всех элементов, результирующий список[]
copy()	Копирование списка
count(x)	Подсчитывает количество элементов x в списке
extend(e)	Сложение списков list и e
index(x)	Позиция первого элемента x в списке
insert(i, x)	Добавление элемента x на позицию i
pop(i)	Удаление элемента из позиции i
remove(x)	Удаление элемента x из списка(только 1-ое вхождение)
reverse()	Изменение порядка элементов на обратный
sort()	Сортирует список list по возрастанию

Рис. 14.4. Методы списков

Кортежи

Кортеж (tuple) является сложной структурой данных и является последовательностью упорядоченных и неизменяемых элементов.

Кортеж в отличие от списков считается неизменяемым массивом. Таким образом, кортеж – это неизменяемый список. При определении кортежа его элементы перечисляются в круглых скобках вместо квадратных. Если не возникает неоднозначностей, элементы кортежа можно перечислять без скобок.

k1 = (3, 22, 444, -5)

k2 = 1,2,3,4

Пустой кортеж имеет вид:

k3 = ()

Кортеж из одного элемента объявляется следующим образом:

k4 = (“one”,)

Запятая в конце игнорируется.

Синтаксис кортежей можно использовать в левой части оператора присваивания. Например, на основе вычислений в правой части оператора присваивания формируется кортеж и связывается один в один с именами в левой части. Если a, b, c объявлены кортежами, то

a, b, c = b, a, a

a, b = b, a

Аналогичные действия верно выполнять над списками. Кроме того, можно обменять значения элементов в списке, например, m[i], m[i] = m[j], m[i], где m[i], m[j] – элементы массива m.

В случае кортежей подобное действие недопустимо в силу свойства неизменности кортежа, несмотря на то, что обращение к элементу кортежа реализуется с использованием квадратных скобок.

Как и в списках, элементы в кортежах нумеруются с нуля. К кортежам можно применять операцию среза, в результате этой операции образуется новый кортеж:

- защита данных от случайного изменения;
- кортежи в памяти компьютера занимают меньший объем по сравнению со списками;
- обработки кортежей меньшего времени, чем обработка списков.

Для обработки кортежей можно использовать функции и методы, применяемые при обработке списков. Однако недопустимо использование методов, которое связаны с изменением элементов:

append()
extend()
remote()
pop()

} Использовать при отработке кортежей недопустимо!

Приведём программу, которая демонстрирует основные правила обработки кортежей. Промежуточные комментарии и результат выполнения программы обеспечивают возможность овладения приёмами обработки кортежей.

```
# обработка кортежей (tuple)
# задание кортежей
k1 = (123, 456, 'abc')
k2 = tuple(('abc', 987, 456))
print(k1, k2)
# преобразование списка в кортеж и обратно
lst = [1, 2, 3, 4, 5]
print(lst, '- список')
k3 = tuple(lst)
print(k3, '- кортеж')
# обращение к элементу кортежа
print(k1[1])
print(k2[0])
# слияние кортежей и количество элементов в кортеже
k = k1 + k2 + (1, 2)
print(k, 'кол-во элементов', len(k))
# k[1] = 3 изменить значение нельзя - ошибка
# минимальный и максимальный элементы
kk = (34, 56, -4, 99, 2)
print(kk, 'мин. =', min(kk), 'макс. =', max(kk))
# сортировка кортежа
print('кортеж по возрастанию', sorted(kk))
# количество вхождений в кортеж указанного элемента
print('количество элементов "abc" в кортеже равно',\
      int(k.count('abc')))
```

Результаты выполнения программы:

```
(123, 456, 'abc')('abc', 987, 456)
[1, 2, 3, 4, 5] - список
(1, 2, 3, 4, 5) - кортеж
456
abc
(123, 456, 'abc', 'abc', 987, 456, 1, 2) кол-во
элементов = 8
(34, 56, -4, 99, 2) мин. = -4 макс. = 99
кортеж по возрастанию - [-4, 2, 34, 56, 99]
количество элементов "abc" в кортеже равно 2
```

Алгоритмы и программы обработки одномерных массивов

При обработке данных, содержащихся в массиве, можно выделить некоторые алгоритмы и программы, которые часто повторяются и позволяют определить некоторые стандартные процедуры. Одной из таких процедур является программа ввода массива с клавиатуры, алгоритм (и программа) приведен на рисунке 14.3. Другой часто встречающейся задачей при обработке одномерных массивов является вывод массива на экран. Это может быть реализовано различными способами. В простейшем случае в операторе print указывается имя выводимого массива, элементы которого выводятся в квадратных скобках через пробел с помощью простейшего цикла, например,

```
for i in range(N):
    print(A[i], end = '')
```

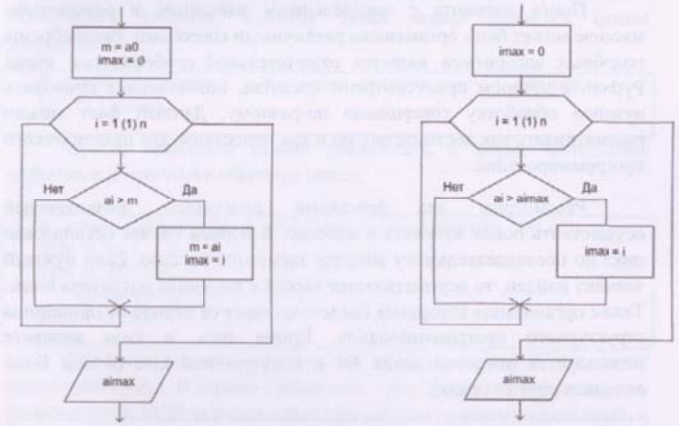
В этом случае, если число элементов выводимого массива A неизвестно, то вывод можно реализовать следующим образом:

```
for x in A:
    print(x, end = '')
```


Определение максимального элемента и его номера в массиве

Поиск максимального (минимального) элемента осуществляется в результате просмотра элементов массива, сравнения текущего значения текущего значения с ранее запомненным, что показано на рис. 14.5. При этом на рис. 14.5.б показан фрагмент улучшенного способа, когда по номеру элемента можно определить его значение.

Фрагменты программы, соответствующие алгоритмам на рис. 14.5.а и рис. 14.5.б, имеют следующий вид:



a) b)

Рис. 14.5. Алгоритмы определения максимального элемента

```
# максимальный элемент и его номер
m = a[0]
imax = 0
for i in range(1, n):
    if a[i] > m :
        m = a[i]
        imax = i
```

```
print("a[", imax, "] = ", m, sep = "")

# максимальный элемент и его номер
imax = 0
for i in range(1, n):
    if a[i] > m :
        imax = i
print("a[", imax, "] = ", a[imax], sep = "")
```

Поиск элемента в массиве

Поиск элемента с определенным значением в одномерном массиве может быть организован различными способами. Разнообразие подобных алгоритмов является отличительной особенностью языка Python, в котором предусмотрены средства, позволяющие проводить нужную обработку совершенно по-разному. Данный факт можно рассматривать как достоинство, но и как недостаток для практического программирования.

Рассмотрим два фрагмента программы, позволяющей осуществить поиск элемента в массиве. В первом случае организован цикл по последовательному анализу элементов массива. Если нужный элемент найден, то осуществляется выход с помощью оператора break. Также организация алгоритма свидетельствует об отходе от принципов структурного программирования. Кроме того, в этом варианте используется оператор цикла for с конструкцией else (о чем было сказано в других темах).

```
# поиск элемента x (используется цикл)
for i in range(n) :
    if a[i] == x:
        print( 'a[', i, ']=', x, sep="")
        break
else :
    print('не нашли')
```

Поиск элемента можно организовать без применения цикла. В этом случае необходимо использовать метод index.


```
# поиск элемента x (используется метод)
if x in a :
    i = a.index(x)
    print( 'a[', i, ']=', x, sep="")
else :
    print('не нашли')
```

Сумма элементов массива

Для нахождения суммы элементов массива в Python реализована стандартная функция, поэтому такая задача решается одним оператором:

```
# сумма элементов в массиве (используется функция)
print(sum(a))
```

Сумму элементов можно реализовать в форме накопления отдельных элементов в обычном цикле:

```
# сумма элементов в массиве (используется цикл)
SUM = 0
for x in a :
    SUM += x
print(SUM)
```

Обратите внимание на то, что объекты sum и SUM имеют различный смысл. В первом случае sum – имя стандартной функции, во втором случае SUM является идентификатором (именем) переменной, в которой накапливается сумма. Использование цикла обеспечивает нахождение суммы элементов, обладающих определенным свойством. С этой целью осуществляется последовательный перебор элементов массива и накапливаются только те элементы, которые определяются с помощью проверки некоторого условия, например, сумма четных элементов реализуется следующим образом: