Введение в Python и Eric

Иван Хахаев, 2009

Обработка массивов. Списки и кортежи.

Задачи по обработке одномерных и двумерных массивов (матриц) являются непременным элементом любого начального курса алгоритмизации, поскольку в них необходимо чётко понимать последовательность действий и объекты, над которыми совершаются эти действия.

Как правило, практическая реализация задач по обработке массивов распадается на две части — формирование массива путём ввода элементов (с клавиатуры или из файла) или генерации элементов и собственно обработка уже имеющегося массива.

Поскольку для массивов количество элементов известно ещё до этапа формирования массива, то для операций с элементами массива разумно использовать цикл (составной оператор) `for`, в котором счётчик повторений связывается с номерами элементов массива.

Python не имеет специального типа данных «массив», для работы с наборами данных используются списки или кортежи.

Однако, поскольку кортеж является набором данных с фиксированным количеством элементов, такую структуру довольно трудно использовать при последовательном формировании массива, в котором количество элементов также задаётся какой-либо переменной.

Поэтому для работы с одномерными массивами хорошо подходят списки, а для работы с двумерными попробуем использовать матрицы из библиотеки `numpy` (Numeric Python).

Рассмотрим задачу по перестановке «частей» одномерного массива (`script-061.py`).

```
#  -*- coding: utf-8 -*-
# Дан одномерный массив числовых значений,
# насчитывающий N элементов.
# Поменять местами M первых элементов с группой элементов,
# начинающихся с позиции K.
# M<N/2, K>M
#
n=input("Количество элементов? ")
# создаём пустой список
massiv=[]
for i in range(0,n):
    a=input("Элемент массива: ")
    # добавляем к списку элемент
    massiv.append(a)
#
print "Исходный массив: ",massiv
#
m=input("Количество элементов? ")
k=input("Позиция старта: ")
#
for i in range(0,m):
```

```
    buffer=massiv[k+i-1]
    massiv[k+i-1]=massiv[i]
    massiv[i]=buffer
 #
print "Итоговый массив: ",massiv
```

Пример выполнения программы в IDE Eric показан на рис. 1.

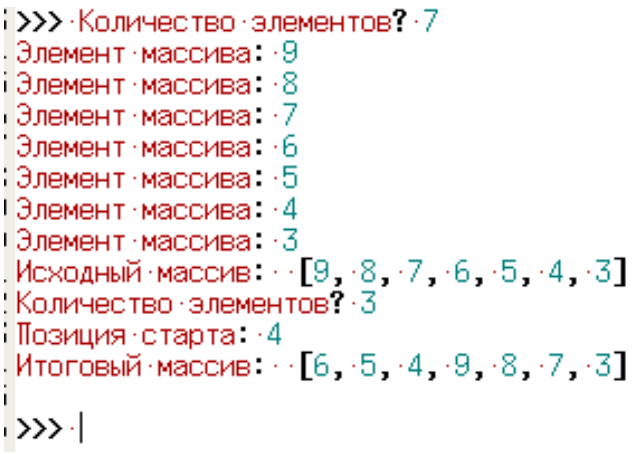

Рисунок 1. Перестановка элементов массива

Важно помнить, что первый элемент в списке имеет номер 0, и все вычисления позиций элементов должны учитывать этот факт.

Для списков в Python (и, соответственно, для одномерных массивов) можно выбирать нужные элементы с помощью так называемых «срезов», указывая через двоеточие начальный и конечный номер элемента в срезе. Так, для рассматриваемого примера команда

```
print "Срез с 3-го по 6-й", massiv[3:6]
```

выведет следующий результат:

```
[9, 8, 7]
```

Для списков и кортежей в Python определены следующие операции:

`len(s)`	Длина списка/кортежа `s`
`x in s`	Проверка принадлежности элемента списку/кортежу. Результат - «Истина» (`True`) или «Ложь» (`False`)
`x not in s`	Проверка отсутствия принадлежности элемента списку/кортежу
`s + s1`	Конкатенация списков/кортежей
`s*n` или `n*s`	Последовательность из `n` раз повторённого `s`.

	Если `n < 0`, результатом будет пустой список/кортеж.
`s[i]`	Возвращает i-й элемент `s` или `len(s)+i`–й, если `i < 0`
`s[i:j:d]`	Срез списка/кортежа `s` от `i` до `j` с шагом `d`. Если шаг не указан, элементы берутся подряд.
`min(s)`	Наименьший элемент `s`
`max(s)`	Наибольший элемент `s`
`s[i] = x`	i-й элемент списка/кортежа `s` заменяется на `x` (замена значения элемента)

Нужно отметить, что строки, рассмотренные в предыдущей главе, в Python являются вариантом списков, поэтому к строкам применимы многие операции, относящиеся к спискам.

Рис. 2 иллюстрирует применение функций `max()` и `min()` к массиву из рассмотренного примера.

```
>>> max(massiv)
9
>>> min(massiv)
3
>>>
```

Рисунок 2. Получение максимального и минимального элементов массива

Кроме того, только для списков (но не для кортежей!) можно использовать следующие функции:

`s[i:j:d] = t`	Срез от `i` до `j` (с шагом `d`) заменяется на список или элемент `t`
`del s[i:j:d]`	Удаление элементов среза из списка

Также для списков (как объектов Python) имеются специальные методы:

`append(x)`	Добавление элемента в конец списка
`count(x)`	Вычисление количества элементов, равных `x`
`extend(s)`	Добавление к концу списка другого списка `s`

`index(x)`	Определение первой с начала списка позиции элемента `x`. Если `x` не найден в `s`, выдаётся ошибка времени выполнения.
`insert(i, x)`	Вставка элемента `x` в позицию `i`
`pop(i)`	Выбор из списка элекмента с номером `i` и удаление его из списка
`reverse(s)`	Изменение порядка элементов списка `s` на обратный
`sort()`	Сортировка элементов списка `s`. Список сортируется по возрастанию, если не указывать в скобках дополнительных условий сортировки.

Таким образам, задачи определения места максимального (минимального) элемента в массиве решается так:

```
a=max(massiv)
pos=massiv.index(a)
```

где `pos` — номер максимального элемента (считая с 0!)

Упражнение: Не используя специфические функции и методы Python, напишите программу определения позиции максимального элемента массива.

Для закрепления умения работать с одномерными массивами рассмотрим ещё одну задачу (`script-071.py`).

```
#  -*- coding: utf-8 -*-
# Дан одномерный массив целых чисел, насчитывающий N элементов.
# Сумму элементов массива вставить на место,
# соответствующее значению первого элемента массива.
#
n=input("Количество элементов? ")
# создаём пустой список
massiv=[]
for i in range(0,n):
   a=input("Элемент массива: ")
   # добавляем к списку элемент
   massiv.append(a)
#
print "Исходный массив: ",massiv
#
summa=0
for i in range(0,n):
```

```
 summa=summa+massiv[i]
#
massiv.insert(massiv[0]-1,summa)
print "Итоговый массив: ",massiv
```

Пример работы программы показан на рис. 3.

```
>>> Количество элементов? 7
Элемент массива: 4
Элемент массива: 3
Элемент массива: 2
Элемент массива: 1
Элемент массива: 0
Элемент массива: 9
Элемент массива: 8
Исходный массив:  [4, 3, 2, 1, 0, 9, 8]
Итоговый массив:  [4, 3, 2, 27, 1, 0, 9, 8]

>>>
```

Рисунок 3. Пример добавления элементов в массив (список)

Для работы с многомерными массивами можно использовать вложенные списки (списки списков, списки списков списков и т.д.).

Однако Python предоставляет более удобный инструмент создания и преобразования многомерных массивов — библиотеку numpy.

Рассмотрим задачу по обработке двумерного массива — матрицы (`script-09.py`).

```
#  -*- coding: utf-8 -*-
# Выполнить обработку элементов прямоугольной матрицы A,
# имеющей N строк и M столбцов.
# Найти наименьшее значение среди средних значений для каждой
# строки матрицы.
#
import numpy
n=input("Количество строк: ")
m=input("Количество столбцов: ")
# Создаём нулевую матрицу
a=numpy.zeros([n,m])
# Заполняем матрицу
for i in range(0,n):
    for j in range(0,m):
        print "Элемент матрицы [",i,"][",j,"]"
        a[i,j]=input("Введите элемент: ")
```

```
    continue
# Выводим исходную матрицу
print "Исходная матрица: "
print a
# Создаём массив для средних значений по строкам
s=[]
#
for i in range(0,n):
    sum=0.0
    for j in range(0,m):
        sum=sum+a[i,j]
# заполняем массив средних значений (список)
    s.append(sum/m)
#
print "Средние значения по строкам: ",s
print "Наименьшее среди средних значений: ",min(s)
```

Пример выполнения программы показан на рис. 4.

Важным является момент создания матрицы, заполненной нулями (`zeros([n,m])`) нужного размера.

Для заполнения матрицы данными используются вложенные циклы — выбирается номер строки, а затем добавляются элементы этой строки. Аналогично, вложенные циклы используются при вычислении средних значений по строкам.

Оператор (команда) `continue` бывает полезен для обеспечения перехода к следующей итерации («обороту») цикла.

Обратите внимание, что для вывода матрицы не нужно предпринимать вообще никаких дополнительных усилий — достаточно простой команды `print`. Кроме того, заметим, что все значения в матрице имеют вещественный тип.

А собственно задача вычисления средних значений и получения минимального из них решается по хорошо известному алгоритму и с применением функций обработки списков в Python.

```
>>> Количество строк: 3
Количество столбцов: 4
Элемент матрицы [ 0 ][ 0 ]
Введите элемент: 1
Элемент матрицы [ 0 ][ 1 ]
Введите элемент: 2
Элемент матрицы [ 0 ][ 2 ]
Введите элемент: 3
Элемент матрицы [ 0 ][ 3 ]
Введите элемент: 4
Элемент матрицы [ 1 ][ 0 ]
Введите элемент: 5
Элемент матрицы [ 1 ][ 1 ]
Введите элемент: 6
Элемент матрицы [ 1 ][ 2 ]
Введите элемент: 7
Элемент матрицы [ 1 ][ 3 ]
Введите элемент: 8
Элемент матрицы [ 2 ][ 0 ]
Введите элемент: 9
Элемент матрицы [ 2 ][ 1 ]
Введите элемент: 0
Элемент матрицы [ 2 ][ 2 ]
Введите элемент: 1
Элемент матрицы [ 2 ][ 3 ]
Введите элемент: 2
Исходная матрица:
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9.  0.  1.  2.]]
Средние значения по строкам:  [2.5, 6.5, 3.0]
Наименьшее среди средних значений:  2.5

>>>
```

Рисунок 4. Пример обработки матрицы