Лекция №10.

Тема: **Списочный тип данных в языке Python.**

Тип данных list. Инициализация переменных типа list. Операции над списками. Стандартные методы по работе со списками.

Ключевые слова: список, индекс, массив, генератор списка, операции над списками, методы по работе со списками.

Keywords: list, index, array, generator expressions, list operators, list methods.

1 Тип данных list

В языке Python для представления списков вводится тип данных list (см. https://docs.python.org/3.5/library/stdtypes.html?highlight=list#list). Особенности типа данных list:

- список представляет собой изменяемую (mutable) упорядоченную последовательность элементов с произвольным доступом. Примеры

```
>>> # Элементы списка разделяются запятой и заключаются в квадратные скобки.
>>> a = [15, 65, 87]
>>> a
[15, 65, 87]
```

```
>>> # Список - изменяемая последовательность элементов.
>>> a = [15, 65, 87]
>>> a.append(53)
>>> a.remove(15)
>>> a
[65, 87, 53]
```

```
>>> # Список - последовательность с произвольным доступом, т.е. допустимо обращение к элементам списка по индексу. Индекс – целое число или целое выражение
>>> a = [1, 2, 3]
>>> a[1] = a[0] + a[2]
>>> a
[1, 4, 3]
```

- в состав списка могут включаться объекты различных типов данных. Пример

```
>>> a = [1, 2., 'hello', True, False, 5+6j]
>>> a
[1, 2.0, 'hello', True, False, (5+6j)]
```

- тип данных list может вызываться как функция для создания списочных объектов. Примеры:

```
>>> # Список на основе последовательности символов.
>>> a = list('Python')
>>> a
['P', 'y', 't', 'h', 'o', 'n']
```

```
>>> # Список на основе кортежа.
>>> a = list((1, 2, 3, 4, 5))
>>> a
[1, 2, 3, 4, 5]
```

```
>>> # Список на основе множества.
>>> a = list({1, 1, 2, 2, 3, 3})
>>> a
[1, 2, 3]
```

```
>>> # Копирование списка.
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

- время доступа к элементу списка есть величина постоянная и не зависит от размера списка;
- время на добавление одного элемента в конец списка есть величина постоянная;
- время на вставку/удаление элемента зависит от того, сколько элементов находится справа от него (чем ближе элемент к концу списка, тем быстрее выполняется операция);
- время реверса списка пропорционально количеству элементов в списке.

2 Инициализация переменных типа list

Инициализация переменной типа данных list может быть выполнена несколькими способами:

1. Инициализация с помощью конструктора – перечисление элементов списка через запятую в квадратных скобках. Пример

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> a
['P', 'y', 't', 'h', 'o', 'n']
```

2. Инициализация пустым списком. Пример

```
>>> a = list()
>>> a
[]
>>> a = []
>>> a
[]
```

3. Инициализация списка заданным значением

```
>>> # Инициализация элементов списка нулевым значением.
>>> n = int(input('Количество элементов в списке = '))
Количество элементов в списке = 5
>>> a = [0] * n
>>> a
[0, 0, 0, 0, 0]
```

4. Инициализация заданной последовательностью элементов. Примеры:

```
>>> # Инициализация последовательностью строкового типа.
>>> a = list('Python')
>>> a
['P', 'y', 't', 'h', 'o', 'n']
```

```
>>> # Инициализация элементами множества.
>>> a = list({'P', 'y', 't', 'h', 'o', 'n'})
>>> a
['t', 'y', 'P', 'o', 'h', 'n']
```

```
>>> # Инициализация элементами кортежа.
>>> a = list(('P', 'y', 't', 'h', 'o', 'n'))
>>> a
['P', 'y', 't', 'h', 'o', 'n']
```

5. Инициализация с помощью генератора. Примеры:

```
>>> # Генерация квадратов нечетных натуральных чисел от 1 до 10.
>>> a = [ i * i for i in range(1, 11) if i % 2 != 0]
>>> a
[1, 9, 25, 49, 81]
```

```
>>> # Генерация календаря високосных лет с 1900 по 2015 гг.
>>> a = [i for i in range(1900, 2016) if (i%4==0 and i%100!=0) or
(i%400==0)]
>>> a
[1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936, 1940,
1944, 1948, 1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984,
1988, 1992, 1996, 2000, 2004, 2008, 2012]
```

6. Инициализация с помощью данных, полученных из потока ввода. Пример

```
>>> n = 5
>>> a = [0] * n
>>> for i in range(n):
      a[i] = int(input('a[' + str(i) + ']='))

a[0] = 1
a[1] = 2
a[2] = 4
a[3] = 8
a[4] = 16
>>> a
[1, 2, 4, 8, 16]
```

3 Операции над элементами списка и над списком в целом

Рассмотрим особенности операций над списками в языке Python (см. https://docs.python.org/3/tutorial/introduction.html#lists) (табл. 1).

Таблица 1 – Операции над списками

Оператор	Описание	Пример

=	Оператор присваивания. Результат работы оператора – копия ссылки на объект.	`>>> a = [1, 2, 3]`
		`>>> # Создание копии ссылки.`
		`>>> b = a.`
		`>>> b`
		`[1, 2, 3]`
		`>>> # a и в ссылаются на один объект.`
		`>>> b[1] = 20`
		`>>> a`
		`[1, 20, 3]`

+	Оператор конкатенации списков.	`>>> p1 = [1, 2, 3]`
		`>>> p2 = [4, 5, 6]`
		`>>> a = p1 + p2`
		`>>> a`
		`[1, 2, 3, 4, 5, 6]`

+=	Составной оператор конкатенации списков.	`>>> a = [1, 2, 3]`
		`>>> a += [4, 5, 6]`
		`>>> a`
		`[1, 2, 3, 4, 5, 6]`

*	Оператор дублирования элементов списка.	`>>> a = [1, 2] * 3`
		`>>> a`
		`[1, 2, 1, 2, 1, 2]`

*=	Составной оператор дублирования элементов списка.	`>>> a = [1, 2, 3]`
		`>>> a *= 2`
		`>>> a`
		`[1, 2, 3, 1, 2, 3]`

[]	Оператор доступа к элементам списка (нумерация элементов списка начинается с нуля; допускаются отрицательные индексы).	`>>> a = [0] * 5`
		`>>> for i in range(5):`
		`  a[i] = i*i`
		`>>> a`
		`[0, 1, 4, 9, 16]`
		
		`>>> n = 5`
		`>>> a = [0] * n`
		`>>> for i in range(-1 * n, 0):`
		`  n -= 1`
		`  a[i] = n * n`
		`>>> a`
		`[16, 9, 4, 1, 0]`

[::]	Оператор извлечения среза.	`>>> # Создание копии списка.`
		`>>> a = [1, 2, 3]`
		`>>> b = a[:]`
		`>>> b`
		`[1, 2, 3]`
		`>>> # Добавление элемента в конец списка.`
		`>>> a[len(a):] = [4]`
		`>>> a`
		`[1, 2, 3, 4]`

		`>>> # Замена элементов списка.`
		`>>> a[1: ] = [4, 5, 6]`
		`>>> a`
		`[1, 4, 5, 6]`
		`>>> # Вставка элементов в список.`
		`>>> a[1:2] = [7, 8, 9, 10]`
		`>>> a`
		`[1, 7, 8, 9, 10, 5, 6]`
		`>>> # Выборка элементов из списка.`
		`>>> a[::2]`
		`[1, 8, 10, 6]`

`del`	Оператор удаления.	`>>> # Удаление переменной типа list.`
		`>>> a = [1, 2, 3]`
		`>>> del a`
		`>>> a`
		`>>> # Удаление элемента.`
		`>>> a = [1, 2, 3]`
		`>>> del a[1]`
		`>>> a`
		`[1, 3]`
		`>>> # Удаление группы элементов.`
		`>>> a = [1, 2, 3, 4]`
		`>>> del a[1:3]`
		`>>> a`
		`[1, 4]`
		`>>> # Удаление четных элементов.`
		`>>> a = [1, 2, 3, 4, 5]`
		`>>> del a[::2]`
		`>>> a`
		`[2, 4]`

in	Оператор проверки на вхождение – возвращает True, если элемент присутствует в списке; в противном случае возвращает False.	`>>> a = [1, 2, 3, 4]`
		`>>> 2 in a`
		`True`
		`>>> 5 in a`
		`False`

not in	Оператор проверки на вхождение – возвращает True, если элемент не присутствует в списке; в противном случае возвращает False.	`>>> 2 not in a`
		`False`
		`>>> 5 not in a`
		`True`

is	Оператор идентичности – возвращает True, если списки хранятся в памяти по одному и тому же адресу; в противном случае возвращает False.	`>>> a = [1, 2, 3]`
		`>>> b = a`
		`>>> a is b`
		`True`

is not	Оператор идентичности –	`>>> a = [1, 2, 3]` `>>> b = a`

	возвращает True, если списки хранятся в памяти по разным адресам; в противном случае возвращает False.	`>>> a is not b`
		`False`
		`>>> b = a[:]`
		`>>> a is not b`
		`True`

len(s)	Возвращает количество элементов списка.	`>>> a = []`
		`>>> len(a)`
		`0`
		
		`>>> a = [1, 2, 3]`
		`>>> len(a)`
		`3`

min(s)	Возвращает элемент списка с минимальным значением (элементы массивы должны быть одного типа).	`>>> a = [4, 0, 5, -1, 2]`
		`>>> min(a)`
		`-1`
		
		`>>> a = [True, 2, 3, 4]`
		`>>> min(a)`
		`True`

max(s)	Возвращает элемент списка с максимальным значением (элементы массивы должны быть одного типа).	`>>> a = [4, 0, 5, -1, 2]`
		`>>> max(a)`
		`5`
		
		`>>> a = [True, 2, 3, 4]`
		`>>> max(a)`
		`4`

4 Встроенные методы по работе с элементами списка и со списком в целом

Рассмотрим основные методы по работе со списками (см. https://docs.python.org/3.5/tutorial/datastructures.html) (табл. 2).

Таблица 2 – Встроенные методы типа данных list

Метод	Описание	Пример

A.append(x)	Добавляет элемент *x* в конец списка *A*. Аналогично операции: `A[len(A):] = [x]`	`>>> A = [1, 2, 3]`
		`>>> A.append(4)`
		`>>> A`
		`[1, 2, 3, 4]`

A.extend(L)	Добавляет в конец списка *A* все элементы итерируемого объекта *L*. Аналогично операциям:
	`A[len(A):] = L`
	`A += L`
		`>>> A = [1, 2]; L = [4, 5]`
		`>>> S = 'Python'; M = {6, 5}`
		`>>> A.extend(L)`
		`>>> A`
		`[1, 2, 4, 5]`
		`>>> A.extend(S)`
		`>>> A`
		`[1, 2, 4, 5, 'P', 'y', 't', 'h', 'o', 'n']`
		`>>> A.extend(M)`
		`>>> A`
		`[1, 2, 4, 5, 'P', 'y', 't', 'h', 'o', 'n', 5, 6]`

A.insert(i, x)	Вставляет элемент *x* в i-ую позицию списка *A*.	`>>> A = [1, 2, 3]`
		`>>> A.insert(1, 4)`
		`>>> A`
		`[1, 4, 2, 3]`

A.remove(x)	Удаляет первый слева найденный элемент x из списка *A*.	`>>> A = [0, 1, 2, 0, 1]`
		`>>> A.remove(1)`
		`>>> A`
		`[0, 2, 0, 1]`
		
		`>>> # В случае отсутствия элемента возбуждается исключение.`
		`>>> A.remove(8)`
		`Traceback (most recent call last):`
		`  File "<pyshell#52>", line 1, in <module>`
		`  A.remove(8)`
		`ValueError: list.remove(x): x not in list`

A.pop([i])	Удаляет из списка *A* элемент по указанному индексу и возвращает его. Если индекс не указан, то удаляет и возвращает последний элемент.	`>>> A = [1, 2, 3, 4]`
		`>>> x = A.pop()`
		`>>> A`
		`[1, 2, 3]`
		`>>> x`
		`4`

A.clear()	Очищает список *A*. Аналогично операции: `del A[:]`	`>>> A = [1, 2, 3]`
		`>>> A.clear()`
		`>>> A`
		`[]`

A.index(x, [start, [stop]]))	Возвращает индекс первого слева вхождения элемента *x* в список *A* (или в срез *A[start:stop]*).	`>>> A = [1, 2, 1, 2]`
		`>>> A.index(2)`
		`1`
		
		`>>> # В случае отсутствия элемента возбуждается исключение.`
		`>>> A.index(8)`
		`Traceback (most recent call last):`
		`  File "<pyshell#16>", line 1, in <module>`
		`  A.index(8)`
		`ValueError: 8 is not in list`

A.count(x)	Возвращает число вхождений элемента *x* в список *A*.	`>>> A = [1, 2, 1, 2, 1]`
		`>>> A.count(1)`
		`3`
		`>>> A.count(5)`
		`0`

A.sort(key=None, reverse=False)	Сортирует элементы списка *A*.	`>>> # Сортировка по возрастанию элементов списка.`
		`>>> A = [1, 2, 8, 10, -5]`
		`>>> A.sort()`
		`>>> A`
		`[-5, 1, 2, 8, 10]`
		
		`>>> # Сортировка по убыванию элементов списка.`
		`>>> A = [1, 2, 8, 10, -5]`
		`>>> A.sort(reverse=True)`
		`>>> A`
		`[10, 8, 2, 1, -5]`

A.reverse()	Переставляет элементы списка *A* в обратном порядке.	`>>> A = [1, 2, 3, 4]`
		`>>> A.reverse()`
		`>>> A`
		`[4, 3, 2, 1]`

A.copy()	Создает поверхностную копию списка *A* (см. примечание 1). Аналогично операции: `A[:]`	`>>> A = [1, 2, 3]`
		`>>> B = A.copy()`
		`>>> B`
		`[1, 2, 3]`
		`>>> id(A) == id(B)`
		`False`

Примечание 1. В языке Python различают поверхностное и глубокое копирование (см. https://docs.python.org/3.5/library/copy.html?highlight=deep%20copy#copy.deepcopy). Разница между двумя видами копирования существенна только для составных объектов (например, вложенных списков). При поверхностном копировании создается новый объект *B*, подобный объекту *A*, и наполняется (по мере возможности) ссылками на объекты из оригинального объекта *A*. Глубокое копирование создает новый объект *B* и рекурсивно копирует все составные части объекта *A*. Пример:

```
>>> # Разница между видами копирования не существенна для несоставных объектов.
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> b[1] = 20
>>> a
[1, 2, 3]
>>> b
[1, 20, 3]
```

```
>>> # Для составных объектов поверхностное копирование создает ссылку на оригинальный объект.
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> b = a.copy()
>>> b[1][1] = 20
>>> a
[[1, 2, 3], [4, 20, 6]]
>>> b
[[1, 2, 3], [4, 20, 6]]
```

5 Обработка многомерных последовательностей путем использования вложенных списков

Хранение и обработка группы связанных элементов данных является основным требованием большинства программ. Для решения данной задачи, как правило, используются массивы (см. https://en.wikipedia.org/wiki/Array_data_structure). Массивы могут быть одно, двух, ..., многомерные. Концептуально, одномерный массив соответствует вектору в математике, многомерный массив с двумя измерениями соответствует матрице (таблице), многомерный массив с тремя измерениями напоминает куб. Многомерные массивы индексируются двумя и более целыми числами.

В языке Python отсутствует встроенный тип данных «массив». Для одномерных наборов данных применяются списки; для многомерных – вложенные

списки. Списки в языке Python поддерживают возможность создания вложенных конструкций произвольной глубины и в любых комбинациях. На примере матриц рассмотрим базовые задачи по обработке многомерных последовательностей:

- объявление и инициализация матрицы;
- ввод-вывод элементов матрицы;
- обработка элементов матрицы.

1. Для объявления матрицы с помощью вложенных списков необходимо определить ее размерность. Пусть n – количество строк, а m – количество столбцов. Алгоритм решения задачи: создание списка из n элементов, в котором каждый элемент является ссылкой на список из m элементов (рис. 1).

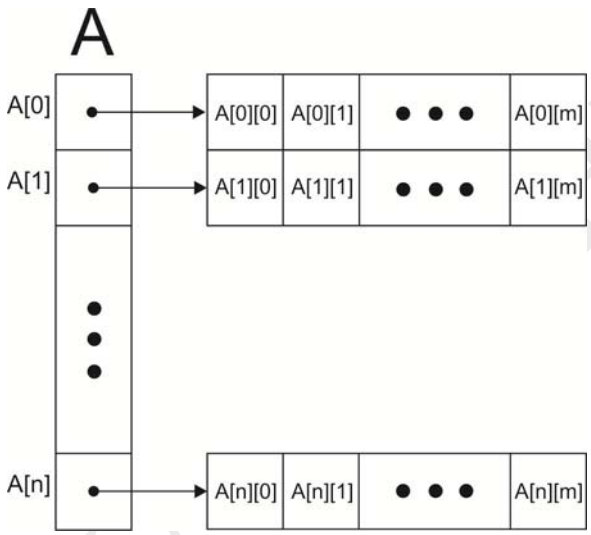

Рис. 1 – Представление двумерного массива с помощью вложенных списков

Рассмотрим несколько способов решения данной задачи средствами языка Python.

```
>>> # Создание списка из n элементов.
>>> A = [0] * n
>>> # В каждый элемент списка A добавляем список из m элементов. Элементы инициализируются нулем.
>>> for i in range(n):
      A[i] = [0] * m
```

```
>>> # Создание матрицы с помощью генератора списков.
>>> A = [[0] * m for i in range(n)]
```

2. В языке Python ввод-вывод элементов матрицы выполняется поэлементно. Для ввода-вывода элементов матрицы, как правило, используются два вложенных цикла. Внешний цикл выполняет обход строк матрицы, а внутренний – обход элементов внутри строки матрицы. Пример

```
>>> # Объявление матрицы.
>>> n = int(input('Количество строк: '))
```

```
Количество строк: 2
>>> m = int(input('Количество столбцов: '))
Количество столбцов: 3
>>> A = [[0] * m for i in range(n)]
>>> # Ввод элементов матрицы.
>>> for i in range(n):
      for j in range(m):
            A[i][j] = float(input('A[' + str(i) + '][' + str(j) + ']='))

A[0][0]=1
A[0][1]=2
A[0][2]=3
A[1][0]=4
A[1][1]=5
A[1][2]=6


>>> # Вывод элементов матрицы.
>>> for i in range(n):
      for j in range(m):
            print(A[i][j], end = '\t')
      print()

1.0   2.0   3.0
4.0   5.0   6.0
```

3. Обработка элементов матрицы. Обращение к отдельному элементу при выполнении операций над элементами матрицы выполняется либо по индексу (индексам) элемента, либо по значению элемента. Примеры:

```
>>> # Вычисление суммы элементов матрицы. Доступ к элементам выполняется по индексу.
>>> S = 0
>>> for i in range(n):
      for j in range(m):
            S += A[i][j]
>>> print(S)
21.0
```

```
>>> # Вычисление произведения элементов матрицы. Доступ к элементам выполняется по значению элемента.
>>> P = 1
>>> for row in A:
      for cell in row:
            P *= cell
>>> print(P)
720.0
```

6 Примеры

Рассмотрим простые примеры программ, использующих стандартные операторы и методы по работе со списками.

Пример №1. Написать программу, которая вычисляет сумму и произведение элементов списка, максимальный и минимальный элемент списка, позиции максимального и минимального элементов списка. Инициализация списка выполняется с помощью генератора случайных чисел.

```
>>> n = int(input('Количество элементов в списке = '))
Количество элементов в списке = 10
>>> A = [random.randint(0,9) for i in range(n)]
>>> A
[2, 4, 5, 3, 0, 3, 7, 9, 9, 0]
>>> # Вычисление суммы и произведения элементов списка
>>> S = 0
>>> P = 1
>>>  # Вычисление произведения элементов списка. Доступ к элементам выполняется по значению элемента.
>>> for el in A:
        S += el
        P *= el
>>> print('Сумма элементов списка: ', S)
Сумма элементов списка:  42
>>> print('Произведение элементов списка: ', P)
Произведение элементов списка:  0
>>> print('Минимальный элемент списка: ', min(A))
Минимальный элемент списка:  0
>>> print('Максимальный элемент списка: ', max(A))
Максимальный элемент списка:  9
>>>  print('Номер первого минимального элемента в списке: ', A.index(min(A)))
Номер первого минимального элемента в списке:  4
>>>  print('Номер первого максимального элемента в списке: ', A.index(max(A)))
Номер первого максимального элемента в списке:  7
```

Пример №2. Написать программу, которая для матрицы вещественных чисел размером 3x4 выполняет обмен строк, которые содержат максимальный и минимальный элементы. Если такие элементы находятся на одной строке, то обмен не выполнять.

```
# Объявление матрицы.
n, m = 3, 4
A = [[0] * m for i in range(n)]
# Инициализация матрицы.
for i in range(n):
    for j in range(m):
        A[i][j] = float(input('A[' + str(i) + '][' + str(j) + ']='))
# Поиск строк с минимальным и максимальным элементами.
min_pos, max_pos = 0, 0
min_el, max_el = min(A[min_pos]), max(A[0])
for i in range(1, n):
    if min(A[i]) < min_el:
        min_el = min(A[i])
        min_pos = i
    if max(A[i]) > max_el:
```

```
                max_el = max(A[i])
                max_pos = i
    # Обмен элементами.
    if min_pos != max_pos:
        A[min_pos], A[max_pos] =  A[max_pos], A[min_pos]
    # Вывод результирующей матрицы.
    for i in range(n):
         for j in range(m):
              print(A[i][j], end = '\t')
         print()
```

Результат работы программы:

```
A[0][0]=2.3
A[0][1]=2.5
A[0][2]=-1
A[0][3]=8
A[1][0]=10
A[1][1]=20
A[1][2]=-78
A[1][3]=45
A[2][0]=65
A[2][1]=90
A[2][2]=-1
A[2][3]=-2

2.3   2.5   -1.0  8.0
65.0  90.0  -1.0  -2.0
10.0  20.0  -78.0 45.0
```