

2. ОБРАБОТКА ТАБЛИЧНЫХ ДАННЫХ И ИХ ВИЗУАЛИЗАЦИЯ

2.1. Списки

Списки в Python – упорядоченные изменяемые коллекции объектов произвольных типов (рисунок 4).



Рисунок 4. Список

Чтобы использовать списки, их нужно **создать**. Например, создадим пустой список:

```
x_list = []
```

или сразу сформируем список из нескольких элементов:

```
x_list = [23, 34.4, -0.45, -3.1, 5]
```

Нумерация элементов в списке начинается с 0, для того чтобы **обратиться** ко второму элементу списка `x_list`, используется запись:

```
x_list[1]
```

значение этого элемента равно 34.4.

Чтобы **заменить** элемент списка `x_list` номером 1, используется оператор присваивания:

```
x_list[1] = -23.14
```

в результате наш список будет иметь вид:

```
[23, -23.14, -0.45, -3.1, 5]
```

Длину списка вычисляют с помощью встроенной функции `len()`:

```
n = len(x_list)
```

В данном случае в переменную `n` будет занесено значение 5.

2.1.1. Оператор цикла, диапазон

Для перебора элементов списка используется **оператор цикла**, с помощью которого осуществляется заданное число повторений вложенного в цикл блока:

```
for ... :  
    операторы
```

Для управления перебором в цикле используется **диапазон**, который описывает неизменяемую последовательность целых чисел от начального значения до границы (не включая ее) с указанным шагом:

```
range(нач_знач, граница, шаг)
```

Формируется диапазон следующим образом:

1. первое число диапазона – это начальное значение;
2. второе (и последующие) вычисляется как предыдущее плюс шаг;
3. число большее или равное границе в диапазон не включается, заполнение диапазона заканчивается.

Формирование значений диапазона рассмотрим на следующем примере:

```
range(3, 11, 2)
```

В этом диапазоне начальное значение равно 3, граница 10, шаг 2:

- первое число диапазона – это начальное значение: 3

- второе число вычисляется как предыдущее плюс шаг $3+2 = 5$: **3 5**
- третье число вычисляется как предыдущее плюс шаг $5+2 = 7$: **3 5 7**
- четвертое число вычисляется как предыдущее плюс шаг $7+2 = 9$: **3 5 7 9**
- пятое число вычисляется как предыдущее плюс шаг $9+2 = 11$, получили число большее или равное границе, поэтому оно в диапазон не включается, заполнение диапазона заканчивается: **3 5 7 9**

Таким образом, получили диапазон, состоящий из четырех чисел.

Описание диапазона может включать 3, 2 и одно значение в качестве параметров.

Принято считать:

- если **range()** содержит 2 значения – пропущен шаг, который по умолчанию равен 1;
- если **range()** содержит 1 значение – пропущен шаг, который по умолчанию равен 1, и начальное значение, которое по умолчанию равно 0.

Например:

```
range(0, 5, 2) результат: 0, 2, 4
range(1, 4) результат: 1, 2, 3
range(5) результат: 0, 1, 2, 3, 4
```

Диапазоны используются в цикле **for**, в котором на каждом шаге управляющей переменной цикла присваивается очередное значение диапазона.

Рассмотрим, как работает цикл на следующем примере:

```
for i in range(0, 4):
    print(i)
```

Диапазон генерирует следующую последовательность чисел:

```
range(0, 4) → 0, 1, 2, 3
```

Выполняется цикл следующим образом:

Шаг 1. Из диапазона выбирается первое значение и заносится в переменную **i** (**i** = 0), это значение становится доступным в теле цикла, с помощью оператора в консоль выводится значение переменной **i**:

```
0
```

Шаг 2. Из диапазона выбирается второе значение и заносится в переменную **i** (**i** = 1), это значение становится доступным в теле цикла, с помощью оператора в консоль выводится значение переменной **i**:

```
0
```

```
1
```

Шаг 3. Из диапазона выбирается третье значение и заносится в переменную **i** (**i** = 2), с помощью оператора в консоль выводится значение переменной **i**:

```
0
```

```
1
```

```
2
```

Шаг 4. Из диапазона выбирается четвертое значение и заносится в переменную **i** (**i** = 3), с помощью оператора в консоль выводится значение переменной **i**:

```
0
```

```
1
```

```
2
```

```
3
```

Шаг 5. Элементы диапазона исчерпаны, значит, цикл заканчивает свою работу.

Другими словами, в цикле последовательно перебираются числа от 0 до 3, составляющие диапазон, последовательно заносятся в переменную **i** и выводятся в консоль.

Пример. Арифметическая прогрессия – это последовательность, в которой каждый следующий член можно найти, прибавив к предыдущему одно и то же число **h**. Вывести члены арифметической прогрессии, первый элемент которой равен целому числу **a**, последний не превышает целого числа **b**, шаг – целое число **h**.

Код:

```
# вводим параметры арифметической прогрессии
a = int(input("a = "))
b = int(input("b = "))
h = int(input("h = "))

# формируем члены арифметической прогрессии и выводим их
for x in range(a, b + 1, h):
    print(x)
```

Результат:

Тест 1

```
a = 6
b = 12
h = 3
6
9
12
```

Тест 2

```
a = 5
b = 12
h = 2
5
7
9
11
```

Задания для самопроверки

1. Для каждой числовой последовательности напишите диапазон, который ее формирует.

- 0, 2, 4, 6, 8;
- -6, -7, -8, -9, -10;
- 3, 6, 9, 12;
- 0, 1, 2, 3, 4, 5.

2. Заполните пропущенные фрагменты программы:

```
a = _
for i in range(2, _):
    a = a + a % i
    print(a)
```

так, чтобы в консоль был выведен следующий результат:

```
10
11
14
18
18
22
```

2.1.2. Генератор списка

Кроме явного создания списков как, например:

```
x_list = [3, 6, 23]
```

их можно создавать с помощью генератора.

Генератор списков – способ построить новый список, применяя выражение к каждому элементу диапазона или другого списка.

Для создания списка, элементы которого формируются в зависимости от значения **переменной-диапазона i**, используется следующий оператор:

```
список = [формула_от_i for i in range(описание_диапазона)]
```

Формирование списка с помощью генератора списка рассмотрим на примере:

```
x_list = [i / (i + 1) for i in range(4)]
```

Диапазон в генераторе имеет вид:

```
range(4) → 0, 1, 2, 3
```

Последовательность генерации списка **x_list**:

- для формирования первого числа в списке из диапазона выбирается первое значение и заносится в переменную **i** ($i = 0$), затем для этой переменной вычисляется выражение $i / (i + 1) = 0 / (0 + 1) = 0$:

```
x_list = [0]
```

- для формирования второго числа в списке из диапазона выбирается второе значение и заносится в переменную **i** ($i = 1$), затем для этой переменной вычисляется выражение $i / (i + 1) = 1 / (1 + 1) = 0.5$:

```
x_list = [0, 0.5]
```

- для формирования третьего числа в списке из диапазона выбирается третье значение и заносится в переменную **i** ($i = 2$), затем для этой переменной вычисляется выражение $i / (i + 1) = 2 / (2 + 1) = 0.666667$:

```
x_list = [0, 0.5, 0.666667]
```

- для формирования четвертого числа в списке из диапазона выбирается четвертое значение и заносится в переменную **i** ($i = 3$), затем для этой переменной вычисляется выражение $i / (i + 1) = 3 / (3 + 1) = 0.75$:

```
x_list = [0, 0.5, 0.666667, 0.75]
```

- элементы диапазона исчерпаны, это значит, что генерация списка заканчивается:

```
x_list = [0, 0.5, 0.666667, 0.75]
```

Таким образом, получили список, состоящий из четырех чисел.

Генерация списков осуществляется с использованием не только диапазонов, но и **других списков**:

```
новый_список = [формула_от_x for x in уже_существующий_список]
```

Формирование списка на основе другого списка рассмотрим на примере:

```
x_list = [5, 8, -5]
```

```
y_list = [x ** 2 for x in x_list]
```

Последовательность генерации списка **y_list**:

- для формирования первого числа в списке **y_list** из списка **x_list** выбирается первое значение и заносится в переменную **x** ($x = 5$), затем для этой переменной вычисляется выражение $x ** 2 = 5 ** 2 = 25$:

```
y_list = [25]
```

- для формирования второго числа в списке **y_list** из списка **x_list** выбирается второе значение и заносится в переменную **x** ($x = 8$), затем для этой переменной вычисляется выражение $x ** 2 = 8 ** 2 = 64$:

```
y_list = [25, 64]
```

- для формирования третьего числа в списке **y_list** из списка **x_list** выбирается третье значение и заносится в переменную **x** ($x = -5$), затем для этой переменной вычисляется выражение $x ** 2 = -5 ** 2 = 25$:

```
y_list = [25, 64, 25]
```

• элементы списка **x_list** исчерпаны, значит генерация списка **y_list** заканчивается:

```
y_list = [25, 64, 25]
```

Таким образом, получили список, состоящий из трех чисел.

Примеры создания списков:

```
x_list = [2 * i + 1 for i in range(5)] → [1, 3, 5, 7, 9]
x_list = [i for i in range(5,10,2)] → [5, 7, 9]
x_list = [2*i for i in range(3,-3,-1)] → [6, 4, 2, 0, -2, -4]
x_list = [3, 2, 7]
y_list = [x % 2 for x in x_list] → [1, 0, 1]
```

Пример. Дан список, в который занесены различные года. В современном календаре каждый год, номер которого делится на 4, является високосным, за исключением тех, которые делятся на 100 и при этом не делятся на 400. Создать новый список, каждый элемент которого равен True, если соответствующий год из списка високосный, False – если нет.

Код:

```
# список лет
year_list = [1900, 1945, 1980, 2000, 2019, 2020, 2300]

# по году определяем, високосный он или нет
leap_year = [year % 4 == 0 and not (year % 100 == 0 and year % 400 != 0)
              for year in year_list]

# выводим результаты
print(year_list)
print(leap_year)
```

Результат:

```
[1900, 1945, 1980, 2000, 2019, 2020, 2300]
[False, False, True, True, False, True, False]
```

Задания для самопроверки

- Создайте следующие списки с помощью генератора:
 - [0.01, 0.001, 0.0001, 0.00001];
 - [0.5, 0.25, 0.125, 0.0625, 0.03125];
 - [2.1, 5.1, 8.1, 11.1, 14.1].
- Список **y_list** формируется на основе списка **x_list**. Заполните пропущенные фрагменты кода:

```
x_list = [_____]
y_list = [i + i / 10 for i in x_list]
print(y_list)
```

так, чтобы в консоль был выведен следующий результат:

```
[16.5, 22.0, 19.8, 18.7, 35.2]
```

2.1.3. Основные операции со списками

В Python предусмотрено большое количество различных функций для работы со списками. В таблице 14 приведены некоторые из них. В примерах функции применяются к следующему списку:

```
x_list = [1, 2, 3, 4, 1, 3]
```

Таблица 14 – Функции работы со списком

Функция	Описание	Пример
<code>x_list.append(x)</code>	Добавляет элемент в конец списка	<code>x_list.append(-1)</code> <code>[1, 2, 3, 4, 1, 3, -1]</code>
<code>x_list.extend(L)</code>	Расширяет список <code>x_list</code> , добавляя в конец все элементы списка <code>L</code>	<code>x_list.extend([0, 5])</code> <code>[1, 2, 3, 4, 1, 3, 0, 5]</code>
<code>x_list.insert(i, x)</code>	Вставляет в <code>i</code> -й элемент значение <code>x</code>	<code>x_list.insert(2, 7)</code> <code>[1, 2, 7, 3, 4, 1, 3]</code>
<code>x_list.remove(x)</code>	Удаляет первый элемент в списке, имеющий значение <code>x</code> .	<code>x_list.remove(3)</code> <code>[1, 2, 4, 1, 3]</code>
<code>x_list.pop(i)</code>	Удаляет <code>i</code> -й элемент и возвращает его. Если индекс не указан, удаляется последний элемент	<code>x_list.pop(3)</code> <code>[1, 2, 3, 1, 3]</code>
<code>x_list.index(x, start, end)</code>	Возвращает положение первого элемента со значением <code>x</code> (при этом поиск ведется от <code>start</code> до <code>end</code>). Если <code>start</code> и <code>end</code> опущены, то поиск осуществляется по всему списку	<code>x_list.index(1, 3, 5)</code> 4
<code>x_list.count(x)</code>	Возвращает количество элементов со значением <code>x</code>	<code>x_list.count(1)</code> 2
<code>x_list.sort()</code>	Сортирует список по возрастанию	<code>x_list.sort()</code> <code>[1, 1, 2, 3, 3, 4]</code>
<code>x_list.reverse()</code>	Разворачивает список	<code>x_list.reverse()</code> <code>[3, 1, 4, 3, 2, 1]</code>
<code>x_list.clear()</code>	Очищает список	<code>x_list.clear()</code> <code>[]</code>

Пример. В список занесен рост студентов группы. Найти самый маленький и самый большой рост в группе. Посчитать, сколько студентов имеют самый маленький и самый большой рост.

Код:

```
# создадим список с ростом студентов
height_list = [ 1.67, 1.85, 1.58, 1.89, 1.76, 1.89, 1.7, 1.89]

# отсортируем список по возрастанию
height_list.sort()

# первый элемент этого списка - самый маленький рост
height_min = height_list[0]

# последний элемент отсортированного списка - самый большой рост
height_max = height_list[len(height_list) - 1]

# посчитаем количество студентов с самым маленьким ростом
min_count = height_list.count(height_min)

# посчитаем количество студентов с самым большим ростом
max_count = height_list.count(height_max)

# выведем результат:
print("минимальный рост:", height_min, "количество:", min_count)
print("максимальный рост:", height_max, "количество:", max_count)
```

Результат:

```
минимальный рост: 1.58 количество: 1
максимальный рост: 1.89 количество: 3
```

Задание для самопроверки

Решить следующую задачу:

В список занесены численности городов некоторого региона:

```
population = [300125, 50000, 600050, 212000, 120000, 50000, 17000]
```

Из этого списка удалить одно максимальное и одно минимальное значения.

2.1.4. Вывод списка

Пусть список заполнен следующим образом:

```
x_list = [23, -23.14, -0.45, 15]
```

Для вывода этого списка можно использовать несколько способов:

- вывести список с помощью оператора `print()`:

```
print(x_list)
```

список будет выведен в одну строку в квадратных скобках через запятую:

```
[23, -23.14, -0.45, 15]
```

- вывести список в цикле:

```
for x in x_list:
    print(x)
```

в этом случае каждый элемент будет размещен на новой строке:

```
23
-23.14
-0.45
15
```

- вывести список с использованием его индексов:

```
for i in range(len(x_list)):
    print("x[" + i + 1, "]" + "=", x_list[i])
```

результат:

```
x[ 1 ] = 23
x[ 2 ] = -23.14
x[ 3 ] = -0.45
x[ 4 ] = 15
```

Задание для самопроверки

Написать фрагмент программы вывода списка [23, 76, 12, -6, 50], чтобы результат в консоли выглядел следующим образом.

Список:

```
23 - элемент с индексом 0
76 - элемент с индексом 1
12 - элемент с индексом 2
-6 - элемент с индексом 3
50 - элемент с индексом 4
```

2.1.5. Ввод списка

Ввод списка может быть осуществлен разными способами.

Первый способ – **поэлементный ввод**, который реализуется по следующему алгоритму:

- спросить у пользователя количество элементов списка, занести значение в переменную **n** (или просто занести в **n** нужное значение);
- сформировать пустой список;
- реализовать перебор по переменной **i** в цикле для **n** элементов;
- ввести очередной элемент списка и занести его в переменную **x**;
- добавить **x** в конец списка.

Пример. Введем с клавиатуры 5 вещественных чисел и занесем их в список:

```
# количество элементов в списке
n = 5

# формируем пустой список
x_list = []
```

```
# цикл перебора индексов списка от 0 до 4
for i in range(n):
    # вводим очередной элемент списка
    x = float(input("Введите элемент списка: "))
    # добавляем введенный элемент в конец списка
    x_list.append(x)

#выводим список на экран
print(x_list)
```

Результат:

```
Введите элемент списка: 5.9
Введите элемент списка: -7.2
Введите элемент списка: 2.1
Введите элемент списка: 12.45
Введите элемент списка: -2.12
[5.9, -7.2, 2.1, 12.45, -2.12]
```

Вторым способом является ввод элементов списка **в одну строку** через пробел, а затем формирование списка на основе введенной строки. Алгоритм формирования списка:

- попросить пользователя ввести элементы списка в одну строку через пробел;
- полученную строку преобразовать в список с помощью метода **split()**, при этом в список будет занесены введенные числа в виде строк;
- преобразовать элементы списка из строковых переменных в вещественные (или целые).

Пример. Введем с клавиатуры вещественные числа и занесем их в список.

```
# вводим строку с числами
line = input("Введите элементы списка через пробел: ")

# преобразуем строку в список, элементы которого - строки
str_list = line.split()
print("Список из строк:", str_list)

# создаем новый список на основе уже сформированного,
# переводя каждый элемент в вещественное число
x_list = [float(x) for x in str_list]
print("Список из чисел:", x_list)
```

Результат:

```
Введите элементы списка через пробел: 45.7 34.2 -32.2 0.12 -3.1
Список из строк: ['45.7', '34.2', '-32.2', '0.12', '-3.1']
Список из чисел: [45.7, 34.2, -32.2, 0.12, -3.1]
```

Задание для самопроверки

Необходимо ввести два списка: первый список **int_list** состоит из целых чисел, второй **float_list** из вещественных. Исправьте фрагмент программы, в котором эти списки вводятся:

```
str_int = int(input(""))
str_float = input("")
int_list = str_int.split()
float_list = str_float.split()
float_list = [int(x) for x in float_list]
print(int_list)
print(float_list)
```