Практическая работа №6. Работа со списками (list)

Какие мы знаем структуры данных?

- ⇨ списки
- ⇨ кортежи
- ⇨ словар
- ⇨ множества.

Что мы изучим? работу со списками.

НЕМНОГО ТЕОРИИ

1) **Что такое список (list) в Python?**
2) **Как списки хранятся в памяти?**
3) **Создание, изменение, удаление списков и работа с его элементами**
4) **Методы работы со списками**

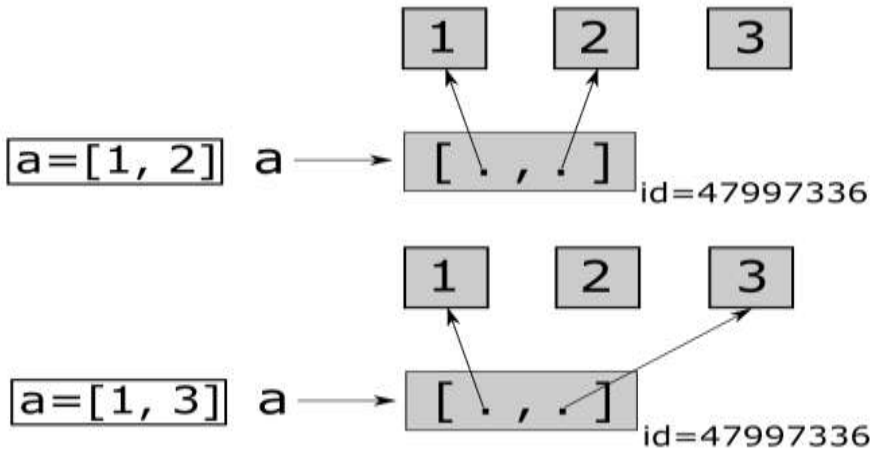
⇨ ЧТО ТАКОЕ СПИСОК (LIST) В PYTHON?

1) **Список (list)** – это структура данных для хранения объектов различных типов.
2) Список очень похож на массив, но в нем можно хранить объекты различных типов.
3) Размер списка можно изменять, т.к. список по своей природе является изменяемым типом данных.
4) ***Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.***

⇨ КАК СПИСКИ ХРАНЯТСЯ В ПАМЯТИ?

Так как список является изменяемым типом данных, при его создании в памяти резервируется область, в котором хранятся ссылки на другие элементы данных в памяти (рисунок 1).

Изначально был создан список, содержащий ссылки на объекты 1 и 2, после операции *a[1] = 3*, вторая ссылка в списке стала указывать на объект 3.



⇨ СОЗДАНИЕ, ИЗМЕНЕНИЕ, УДАЛЕНИЕ СПИСКОВ И РАБОТА С ЕГО ЭЛЕМЕНТАМИ

1) Создать пустой список.

```
1. >>> a = []
2. >>> type(a)
3. <class 'list'>
4. >>> b = list()
5. >>> type(b)
6. <class 'list'>
```

2) Создать список с заранее заданным набором данных.

```
1. >>> a = [1, 2, 3]
2. >>> type(a)
3. <class 'list'>
```

3) Создать копию имеющегося списка

```
>>> a = [1, 3, 5, 7]
1. >>> b = a[:]
2. >>> print(a)
3. [1, 3, 5, 7]
4. >>> print(b)
5. [1, 3, 5, 7]
```

или сделать это так:

```
1. >>> a = [1, 3, 5, 7]
2. >>> b = list(a)
3. >>> print(a)
4. [1, 3, 5, 7]
5. >>> print(b)
6. [1, 3, 5, 7]
```

!!! При ***простом присвоении*** списков друг другу, то переменной *b* будет присвоена ссылка на тот же элемент данных в памяти, на который ссылается *a*, а не копия списка *a*.

!!! Поэтому, если вы будете изменять список ***a, то и*** список ***b тоже будет изменяться.***

```
1. >>> a = [1, 3, 5, 7]
2. >>> b = a
3. >>> print(a)
4. [1, 3, 5, 7]
5. >>> print(b)
6. [1, 3, 5, 7]
7. >>> a[1] = 10
8. >>> print(a)
9. [1, 10, 5, 7]
10.>>> print(b)
11.[1, 10, 5, 7]
```

***4) Добавление элемента в список осуществляется с помощью метода* append().**

```
>>> a = []
>>> a.append(3)
>>> a.append("hello")
>>> print(a)
[3, 'hello']
```

5) Для удаления элемента из списка используется метод* remove(x), *при этом будет удалена первая ссылка на данный элемент.

```
>>> b = [2, 3, 5]
>>> print(b)
[2, 3, 5]
>>> b.remove(3)
>>> print(b)
[x2, 5]
```

***6) Чтобы удалить элемент из списка по его индексу используется команда* del имя_списка[индекс].**

```
>>> c = [3, 5, 1, 9, 6]
>>> print(c)
[3, 5, 1, 9, 6]
>>> del c[2]
>>> print(c)
[3, 5, 9, 6]
```

7) Чтобы ***изменить значение элемента списка по индексу,*** можно использовать операцию присвоения по индексу.

```
>>> d = [2, 4, 9]
>>> print(d)
[2, 4, 9]
>>> d[1] = 17
>>> print(d)
[2, 17, 9]
```

8) Чтобы ***очистить список,*** нужно заново его проинициализировать, так как будто вы его вновь создаете.

```
1.>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
a=[]
2.>>> a = [12, 62, 100]
3. >>> print(a)
4. [12, 62,100]
>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
```

9) Для получения* доступа к элементу списка *укажите индекс этого элемента в квадратных скобках.

```
>>> a = [3, 5, 7, 10, 3, 2, 6, 12]
>>> a[5]
2
```

10) При* указании отрицательных индексов*, отсчет будет идти с конца списка

Например, для доступа к последнему элементу списка можно использовать вот такую команду:

```
>>> a[-2]
6
```

11) Для получения* из списка некоторого подсписка *в определенном* диапазоне индексов*, укажите начальный и конечный индекс в квадратных скобках, разделив их двоеточием.

```
>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
>>> a[5:7]
[5, 7, 10]
```

⇨ **МЕТОДЫ СПИСКОВ**

1) list.append(x) -Добавляет элемент в конец списка.

2) эту же операцию можно сделать - ***a[len(a):] = [x].***

```
>>> a = [1, 2,5.9,89……]
>>> a.append(30)
a[len(a):] = [3]
>>> print(a)
[1, 2, 3]
```

3) list.extend(L)- Расширяет существующий список за счет добавления всех элементов из списка *L*.

4) эту же операцию можно сделать ***a[len(a):] = L.***

```
>>> a = [1, 2]
>>> b = [3, 4]
>>> a.extend(b)
>>> print(a)
[1, 2, 3, 4]
```

5) list.insert(i, x)- Вставить элемент *x* в позицию *i*. Первый аргумент – индекс элемента, после которого будет вставлен элемент *x*.

```
>>> a = [1, 2]
>>> a.insert(9, 5)
>>> print(a)
[5, 1, 2]
>>> a.insert(len(a), 9)
>>> print(a)
[5, 1, 2, 9]
```

6) ***list.remove(x)*** - Удаляет первое вхождение элемента *x* из списка.

```
>>> a = [1, 1, 3,6,9,1]
>>> a.remove(1)
>>> print(a)
[2, 3]
```

7) ***list.pop([i])***-Удаляет элемент из позиции *i* и возвращает его. Если использовать метод без аргумента, то будет удален последний элемент из списка.

```
>>> a = [1, 2, 15, 4, 5]
>>> print(a.pop(2))
15
>>> print(a.pop())
5
>>> print(a)
[1, 2, 4]
```

8) ***list.clear()***- Удаляет все элементы из списка. Эквивалентно ***del*** *a[:]*.

```
>>> a = [1, 2, 3, 4, 5]
>>> print(a)
[1, 2, 3, 4, 5]
>>> a.clear()
>>> print(a)
[]
```

9) ***list.index(x[, start[, end]])***- Возвращает индекс элемента.

```
>>> a = [1, 2, 3, 4, 5,4,4,4,8,9,8]
>>> a.index(4)
3
```

10) ***list.count(x)*** -Возвращает количество вхождений элемента ***X*** в список.

```
>>> a=[1, 2, 2, 3, 3]
>>> print(a.count(2))
2
```

11) ***list.sort(key=None, reverse=False)***- Сортирует элементы в списке по возрастанию.

12) ***list.sort(key=None, reverse= True)***- Для сортировки в обратном порядке Дополнительные возможности открывает параметр ***key,*** за более подробной информацией обратитесь к документации.

```
>>> a = [1, 4, 2, 8, 1]
>>> a.sort()
>>> print(a)
[1, 1, 2, 4, 8]
```

13) list.reverse()- Изменяет порядок расположения элементов в списке на обратный.

```
>>> a = [1, 3, 5, 7]
>>> a.reverse()
>>> print(a)
[7, 5, 3, 1]
```

14) list.copy() - Возвращает копию списка. Эквивалентно *a[:]*.

```
>>> a = [1, 7, 9]
>>> b = a.copy()
>>> print(a)
[1, 7, 9]
>>> print(b)
[1, 7, 9]
>>> b[0] = 8
>>> print(a)
[1, 7, 9]
>>> print(b)
[8, 7, 9]
```

15) List Comprehensions

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка. Представляет простой способ построения списков.

Допустим, вам необходимо создать список целых чисел от *0* до *n*, где *n* предварительно задается.

Классический способ решения данной задачи выглядел бы так:

```
>>> n = int(input())
7
>>> a=[]
>>> for i in range(n):
a.append(i)
>>> print(a)
[0, 1, 2, 3, 4, 5, 6]
```

Использование *list comprehensions* позволяет сделать это значительно проще:

```
>>> n = int(input())
7
>>> a = [i for i in range(n)]
>>> print(a)
[0, 1, 2, 3, 4, 5, 6]
```

А если вам не нужно больше использовать *n:*

```
>>> a = [i for i in range(int(input()))]
78
>>> print(a)
[0, 1, 2, 3, 4, 5, 6,….,77]
```

List Comprehensions как обработчик списков

В языке *Python* есть две очень мощные функции для работы с коллекциями: ***map* и *filter*.**
Они позволяют использовать функциональный стиль программирования, не прибегая к помощи циклов, для работы с такими типами как ***list, tuple, set, dict*** и т.п.

Списковое включение позволяет обойтись без этих функций.

Пример с заменой функции *map*.

Пусть у нас есть список и нужно получить на базе него новый, который содержит элементы первого, возведенные в квадрат.

Решим эту задачу с использованием циклов:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = []
>>> for i in a:
b.append(i**2)

>>> print('a = {}\n b = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Та же задача, решенная с использованием ***map*,** будет выглядеть так:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = list(map(lambda x: x**2, a))
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

В данном случае применена ***lambda***-функция, Лямбда-выражения в Python позволяют функции быть созданной и переданной (зачастую другой функции) в **одной строчке кода**.

lambda-выражения используются при вызове функций (или классов), которые принимают функцию в качестве аргумента.

Через списковое включение эта задача будет решена так:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = [i**2 for i in a]
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Пример с заменой функции filter.

Построим на базе существующего списка новый, состоящий только из четных чисел:

```
>>> a2 = [1, 2, 3, 4, 5, 6, 7]
>>> b = []
>>> for i in a2:
if i%2 == 0:
b.append(i)

>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 50, 6, 7]
b = [2, 4, 50,6]
```

Решим эту задачу с использованием ***filter*:**

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = list(filter(lambda x: x % 2 == 0, a))
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Решение через списковое включение:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = [i for i in a if i % 2 == 0]
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Слайсы / Срезы

Слайсы (срезы) являются очень мощной составляющей *Python*, которая позволяет быстро и лаконично решать задачи выборки элементов из списка.

Выше уже был пример использования слайсов, здесь разберем более подробно работу с ними. Создадим список для экспериментов:

```
>>> a = [i for i in range(10)]
```

Слайс задается тройкой чисел, разделенных запятой: *start:stop:step*.
Start – позиция с которой нужно начать выборку, *stop* – конечная позиция, *step* – шаг.
При этом необходимо помнить, что выборка не включает элемент определяемый *stop*.
Рассмотрим примеры:

```
>>> # Получить копию списка
>>> a[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> # Получить первые пять элементов списка
>>>b= a[0:5]
[0, 1, 2, 3, 4]

>>> # Получить элементы с 3-го по 7-ой
>>> a[2:7]
[2, 3, 4, 5, 6]

>>> # Взять из списка элементы с шагом 2
>>> a[::2]
[0, 2, 4, 6, 8]

>>> # Взять из списка элементы со 2-го по 8-ой с шагом 2
>>> a[1:8:2]
[1, 3, 5, 7]
```

Слайсы можно сконструировать заранее, а потом уже использовать по мере необходимости. Это возможно сделать, в виду того, что слайс – это объект класса *slice*. Ниже приведен пример, демонстрирующий эту функциональность:

```
>>> s = slice(0, 15, 3)
>>> a[s]
[0, 1, 2, 3, 4]

>>> s = slice(1, 8, 2)
>>> a[s]
[1, 3, 5, 7]
```

Типо “*List Comprehensions*”… в генераторном режиме

Есть ещё одни способ создания списков, который похож на списковое включение, но результатом работы является не объект класса *list*, а генератор.

Итератор представляет собой объект перечислитель, который для данного объекта выдает следующий элемент, либо сообщает об ошибке / исключение/, если элементов больше нет.

***Для итераторов использования цикл* for**.

Если ***вы перебираете элементы в некотором списке или символы в строке с помощью цикла* for**, то при каждой итерации цикла происходит обращение к итератору, содержащемуся в строке/списке, с требованием выдать следующий элемент.
Если элементов в объекте больше нет, то итератор генерирует исключение, обрабатываемое в рамках цикла ***for*** незаметно для пользователя.

Приведем несколько примеров, которые помогут лучше понять эту концепцию. Для начала выведем элементы произвольного списка на экран.

```
>>> num_list = [1, 2, 3, 4, 5]
>>> for i in num_list:
    print(i)
1
2
3
4
5
```

Как уже было сказано, объекты, элементы которых можно перебирать в цикле *for*, содержат в себе объект итератор, для того, чтобы его получить необходимо использовать функцию ***iter(),*** а для извлечения следующего элемента из итератора – функцию ***next().***

```
>>> itr = iter(num_list)
>>> print(next(itr))
1
>>> print(next(itr))
2
>>> print(next(itr))
3
>>> print(next(itr))
4
>>> print(next(itr))
5
>>> print(next(itr))
```

```
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    print(next(itr))
StopIteration
```

Как видно из приведенного выше примера вызов функции ***next(itr)*** каждый раз возвращает следующий элемент из списка, а когда эти элементы заканчиваются, генерируется исключение ***StopIteration.***

Предварительно импортируем модуль *sys*, он нам понадобится:

```
>>> import sys
```

Создадим список, используя списковое включение :

```
>>> a = [i for i in range(10)]
```

проверим тип переменной *a*:

```
>>> type(a)
<class 'list'>
```

и посмотрим сколько она занимает памяти в байтах:

```
>>> sys.getsizeof(a)
192
```

Для создания объекта-генератора, используется синтаксис такой же как и для спискового включения, только вместо квадратных скобок используются круглые:

```
>>> b = (i for i in range(10))

>>> type(b)
<class 'generator'>

>>> sys.getsizeof(b)
120
```

Обратите внимание, что тип этого объекта *'generator'*, и в памяти он занимает места меньше, чем список, это объясняется тем, что в первом случае в памяти хранится весь набор чисел от 0 до 9, а во втором функция, которая будет нам генерировать числа от 0 до 9. Для наших примеров разница в размере не существенна, рассмотрим вариант с 10000 элементами:

```
>>> c = [i for i in range(10000)]
>>> sys.getsizeof(c)
87624
>>> d = (i for i in range(10000))
>>> sys.getsizeof(d)
120
```

Сейчас уже разница существенна, как вы уже поняли, размер генератора в данном случае не будет зависеть от количества чисел, которые он должен создать.
Если вы решаете задачу обхода списка, то принципиальной разницы между списком и генератором не будет:

```
>>> for val in a:
print(val, end=' ')

0 1 2 3 4 5 6 7 8 9

>>> for val in b:
print(val, end=' ')

0 1 2 3 4 5 6 7 8 9
```

Но с генератором нельзя работать также как и со списком: нельзя обратиться к элементу по индексу и т.п.

Преобразование строки в список осуществляется с помощью метода ***split(),***
В качестве разделителя по умолчанию используется пробел.

```
>>> l = input().split()
1 2 3 4 5 6 7
>>> print(l)
['1', '2', '3', '4', '5', '6', '7']
```

Разделитель можно заменить, указав его в качестве аргумента метода *split()*.

```
>>> nl = input().split("-")
1-2-3-4-5-6-7
>>> print(nl)
['1', '2', '3', '4', '5', '6', '7']
```

Для считывания списка чисел с одновременным приведением их к типу *int* можно воспользоваться вот такой конструкцией.

```
>>> nums = map(int, input().split())
1 2 3 4 5 6 7
>>> print(list(nums))
[1, 2, 3, 4, 5, 6, 7]
```

Задание:

1) **Отработать Пайтоне все методы по теме**
2) **Переписать в конспект на память**
3) **Оформить в виде отчета в Ворде(скрины)**
4) **Сохранить отчет в PDFформате с титульным листом**
5) **Отправить до 9-00 следующего дня**

С уважением Баян Е