Python.

Списки. Начало.

В предыдущих уроках мы работали с последовательностями чисел, символов, строк, но не сохраняли всю последовательность в памяти компьютера, а обрабатывали ее поэлементно, считывая раз за разом новый элемент. Однако во многих задачах требуется **сохранять всю последовательность**. Например, классическая задача сортировки (упорядочения) некоторой последовательности требует сохранения всех данных в памяти компьютера. Увы, не сохранив, их невозможно отсортировать. И тут на помощь приходит структура данных, которая в большинстве языков программирования называется массивом. В Python она называется **списком**. На самом деле списки имеют более широкие возможности, чем массивы. В процессе изучения мы это заметим.

Элементы списка записываются через запятую в квадратных скобках.

```
a = [2, 12, 23, 17, 156]
b = [2.43, 12.234, 32.11]
c = ["Иван", "Петр", "Василий", "Максим"]
d = [2, 'begin', True, 12.345]
```

В отличии от массивов, элементами одного списка могут быть значения не обязательно одного типа (последняя строка в примере).

Обращаться к элементу списка можно по номеру индекса (нумерация начинается с нуля). Например, если мы обратимся к спискам из предыдущего примера

```
x = a[0]
y = d[3]
```

то в переменной x будет число 2, а в переменную y запишется строка “Максим”.

В качестве индекса может быть любое выражение целого типа. Однако, если индекс будет меньше нуля или больше номера последнего элемента, то возникнет ошибка.

IndexError: list index out of range

Для вывода всего списка можно применить функцию ***print()***:

```
c = ["Иван", "Петр", "Василий", "Максим"]
d = [2, 'begin', True, 12.345]
print(c)
print(d)
```

После выполнения данной программы на экран будет выведено

```
['Иван', 'Петр', 'Василий', 'Максим']
[2, 'begin', True, 12.345]
```

Обратите внимание, что элементы списков выводятся в квадратных скобках и разделяются запятыми, а элементы списка, являющиеся строками - в одинарных кавычках.

Можно выводить элементы списков в более удобном виде, но об этом чуть позже.

Важной особенностью работы со списками является возможность изменять значения отдельных элементов.

```
c = ["Иван", "Петр", "Василий", "Максим"]
c[2] = 'Архип'
print(c)
```

В результате получим

```
['Иван', 'Петр', 'Архип', 'Максим']
```

В отличие от массивов можно добавлять новые элементы в список и удалять существующие элементы из списка. Как это делать мы разберем несколько позже. Но так как количество элементов в списке может меняться, то существует ситуация, когда в списке нет ни одного элемента, хотя сам список существует. Такой список называется пустым.

Пустой список можно задать с помощью квадратных скобок, между которыми ничего не стоит.

```
d = []
```

Кроме того, есть специальная функция, которая создает списки ***list().*** Если ее вызвать без параметров, то она тоже создаст пустой список.

```
b = list()
print(b)
```

В этом случае будет напечатано

[]

Функция ***list()*** помимо создания пустого списка может преобразовывать некоторые типы объектов в списки.

Например, строку она переводит в список символов, из которых состоит строка.

```
s = 'Архип'
b = list(s)
print(b)
```

В результате выполнения программы будет напечатано

['А', 'р', 'х', 'и', 'п']

Можно воспользоваться функциями ***list()*** и ***range()*** для того, чтобы создать список из чисел. Как мы знаем функция ***range()*** создает последовательность целых чисел в заданном диапазоне.

```
b = list(range(10))
print(b)
```

Будет напечатано

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Здесь функция ***range()*** создает последовательность чисел от 0 до 9, а функция ***list()*** преобразует их в список.

```
b = list(range(10,30,3))
print(b)
```

Получим

[10, 13, 16, 19, 22, 25, 28]

Рассмотрим другие функции для работы со списками. Работа со списками напоминает работу со строками. Однако есть одна существенная разница. Саму строку мы изменить никогда не можем, а можем создать новую строку с некоторыми изменениями. Списки мы можем менять.

Функция len()

Функция ***len()*** возвращает длину списка, то есть количество элементов списка.

```
a = [2, 12, 23, 17, 156]
b = [2.43, 12.234, 32.11]
c = ["Иван", "Петр", "Василий", "Максим"]
d = []
print(len(a), len(b), len(c), len(d))
```

будет напечатано

5 3 4 0

Обратите внимание на то, что длина пустого списка равна нулю.

Также следует помнить, что нумерация элементов списка начинается с нуля, и следовательно, номер последнего элемента на 1 меньше длины списка.

Функции min(), max()

Функции ***min(), max()*** возвращают соответственно значение минимального и максимального элемента списка.

```
a = [2, 12, 23, 1, 17, 156]
b = [201.43, 12.234, 32.11]
c = ["Иван", "Петр", "Василий", "Максим"]
print(min(a), max(b), min(c))
```

будет напечатано

1 201.43 Василий

Для строк минимум (максимум) определяется по лексикографическому порядку.

Эти функции работают только если элементы списка одинакового типа, например, числового (как целого, так и вещественного), строкового, логического. Если попытаться найти максимум или минимум в списке со значениями разного типа, а также у пустого списка, то будет ошибка.

Функция sum()

Функция ***sum()*** возвращает сумму всех элементов списка, Элементами списка должны быть только числа. Допустимы булевские значения, так как они могут представляться числами 1 и 0 (True и False соответственно)

```
a = [2, 12, 23, 1,17, 156]
b = [201.43, 12.234, 32.11]
d = [False, True, 1]
print(sum(a), sum(b), sum(d))
```

Получим

211 245.774 2

Вывод элементов списка.

Мы можем выводить весь список в виде элементов, перечисленных через запятую и взятые в квадратные скобки. Однако чаще нам нужны отельные элементы.

В этом случае удобно использовать цикл ***for***.

```
c = ["Иван", "Петр", "Василий", "Максим"]
for nam in c:
    print(nam)
```

В результате будет выведено

Иван
Петр
Василий
Максим

В ходе выполнения цикла переменной ***nam*** поочередно присваиваются элементы списка ***c***.

То же самое можно сделать, обращаясь к элементам списка по индексу.

```
c = ["Иван", "Петр", "Василий", "Максим"]
for i in range(len(c)):
    print(c[i])
```

Пример задачи, когда без использования индекса не обойтись.

Дан список, содержащий целые числа. Известно, что в списке есть только одно число 100. Определить позицию, на которой находится это число. (Нумерация начинается с 0).

Решение

```
a = [2, 12, 23, 100, 17, 156]
for i in range(len(a)):
    if a[i]==100:
        k=i
print(k)
```

Результат

3

Если требуется выводить элементы списка на одной строке, через пробел, то мы можем использовать необязательный параметр ***end*** функции ***print():***

```
c = ["Иван", "Петр", "Василий", "Максим"]
for nam in c:
    print(nam, end=' ')
```

Результат

Иван Петр Василий Максим

Существует способ вывести элементы списка через пробел без использования цикла. Это можно сделать с помощью распаковки списка, использовав знак * перед именем списка.

```
c = ["Иван", "Петр", "Василий", "Максим"]
print(*c)
```

Результат

Иван Петр Василий Максим

Если необходимо вывести каждый элемент с новой строки, воспользуемся оператором ***sep*** функции ***print():***

```
c = ["Иван", "Петр", "Василий", "Максим"]
print(*c, sep='\n')
```

Результат

Иван
Петр
Василий
Максим

Срезы.

По аналогии со строками при работе со списками можно использовать срезы.

С помощью среза мы можем получить несколько элементов списка, создав диапазон индексов, разделенных двоеточием spisok[a:b].

```
a = [2, 12, 23, 1, 17, 156]
print(a[2:5])
print(a[1:4])
print(a[:3])
print(a[3:])
print(a[1:5:2])
print(a[::-1])
```

Получим

[23, 1, 17]
[12, 23, 1]
[2, 12, 23]
[1, 17, 156]
[12, 1]
[156, 17, 1, 23, 12, 2]

Все точно так же работает, как и со строками. Только там отбирались символы, а здесь элементы списка.

Но в отличии от строк с помощью срезов можно изменять содержимое списка.

```
b = [201.43, 12.234, 32.11]
c = ["Иван", "Петр", "Василий", "Максим"]
c[1:3]=["Леонид","Семен"]
print(c)
```

Получим

['Иван', 'Леонид', 'Семен', 'Максим']

Создание списков.

Конечно, если бы списки можно было создавать только, присваивая конкретные значения в программе, это никого бы не устроило. Познакомимся с различными способами создания списков.

Мы знаем, что некоторому элементу списка можно присвоить значение, обратившись к нему по индексу, например введя это значение с клавиатуры. Однако это можно делать только с существующими элементами.

```
c = ["Иван", "Петр", "Василий", "Максим"]
c[2]=input()
print(c)
```

Здесь, если мы введем, например, имя Архип, то получим

['Иван', 'Петр', 'Архип', 'Максим']

Но если мы захотим введенное значение присвоить новому элементу с номером 4, которого нет

```
c = ["Иван", "Петр", "Василий", "Максим"]
c[4]=input()
print(c)
```

то получим ошибку

IndexError: list assignment index out of range

Для того, чтобы добавить новый элемент в конец списка, существует специальный метод append().

```
c = ["Иван", "Петр", "Василий", "Максим"]
c.append("Архип")
print(c)
```

Получим

['Иван', 'Петр', 'Василий', 'Максим', 'Архип']

Возможно добавления элемента не только в конец списка, но это мы рассмотрим позже.

Соответственно, одним из способов заполнения списка является создание изначально пустого списка и по мере необходимости дополнение его.

```
c=[]
for i in range(5):
    c.append(input())
print(c)
```

Если необходимо изначально заполнить массив одинаковыми значениями, например нулями, то можно это сделать с помощью цикла

```
c=[]
for i in range(5):
    c.append(0)
print(c)
```

Получим

[0, 0, 0, 0, 0]

Но есть и более короткий способ

```
c = 5 * [0]
print(c)
```

Если мы заполняем список введенными с клавиатуры числами необходимо не забывать, что данные всегда поступают как строка и их необходимо преобразовать в число.

```
c=[]
for i in range(5):
    c.append(int(input()))
print(c)
```

Во всех рассмотренных случаях мы предполагали, что каждый отдельный элемент вводился на отдельной строке. Но часто бывает, что данные введены в одной строке и

отделены друг от друга пробелами или какими-то другими символами (например, запятыми или точками). Тогда нам на помощь приходит метод работы со строками, который называется ***split().***

Метод split()

Метод ***split()*** разбивает строку на слова, используя в качестве разделителя последовательность пробельных символов и из этих слов создает список.

```
s = "День сегодня выдался теплый"
c=s.split()
print(c)
```

Получим

['День', 'сегодня', 'выдался', 'теплый']

Рассмотрим такую программу

```
c=input().split()
print(c)
```

Если мы с клавиатуры введем 1 2 3 4 5, то получим

['1', '2', '3', '4', '5']

Что, конечно, логично. Ведь метод ***split()*** разбивает строку на слова, которые являются строками. Если нам необходимо рассматривать введенные данные как числа, то необходимо воспользоваться функцией int(). Рассмотрим два способа как это сделать (позже мы рассмотрим и другие варианты).

```
c=input().split()
for i in range(len(c)):
    c[i]=int(c[i])
print(c)
```

Соответственно получим

[1, 2, 3, 4, 5]

Тот же результат можно получить и следующим образом

```
c = []
for x in input().split():
    c.append(int(x))
print(c)
```

Метод pop()

Метод ***pop()*** позволяет извлечь элемент из списка. Если его использовать без параметра, то извлекается последний элемент из списка. При этом сам элемент удаляется из списка.

```
c = ["Иван", "Петр", "Василий", "Максим"]
s = c.pop()
print(s)
print(c)
```

Будет напечатано

Максим
['Иван', 'Петр', 'Василий']

Как можно видеть, в переменную ***s*** был записан последний элемент списка. При этом он был удален из списка.

Мы рассмотрели далеко не все возможности работы со списками. На следующем уроке мы продолжим знакомство со списками