Программа Python.
2021-2022 год.

Давайте вспомним и соберем все функции и методы, которые мы изучили для работы со списками (с массивами).

Пусть a - это массив

a.append(n) - добавляет в конец списка a элемент n

Пример:

```
a = [1,2,3,4,5,3]
a.append(999)

print(a)
```

Результат:

```
[1, 2, 3, 4, 5, 3, 999]
```

Причем можно заменить использование метода **append()** на другой синтаксис:

```
a = [1,2,3,4,5,3]
a += [999]

print(a)
```

Результат:

```
[1, 2, 3, 4, 5, 3, 999]
```

Единственное надо запомнить:

Синтаксис: **a += [999]** равен записи: **a = a + [999]**

Но применять запись: **a = a + [999]** мы не будем, так как такая запись будет нагружать память, и расходовать лишние ресурсы компьютера, при этом нет совершенно никакой в этом необходимости.

Чем интереснее может быть такой способ добавления элементов нашему списку **a[]**? Тем, что мы можем таким образом

добавлять в конец списка любое количество элементов, а не только один, как в случае с использованием метода **append().**

```
a = [1,2,3,4,5,3]
a += [999, 111, 555]

print(a)
```

Результат:

```
[1, 2, 3, 4, 5, 3, 999, 111, 555]
```

Но если мы попробуем добавить три элемента с помощью метода **append()**:

```
a = [1,2,3,4,5,3]
a.append(999, 111, 555)

print(a)
```

Получим ошибку: **builtins.TypeError: append() takes exactly one argument (3 given) -** ошибка говорит о том, что метод **append()** позволяет передать при его вызове только один аргумент. А мы передали три аргумента и получили ошибку.

Даже, если мы передадим одним аргументом список элементов, то получим список, в котором добавлен все равно один элемент - этот элемент уже является списком элементов, а не числом.

```
a = [1,2,3,4,5,3]
a.append([999, 111, 555])

print(a)
```

Результат:

```
[1, 2, 3, 4, 5, 3, [999, 111, 555]]
```

Так что при добавлении элементов в список нужно держать оба синтаксиса в голове, и применять эти способы в зависимости от задач, которые необходимо выполнить в программе.

len(a) - длина списка a - количество элементов списка. Со строками тоже работает.

"...".join(a) - собирает строку из элементов списка a применяя соединение (разделитель) для элементов, указанный впереди в кавычках. Обязательно элементы списка должны быть строками, иначе необходимо перевести каждый элемент списка чисел в строку, к примеру, используя функцию **map()** для списка чисел: **map(str, a).**

Пример:

```
a = [1,2,3,4,5,3]
'*'.join(map(str,a))

print('*'.join(map(str,a)))
```

Результат:

```
1*2*3*4*5*3
```

Если a это строка символов (не список), разделенных пробелами, то можно к строке применить метод split().

Примечание! Метод split() это все-таки для работы со строками, не со списками:

a.split() - функция split() без параметра режет строку по пробельным символам, причем, отбрасывая любое количество пробельных символов и превращает строку в список. Но список получится только в том случае, если есть между символами строки пробелы!

Пример:

```
a = 'fgh1256'
print(a.split())
```

Результат:
['fgh1256'] - список с одним элементом, так как нет пробелов для разреза строки на элементы списка - только один элемент получился

```
a = 'f g h12 56'
print(a.split())
```

Результат:

```
['f', 'g', 'h12', '56']
```

Можно в методе split() указать в параметре, по каким символам резать строку. Получается метод split() можно применять с конкретным параметром.

a.split('.... символ разреза или символы') - метод split() с параметром.

Пример:

```
a = 'fg*h12*56'
print(a.split('*'))
```

Результат:

```
['fg', 'h12', '56']
```

Разрезали строку на элементы списка по звездочкам **'*'**.

```
a = 'fg*h12*56'
print(a.split('*h12'))
```

Результат:

```
['fg', '*56']
```

Разрезали строку на элементы списка по набору символов **'*h12'**.

При этом мы видим, что символы, которые участвуют как разделители строки, не попадают в полученный список элементов.

Можно посмотреть еще один пример использования метода split() без параметра, для очистки строки от пробелов:

```
a = '45              89 \n    -100    \n  \n 10'

print(a.split())
```

Результат:

```
['45', '89', '-100', '10']
```

Метод собирает список из символов, отбрасывая символы пробела и символы перевода строки. Другими словами он убирает все непечатные символы.

Но, если применить метод **split()** к строке без пробелов между символами и попробовать разделить строку на символы по пустоте '' **- это даже не пробел**, то мы получим ошибку:

```
a = 'gdgdgdg'

print(a.split(''))
```

Ошибка: **builtins.ValueError: empty separator** - Эта ошибка нам говорит о том, что мы должны обязательно указать разделитель в параметре метода split(). Исключение только в том случае, если мы применяем метод **split()** без параметра - тогда параметр по умолчанию будет пробел.

```
a = 'g dg dg dg'

print(a.split())
```

Результат:

```
['g', 'dg', 'dg', 'dg']
```

А если нам понадобится разделить строку без пробелов на отдельные символы, то мы можем использовать функцию list()

```
a = 'gdgdgdg'

print(list(a))
```

Результат:

```
['g', 'd', 'g', 'd', 'g', 'd', 'g']
```

list(.....) - в качестве параметра в функции list() могут находиться разные типы данных. Не только строка. Мы уже узнали о таких объектах (типах данных) как map и range.

Например:

```
a = '123456'

print(list(map(int, a)))
```

Результат:

```
[1, 2, 3, 4, 5, 6]
```

При этом сейчас в списке a элементы списка это числа, не строки. Мы можем с ними производить математические вычисления. Немного перепишем предыдущую программу:

```
a = '123456'
a = list(map(int, a))
b = a[0] + a[5]

print(b)
```

Результат:

7 - Это результат сложения чисел 1 и 6!

А если не переводить элементы списка в числа, то мы получим другой результат! Закомментируем строку:

```
# a = list(map(int, a))

a = '123456'
#a = list(map(int, a))
b = a[0] + a[5]

print(b)
```

Результат:

16 - Это уже результат конкатенации строк: **'1' + '6'**

Так же можно поместить в параметр функции **list()** - функцию **range()**:

```
print(list(range(10)))
```

Результат:

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

При этом заметим, что элементы в списке, созданные с помощью функции **range**(), так же будут числами. Мы можем с ними производить математические действия:

```
print(list(range(10))[1] + list(range(10))[2])
```

Результат:

```
3
```

Для списков, так же как и для строк, можно применять технику срезов:

```
a = list('12345')

print(a)
print(a[1:4])
print(a[4:1:-1])
print(a[4:0:-1])
```

Результат:

['1', '2', '3', '4', '5'] - целый список
['2', '3', '4'] - срез
['5', '4', '3'] - срез с отрицательным шагом -1
['5', '4', '3', '2'] - срез с отрицательным шагом -1

Важно всегда помнить и учитывать, что в Python при срезе, не показывается элемент (символ), который относится к правой границе среза - **'до'.** Границы среза: **'от' - включительно, 'до' - не включительно! Это правило надо всегда держать в голове!**

Что еще можно производить со списками?

Давайте попробуем применить метод replace(), которым мы использовали со строками:

Если a - строка:

```
a = '11111'

print(a.replace('1','0'))
```

Результат:

00000 - метод replace() сработал успешно.

```
a = ['1','1','1','1','1']

print(a.replace('1','0'))
```

Результат:

Ошибка: **builtins.AttributeError: 'list' object has no attribute 'replace' -** говорит нам, что объект list не может быть атрибутом метода replace().

Поэтому метод **replace()** отпадает при работе со списками.

Мы еще использовали со строками метод **find()**. Метод **find()** вспомним - возвращает индекс первого вхождения подстроки (символа, символов) в строку.

```
a = '12343'

print(a.find('3'))
```

Результат:

2 – Это индекс первого символа **'3'**, который Python встретил в строке **'12343'.**

А если a - это список, не строка?

```
a = ['1','2','3','4','3']

print(a.find('3'))
```

Результат:
Ошибка: **builtins.AttributeError: 'list' object has no attribute 'find' -** ошибка говорит, что обьект list не может быть атрибутом для метода find().

Получается и метод **find()** отпадает при работе со списками.

А как же нам проверить содержится ли определенное значение элемента в списке?

Вместо метода **find()** для строк, к спискам мы будем применять метод **index()**:

```
a = ['1','2','3','4','3']

print(a.index('3'))
```

Результат:

2 – Это индекс первого символа **'3'**, который Python встретил в списке элементов **a = ['1','2','3','4','3'].**

При этом если в списке искать элемент, которого нет, то мы получим ошибку:

```
a = ['1','2','3','4','3']

print(a.index('5'))
```

Результат:

Ошибка: **builtins.ValueError: '5' is not in list** - ошибка говорит, что в списке нет **'5'.**

Проверить есть ли символ (число) в списке можно еще другим способом:

```
a = ['1','2','3','4','3']

print('5' in a)
```

```
print('2' in a)
```

Результат:

```
False
True
```

Это мы проверили, что **'5'** и **'2'** есть в списке.

Можно проверить, что **'5'** и **'2'** нет в списке:

```
a = ['1','2','3','4','3']

print('5' not in a)
print('2' not in a)
```

Результат:

True - '5' нет в списке - это правда
False - '2' нет в списке - это ложь

Так же со списками мы можем производить **конкатенацию (сложение) и умножение**, как со строками.

```
a = ['1','2','3','4','3']
b = ['999']
print(a + b)
print(a * 2)
print(b * 3)
```

Результат:

```
['1', '2', '3', '4', '3', '999']
['1', '2', '3', '4', '3', '1', '2', '3', '4', '3']
['999', '999', '999']
```

При этом, даже если в списке будут числа, то результат конкатенации и умножения сработает так же как в случае, когда элементы списка это строки.

```
a = [1,2,3,4,5]
b = [999]
print(a + b)
```

```
print(a * 2)
print(b * 3)
```

Результат:

```
[1, 2, 3, 4, 5, 999]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[999, 999, 999]
```

Конечно, мы можем производить арифметические операции с элементами списка (числовыми элементами), как с обычными числами.

Пусть у нас есть список - **a = [1,2,3,4,5].** Создадим новый пустой массив **b[]** и с помощью конструкции цикла **for** переберем элементы списка **a,** и умножим каждый элемент на 2. При этом мы будем постепенно (каждую итерацию) заполнять элементами список **b = [].**

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

print(a)
print(a * 2)
print(b)
```

Результат:

[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5] # Здесь мы умножили весь список чисел a[] на 2
[2, 4, 6, 8, 10] # Здесь мы получили новый массив из чисел списка a[], умноженные на 2

Можно конечно и без создания нового пустого массива **b[].** С помощью цикла **for и range** поменять внутри самого массива **a[]** значения элементов.

```
a = [1,2,3,4,5]
for i in range(5):
    a[i] = a[i] * 2

print(a)
```

Результат:

[2, 4, 6, 8, 10]