**Управляющие конструкции и коллекции**

Сегодня рассмотрим:

* Операции со строками
* Списки
* Циклы

ОПЕРАЦИИ СО СТРОКАМИ

Базовый набор операций следующий:

* .upper() – сделать буквы заглавными
* .lower() – сделать буквы строчными
* .capitalize() – сделать первую букву заглавной, остальные – строчными
* .replace(старая\_подстрока, новая\_подстрока) – замена подстрок в строке
* len(строка) – длина строки
* + – конкатенация (сложение) строк

**Пример:**

*str1 = ‘Убить'*

*str1 = ‘Билла'*

*str3 = str1 + ‘ ’ + str2 # ‘Убить Билла’*

*str3.upper() # ‘УБИТЬ БИЛЛА’*

*str3.lower() # ‘убить билла’*

*str3.capitalize() # ‘Убить билла’*

*str3.replace(‘Убить’, ‘Помиловать’) # ‘Помиловать Билла’*

*len(str3) # 11*

Важным моментом является возможность обращаться по индексам и делать срезы:

* строка[номер] – обращение к элементу строки. При положительном номере обращение идёт с начала строки, а при отрицательном – с конца
* строка[начальный\_номер : количество\_символов] – сделать срез (вернуть подстроку)
* строка[начальный\_номер : количество\_символов : шаг] – создать подстроку из символов, выбранных с данным шагом

**Пример:**

*str = ‘Как приручить дракона'*

*str[4] # ‘п’*

*str[-4] # ‘к’*

*str[0:5] # ‘Как п’. Аналогично можно написать str[:5]*

*str[-4:6] # ‘кард ь’*

*str[0:5:2] # ‘Ккп’*

*str[-4:6:-3] # ‘кд’*

*str[1:] # ‘ак приручить дракона’*

*str[::-1] # ‘анокард ьтичурирп каК’*

Вывод строк выполняется следующими способами:

* с помощью конкатенации (с приведением всех переменных к строке):

**Пример:**

*number = 7  
print(‘Волк и ’ + str(number) + ‘козлят’)*

* в стиле Python 2.x (устаревший вариант)

**Пример:**

*number = 7*

*print(‘Волк и (%d)козлят’ % (number)) # %d указывает, что будет число*

* с помощью метода .format()

*import math*

*number = math.pi*

*print(‘Число Пи приблизительно равно {:.4f}’.format(number))  
# ‘Число Пи приблизительно равно 3,1415’*

Метод .format() позволяет также указывать порядок вывода:

**Пример:**

*print(‘{0}{1}{0}’.format(‘abra’, ‘cad’)) # abracadabra*

* в стиле Python 3.x

*import math*

*number = math.pi*

*print(f‘Число Пи приблизительно равно {number:.4f}’)  
# ‘Число Пи приблизительно равно 3,1415’*

СПИСКИ

Список (list) – набор любых элементов. В том числе – других списков.

**Пример:**

*new\_list = [‘Ах’, ‘как’, ‘бы’, ‘нам’, ‘пришить’, ‘старушку’]*

*new\_list2 = [[‘Съесть’, ‘мягких’, ‘французских’, ‘булок’], [‘Выпить’, ‘чаю’]]*

Аналогично строкам к элементам списка можно обращаться по индексу, над строками можно делать срезы: синтаксис идентичный. Обращаться можно к элементу любой вложенности, менять его:

**Пример:**

*new\_list2[0][1] = ‘чёрствых’*

*new\_list2[1][0:2] = [‘Попробовать’, ‘кофе’]*

*print(new\_list2)*

*# [[‘Съесть’, ‘чёрствых’, ‘французских’, ‘булок’], [‘Попробовать’, ‘кофе’]]*

В Python 3.x списки можно удобно распаковывать, сопоставив количество переменных и длину списка. Если нужно лишь первое значение, то можно выделить его, а остальные поместить в отдельный список. Аналогично для первого и последнего.

**Пример:**

*first, second, third, fourth = [1, 2, 3, 4]*

*print(third) # 3*

*first, \*other = [1, 2, 3, 4]*

*print(first, other) # [1, [2, 3, 4]]*

*first, \*\_, last = [1, 2, 3, 4]*

*print(first, \_, last) # [1, [2, 3], 4]*

ОПЕРАЦИИ СО СПИСКАМИ

Для работы со списками есть много операций. Самыми распространёнными являются:

* sorted(список, reverse) – сортировка. Параметр reverse позволяет задавать порядок сортировки: True – по убыванию, False – по возрастанию. По умолчанию - False
* sum(список) – суммирование элементов
* + – сложение списков
* del(список[индекс]) – удаление элемента по индексу
* .remove(элемент) – удаление элемента по значению
* .append(элемент) – добавить элемент в конец
* .insert(индекс, элемент) – добавить элемент по нужному индексу
* .index(элемент) – узнать индекс элемента
* .count(элемент) – узнать количество вхождений элемента в список
* len(список) – длина списка

**Пример:**

*new\_list = [1, 5, 2, 4, 3]*

*sorted(new\_list) # [1, 2, 3, 4, 5]*

*sorted(new\_list, reverse = True) # [5, 4, 3, 2, 1]*

*sum(new\_list) # 15*

*[1, 2] + [3, 4, 5] # [1, 2, 3, 4, 5]*

*del(new\_list[0]) # [5, 2, 4, 3]*

*new\_list.remove(5) # [1, 5, 2, 4]*

*new\_list.append(6) # [1, 5, 2, 4, 3, 6]*

*new\_list.insert(0, 0) # [0, 1, 5, 2, 4, 3]*

*new\_list.index(3) # 4*

*new\_list.count(5) # 1*

*len(new\_list) # 5*

**Важно:** в примере выше каждая строка выполняется отдельно и не оказывает влияния на изначальный список (для лучшего понимания логики работы). Но в реальности это не так: если имеется некая переменная, содержащая список, а значение этой переменной было присвоено второй переменной, то при изменении второй переменной меняется и первая, поскольку обе ссылаются на один и тот же объект.

**Пример:**

*new\_list = [1, 2, 3, 4, 5]*

*new\_list2 = new\_list*

*new\_list2.append(6)*

*print (new\_list, new\_list2) # [1, 2, 3, 4, 5, 6] [1, 2, 3, 4, 5, 6]*

Чтобы создать новый объект нужно воспользоваться функцией copy() из модуля copy.

**Пример:**

*import copy*

*new\_list = [1, 2, 3, 4, 5]*

*new\_list2 = copy.copy(new\_list)*

*new\_list2.append(6)*

*print (new\_list, new\_list2) # [1, 2, 3, 4, 5] [1, 2, 3, 4, 5, 6]*

СПИСКИ И СТРОКИ

Часто приходится разбивать строки на списки, а также собирать списки в строки. Для этого существуют 2 операции:

* .split(разделитель) – разбить строку в список
* разделитель.join(список) – собрать список в строку

**Пример:**

*new\_string = ‘Ваша мама пришла,молочка принесла’*

*new\_list = new\_string.split(‘ ’) # [‘Ваша’, ‘мама’, ‘пришла,молочка’, ‘принесла’]*

*‘\_’.join(new\_list) # ‘Ваша\_мама\_пришла,молочка\_принесла’*

Кроме того есть операции проверки вхождения элемента в список:

* in – проверка наличия
* not in – проверка отсутствия

**Пример:**

*new\_list = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’]*

*print(‘6’ in new\_list) # False*

*print(‘6’ not in new\_list) # True*

ЦИКЛЫ

Циклы необходимы для повторения определённых действий.

* Цикл while – выполняется пока верно условие цикла. Обязательно необходима конструкция для выхода из цикла. В противном случае будет выполняться бесконечно и приведёт к зависанию.

**Пример:**

*x = 5*

*while x != 0:*

*print(x)*

*x = x – 1 # Конструкция выхода. Можно иначе: x -= 1*

*# 54321 (в Jupyter Notebook – вывод построчный)*

* Цикл for – выполняется определённое количество шагов

**Пример:**

*new\_list = [1, 2, 3, 4, 5]*

*for number in new\_list:*

*print(number + 1)*

*# 23456 (в Jupyter Notebook – вывод построчный)*

Циклы могут быть вложенными:

**Пример:**

*new\_list = [[1, ‘a’], [2, ‘b’], [3, ‘c’], [4, ‘d’], [5, ‘e’]]*

*for sublist in new\_list:*

*for item in sublist:*

*print(item)*

*print(‘ ’) # Добавим разделитель*

*# 1a 2b 3c 4d 5e*

В особых ситуациях в цикле можно воспользоваться операторами:

* break – досрочно завершить цикл
* continue – досрочно перейти к следующей итерации цикла
* pass – ничего не делать, продолжать цикл как обычно

**Пример:**

*new\_string = ‘640КБ оперативной памяти должно хватить всем’*

*for letter in new\_string:*

*if letter == ‘ ’*

*break*

*print(letter)*

*# 640КБ*

*for letter in new\_string:*

*if letter == ‘ ’*

*continue*

*print(letter)*

*# 640КБоперативнойпамятидолжнохватитьвсем*

*for letter in new\_string:*

*if letter == ‘ ’*

*pass*

*print(letter)*

*# 640КБ оперативной памяти должно хватить всем*

Задавать цикл не обязательно при помощи итерации по объекту. Если нужно выполнять действия определённое количество раз, то можно воспользоваться функцией range:

**Пример:**

*new\_string = ‘640КБ оперативной памяти должно хватить всем’*

*# 1 аргумент: количество элементов*

*for index in range(10): # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*

*print(new\_string[index])*

*# 640КБ опер*

*# 2 аргумента: левая граница, правая граница (не включая)*

*for index in range(3, 10): # [3, 4, 5, 6, 7, 8, 9]*

*print(new\_string[index])*

*# КБ опер*

*# 3 аргумента: левая граница, правая граница (не включая), шаг*

*for index in range(3, 10, 2): # [3, 5, 7, 9]*

*print(new\_string[index])*

*# К пр*

В Python 3.x функция range интересна тем, что она возвращает не список элементов, а некоторый объект, способный эти элементы сгенерировать. Это удобно тем, что при больших значениях не засоряет оперативную память, а подгружает данные по надобности. В Python 2.x функция так себя не вела, а подобное поведение демонстрировала функция xrange. Поэтому не рекомендуется использовать range в Python 2.x.