**Третье занятие**

Темы, которые мы будем разбирать на этом занятии:

***While***

Нам нужно начать изучать while. While – это цикл, который исполняется до той поры, пока будет верно какое-то логическое условие. У while есть некоторые преимущества и сфера использования – к примеру, с помощью него можно более маленькие программы (правда, всего на несколько символов). Кроме того, while используется для выполнения каких-либо действий, пока, верно, какое-либо условие. Например, в компьютерной игре у вас персонаж может наносить урон, только пока у него hp больше какого-либо уровня:

<code>

hp = 100

while hp > 40:

print("a hero can attack")

</code>

Методы работы со строками.

Мы посмотрим на полезные функции, которые есть в строках, к примеру служебные символы экранирования строк, которые можно применять, к примеру, для переноса текста на следующую строку в выводе или подаче звукового сигнала.

После этого мы можем использовать полученные знания и построить несложные программы, которые могут выводить на каждой строке одну и ту же строчку:

<code>

#пример умножения

print(‘Ozon is best company\n’ \* 100)

# выводим фразу 100 раз подряд

</code>

Также мы можем получать значение из строчки по индексу, точно также, как мы получаем элемент строки. Однако, в отличии от массивов, строки в Python неизменяемые. Для подобных операций, можно, к примеру, трансформировать строку в массив и работать с ним, а уже потом его трансформировать его в строку обратно, это классический вариант. Мы можем попробовать воспользоваться программой, которая занимается тем, что переводит строку в массив из строк:

<code>

str = 'ozon is the best company'

arr = str.split()

print(arr)

#['ozon', 'is', 'the', 'best', 'company']

</code>

Так же можно попробовать программу, которая анализирует данные из массива и достает только те данные, которые нам нужны, к примеру, только сковородки:

<code>

products = ['skovodka', 'skovorodki', 'chainick', 'kniga', 'shkag', 'sigara', 'pampersi', 'istoria']

skovorodki = []

#представим, что у нас этот список неизмеримо больше (к примеру, размером хотя бы с половину товаров Ozon)

for word in products:

if word.startswith("sko") == True:

skovorodki.append(word)

print(skovorodki)

#другая проблема в том, что у нас подхватится и другие данные, которые начинаются с sko

</code>

**Генераторы и списки**

С помощью синтаксической конструкции for можно создавать списки такой длины и содержания, как только пожелаете:

<code>

numbers = list(range(1,14))

print(numbers)

#создастся массив от 1 до 14

</code>

Вы можете использовать третий аргумент для того, чтобы уточнить шаг генерации:

<code>

even\_numbers = list(range(2,20,2))

print(even\_numbers)

# выведется [2, 4, 6, 8, 10, 12, 14, 16, 18]

</code>

Кроме этого, списки еще можно создавать с помощью генератора. Генератор объединяет цикл for и создание новых элементов в одну строку и автоматически добавляет к списку новые элементы. Это непростая конструкция, но периодически она встречается:

<code>

cube = [value\*\*3 for value in range(1,10)]

print(cube)

</code>

Генераторы это непростая тема, мы еще вернемся к ней в процессе наших занятий.

Есть несколько других методов для работы со списками, которые можно использовать для получения полезной информации, к примеру, чтобы узнать характеристики в словарях:

<code>

len(cars) - возвращает длину массива

max(cars) – возвращает самое большое значение в массиве

min(cars) – возвращает самое маленькое значение в массиве

</code>

**Работа со срезами**

Срезы дают возможность работать с небольшим кусочком массива. К примеру, вы хотите взяли колонку данных из csv, однако самое первое значение в колонке – это у нас часто только ее название, и ее нужно отсечь:

<code>

data = ["number", 3,3,24,5,2,3,5,3,4]

data = data[1:]

print(data)

print(max(data))

print(min(data))

</code>

С помощью срезов мы можем наглядно объяснить разницу в передаче по ссылке. К примеру, если просто передать какое-либо значение по операнду =, тогда у вас образуется одна ссылочная область. Однако с помощью, например, срезов вы сможете создавать дубль списка, к примеру, если вы захотите не вызывать изменения (мутации) у первоначального списка, а захотите создавать свой отдельный.

**Условные конструкции**

Условные конструкции мы уже разбирали, поэтому мы можем еще посмотреть и на остальные части условных конструкций, которые есть в условных конструкциях. Там есть конструкция elif, которая будет выбираться, если первое логическое условие if не выполняется, также мы можем посмотреть на логические and и or:

Здесь у нас пример логического И:

<code>

age = input(" введите ваш возраст: ")

age = int(age)

prof = input("Введите вашу профессию: ")

if age > 18 and prof =="programist":

print("Здорово, теперь можете приступать к работе")

else:

print("Мы предложим вам другую позицию")

</code>

И здесь мы можем привести пример логического ИЛИ:

<code>

age = input(" введите ваш возраст: ")

age = int(age)

prof = input("Введите вашу профессию: ")

if age > 18 and prof =="programist" or prof == 'программист’:

print("Здорово, теперь можете приступать к работе")

else:

print("Мы предложим вам другую позицию")

</code>