Теоретический материал к занятию Генераторы и итераторы 4 занятие.

Итератор (iterator) - это объект, который возвращает свои элементы по одному за раз. С точки зрения Python - это любой объект(экземпляр класса), у которого есть метод __next__. Этот метод возвращает следующий элемент, если он есть, или возвращает исключение StopIteration, когда элементы закончились. Кроме того, итератор запоминает, на каком объекте он остановился в последнюю итерацию.

Коллекции, строки — это все итерируемые объекты. Они являются итерируемыми контейнерами, из которых вы можете получить итератор. Все эти объекты имеют метод iter(), который используется для получения итератора.
Получим итератор из строки и выведем каждое значение

```
a = 'hello'
iterator = iter(a)
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
```

Когда мы перебрали весь объект, то на следующей итерации мы видим исключение StopIteration.

Также получим итератор из кортежа

```
tuple1 = ("яблоко", "банан", "вишня")
myit = iter(tuple1)
print(next(myit))
print(next(myit))
print(next(myit))
```

Мы также можем использовать цикл for для итерации по итерируему объект.

```
tuple1 = ("яблоко", "банан", "вишня")
for x in tuple1:
    print(x)
```

Чтобы создать объект/класс в качестве итератора, вам необходимо реализовать методы __iter__() и __next__() для объекта.

Как вы узнали из занятий по ООП, у всех классов есть функция под названием __init__(), которая позволяет вам делать инициализацию при создании объекта.

Метод __iter__() действует аналогично, вы можете выполнять операции (инициализацию и т.д), Но всегда должны возвращать сам объект итератора.
Метод __next __ () также позволяет вам выполнять операции и должен возвращать следующий элемент в последовательности.
Создадим итератор, который возвращает числа, начиная с 1, и увеличивает на единицу (возвращая 1,2,3,4,5 и т. д.):

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x


myclass = MyNumbers()
myiter = iter(myclass)
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

Данный пример будет продолжаться вечно, пока вы вызываем оператор next() или если используем в цикле for. Чтобы итерация не продолжалась вечно, мы можем использовать оператор StopIteration.

В метод __next __() мы можем добавить условие завершения, чтобы вызвать ошибку, если итерация выполняется указанное количество раз:

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        if self.a <= 20:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)
for x in myiter:
    print(x)
```

Самый простой способ создания собственных итераторов в Python — это создание генератора.

Генераторы могут быть рекурсивными, как любая другая функция. Напишем генератор перестановок элементов в списке.Логика генератора: функция перемещает каждый элемент списка на первое место, заменяя его первым элементом в списке.

```
def func_gen(items):
    if len(items) == 0:
        yield []
    else:
        pi = items[:]
        for i in range(len(pi)):
            pi[0], pi[i] = pi[i], pi[0]
            for p in func_gen(pi[1:]):
                yield [pi[0]] + p

for p in func_gen([1, 2, 3]):
    print(p)
```