Декораторы

```
def decorator(func):
    # Декорирующая функция
    def decorated(*args, **kwargs):
        # Дополнительные действия "до"
        ...
        res = func(*args, **kwargs)
        # Дополнительные действия "после"
        ...
        return res
    return decorated
# Применяем декоратор через @
@decorator
def my_function(arg): ...
```

Итераторы

```
my_list = [1, 4, 6]
# Получить итератор из списка:
iter_list = iter(my_list)
# Получить следующий объект:
elem = next(iter_list)
# Перебрать все элементы из итератора
for elem in iter_list:
    ...
# Записать все значения в список:
new_list = list(iter_list)
```

Генераторы

```
def my_generator(num):
    for i in range(num):
    # Используем yield для выдачи результата
        yield i
    # Выполнение функции замораживается
# Получить экземпляр итератора:
new_gen = my_generator(5)
# Генератор – тоже итератор:
elem = next(new_gen)
# Сгенерировать всё в список:
```

```
l = list(new_gen)
```

Списочные сокращения

```
# Получить генератор в одну строку:
gen = (x**2 for x in range(10))
# Сохранить результаты в список:
new_list = [x**2 for x in range(10)]
# Или во множество:
new_list = {x**2 for x in range(10)}
# Или в кортеж
new_list = tuple(x**2 for x in range(10))
# Воспользоваться условием:
gen = (x**2 for x in range(10)\
       if x%2 == 0)
```

Функции для итераторов

```
# Применить функцию к итератору
map(function_name, my_iterator)
# Отобрать элементы по условной функции:
filter(condition, my_iterator)

# Сгруппировать элементы из итератора:
list1 = [1,3,4,6]
list2 = [5,9,10,13]
for a, b in zip(list1, list2):
   print(a, b)
```