Ярошевич В.А.

090302-ИСТа-о22

ФИТУ 3 курс

**Лабораторная работа №3**

**РАБОТА С ИТЕРАТОРАМИ, ГЕНЕРАТОРАМИ. РАБОТА С ГЕНЕРАТОРНЫМИ ВЫРАЖЕНИЯМИ**

**Цель работы:** изучить понятия итератора и генератора в Python, а также их преимущества; ознакомиться с примерами их пользования. **Задание №7:** написать функцию, которая принимает два одинаковых по длине списка и с помощью генераторного выражения создает и возвращает новый словарь, в котором ключами являются элементы первого списка, а значениями ключей – элементы второго.

**Краткая теория**

*Итераторы* – популярный поведенческий паттерн проектирования для последовательного обхода коллекции, который позволяет не раскрывать их внутреннего представления.

*Итерируемый объект* – это такой объект, от которого можно получить итератор. В Python итерируемым объектом является такой объект, от которого встроенная функция *iter()* возвращает итератор.

*Итератором* в Python является объект, который реализует метод \_\_*next*\_\_ без аргументов и метод *\_\_iter\_\_.* Метод - \_\_*next\_\_* должен вернуть следующий элемент или ошибку *StopIteration*.

Преимущества использования итераторов, как было сказано выше, заключается в возможности «указывать» на определенный объект коллекции и при этом скрывать его структуру. Все последовательности (*list, tuple, range*) в Python являются итерируемыми объектами.

Основным местом работы с итераторами в данной лабораторной работе будет использование цикла *for*. Например, при переборе элементов списка или другой последовательности, используя цикл *for*, фактически происходит обращение к итератору данной последовательности с просьбой выдать следующий элемент. Когда элементы в последовательности заканчиваются, очередное обращение к следующему объекту итератора сгенерируют исключение, которое при использовании цикла *for* обрабатывается незаметно для пользователя.

*Генератор (генераторная функция)* – это функция, которая возвращает подвид итератора, генерирующий значения. Основное их отличие в том, что они экономят память за счет того, что хранят не все значения, например, списка, а только его предыдущий элемент, предел и формулу, по которой рассчитывается следующий элемент. Данная функция вместо *return* содержит ключевое слово *yield*, которое возвращает объект-генератор, а не выполняет сразу весь код. В *Python* имеется возможность создавать генераторный объект в сокращенной форме. Для этого используются круглые скобки.

Перебрать генератор можно используя цикл *for* как и при работе с итераторами. Однако стоит заметить, что перебрать второй раз генератор не получится, потому что объект генератор уже сгенерировал данные по описанной в нем формуле. Поэтом генераторы стоит использовать, когда требуется один раз пройтись по итерируемому объекту.

**Ход работы**

Ознакомившись с материалом, можно приступать к выполнению задания. В данной лабораторной будет всего три этапа:

1) Проверка длины.

2) Генераторное выражение.

3) Возврат результата.

1. Сначала функция проверяет, что оба списка имеют одинаковую длину. Если нет, генерируется ошибка.

2. Мы используем *zip()* для объединения элементов двух списков парами, а затем создаем словарь с помощью генераторного выражения *{k: v for k, v in zip(keys, values)}.*

3. Просто возвращаем созданный словарь в *print(result\_dict)*

Листинг кода:  
def create\_dict\_from\_lists(keys, values):

if len(keys) != len(values):

raise ValueError()

return {k: v for k, v in zip(keys, values)}

keys\_list = ["Дверь", "Окно", "Проем"]

values\_list = [1, 2, 3]

result\_dict = create\_dict\_from\_lists(keys\_list, values\_list)

print(result\_dict)

**Результаты выполнения кода:**

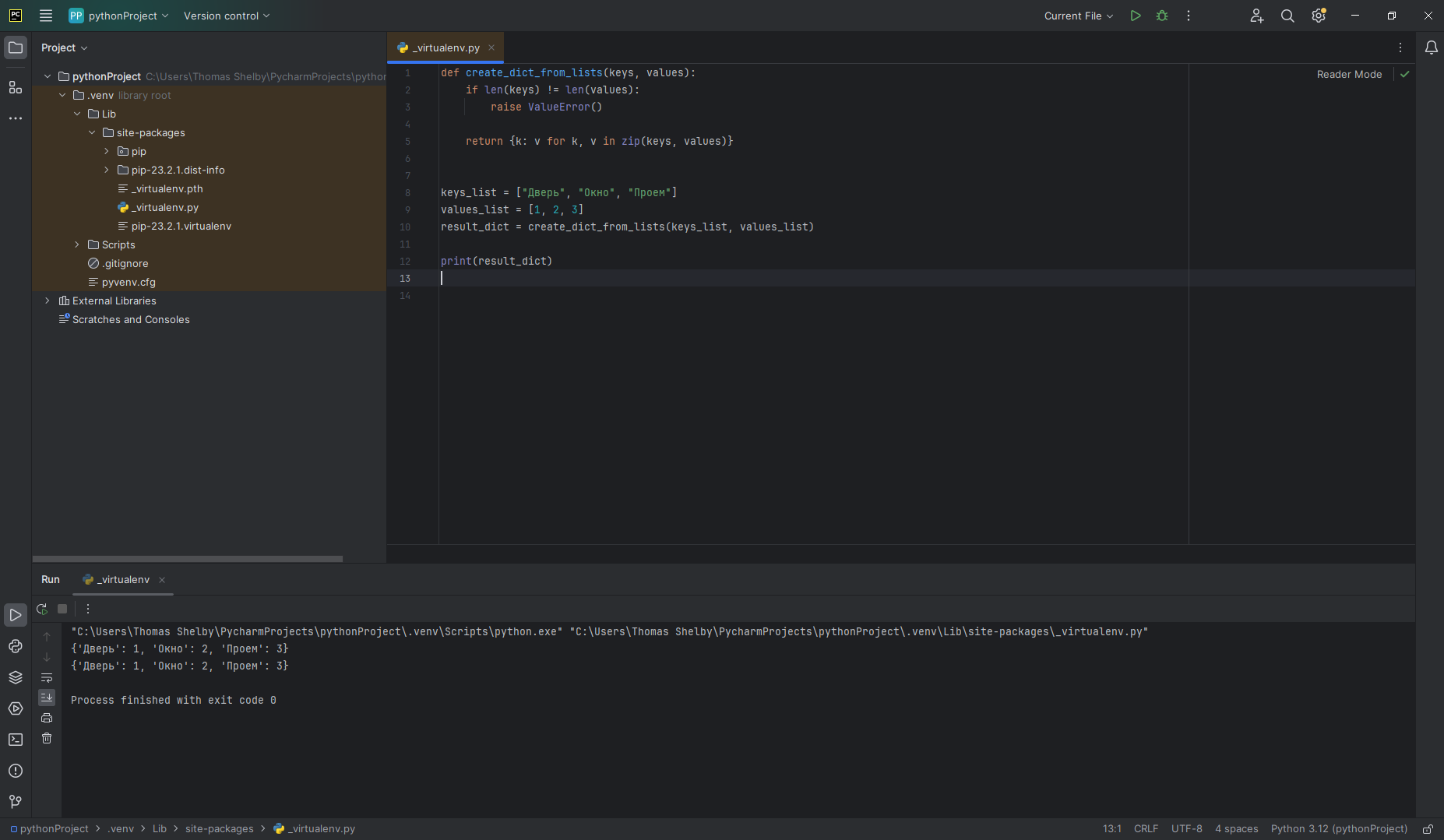


Рис. 1 - результаты работы кода

**Вывод**: в результате работы мы изучили понятия итератора и генератора в Python, а также их преимущества, и ознакомились с примерами их пользования.