**2023/09/30 00:00 |Лекция. Цикл for.**

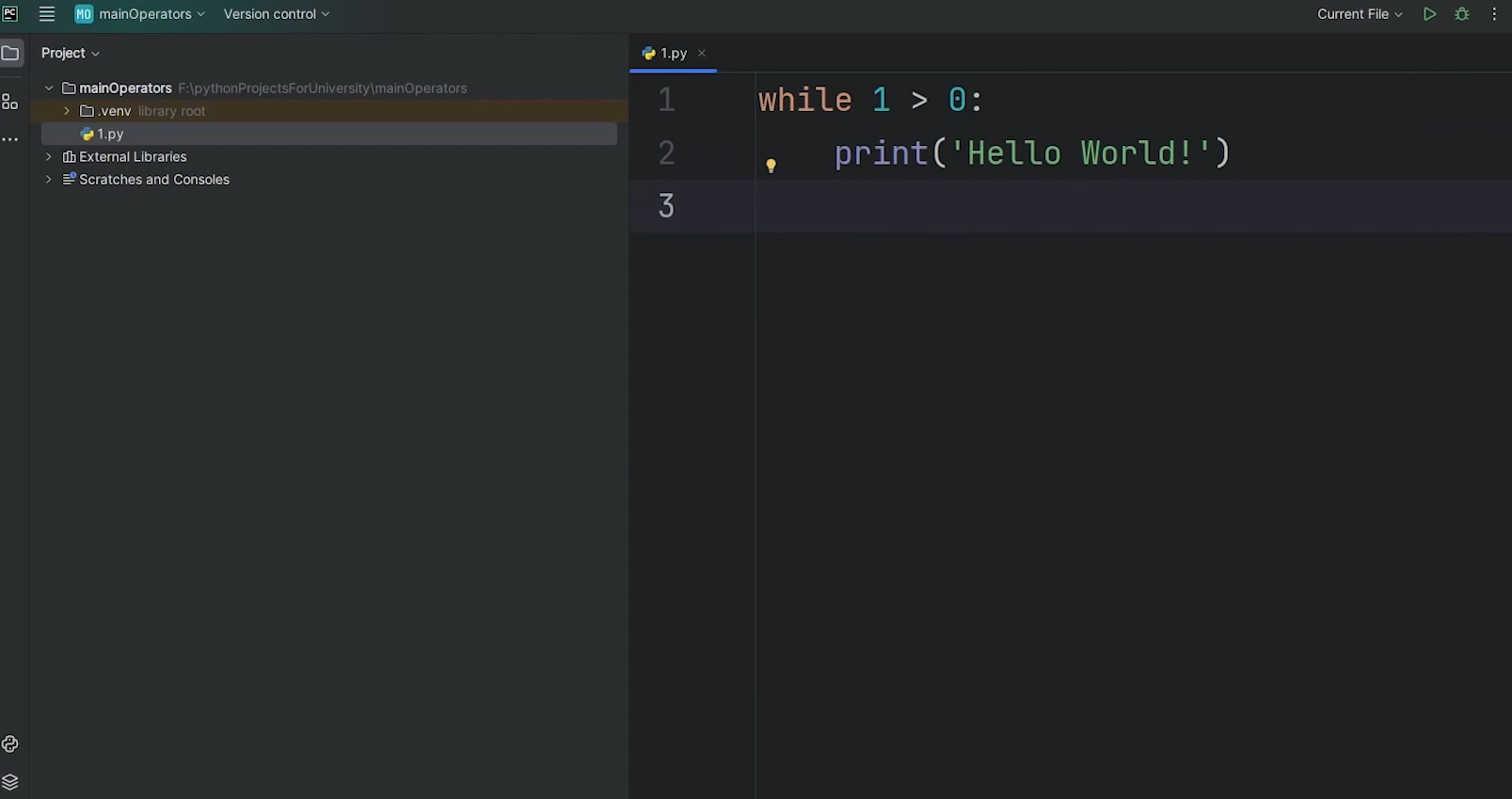
## Цикл for

Сегодня мы поговорим о цикле for, о том, где его можно применять, и как он работает. Давайте скорее приступим!

### Циклы while и for. Как работает цикл for?

Цикл for довольно полезный инструмент, который можно часто увидеть на практике. Мы уже познакомились с циклом while, поэтому понимаем, что циклы нужны, чтобы повторять какие-либо команды.

Как мы помним, цикл while устроен довольно просто, после ключевого слова записываем условие, а затем какие-то команды, выполнение которых будет повторяться, пока ответом на данное условие будет «True» (Рис.1). В нашем случае цикл будет повторяться бесконечно, потому что условие «1 > 0» будет всегда истинно. Поскольку команды выполняются последовательно, компьютер постоянно будет обрабатывать правдивость условия, затем выполнять строчку кода с отступом, прилегающую к этому условию, а потом опять возвращаться к условию и проверять его и так далее, пока условие не потеряет свою правдивость.



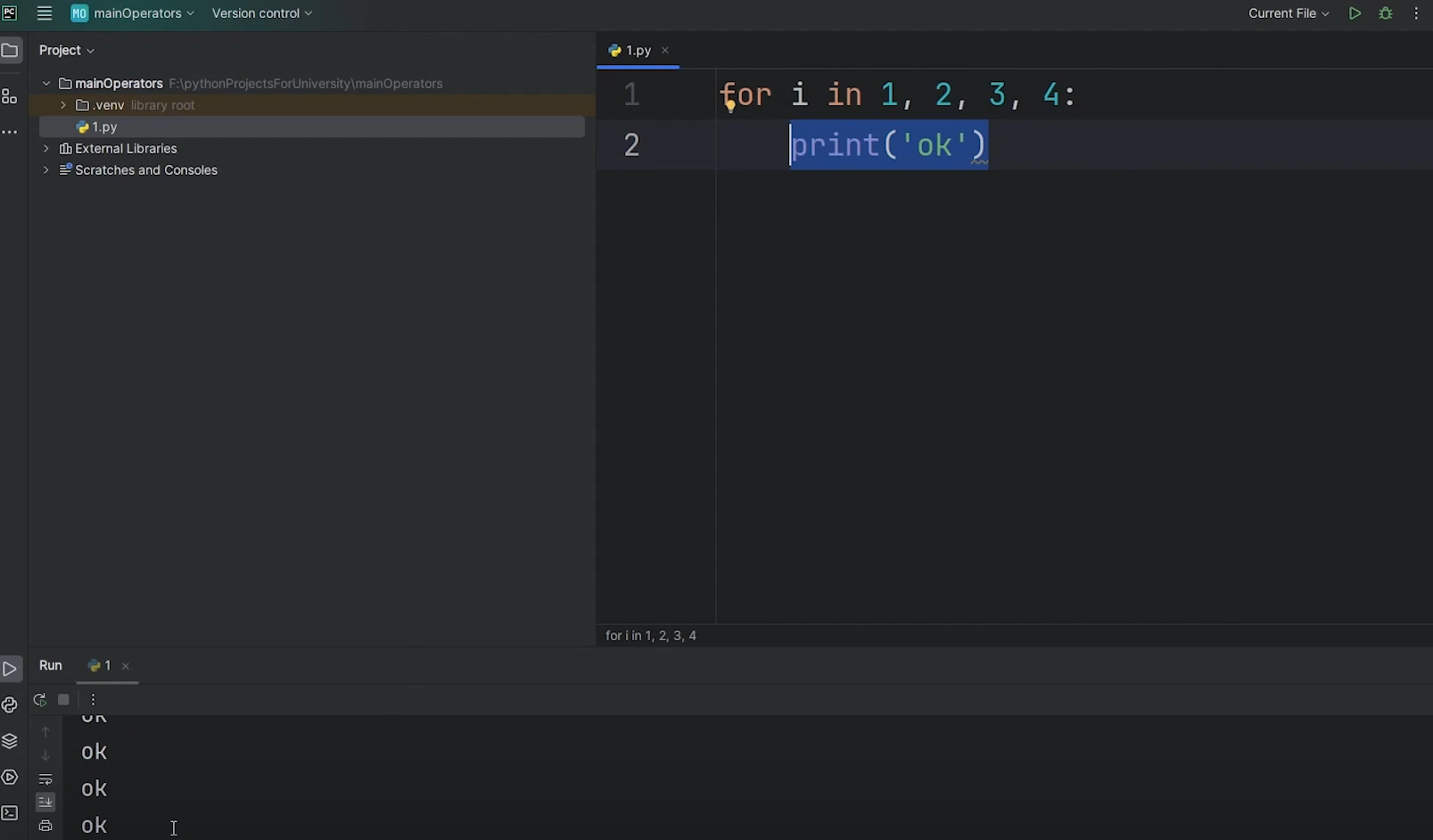
(Рис.1)

Кратко вспомнили, как работает цикл while, а теперь перейдем к нашей главной теме. С циклом for дела обстоят немного интереснее. Он работает в качестве своего рода переборщика. И в случае с циклом for мы сами определяем количество повторений.

Чтобы создать цикл for, пишется ключевое слово «for», а затем идет название переменной, как правило ее называют «i», «j» или «k». Обычно, в случае, когда цикл у нас встречается один раз мы называем ее «i».

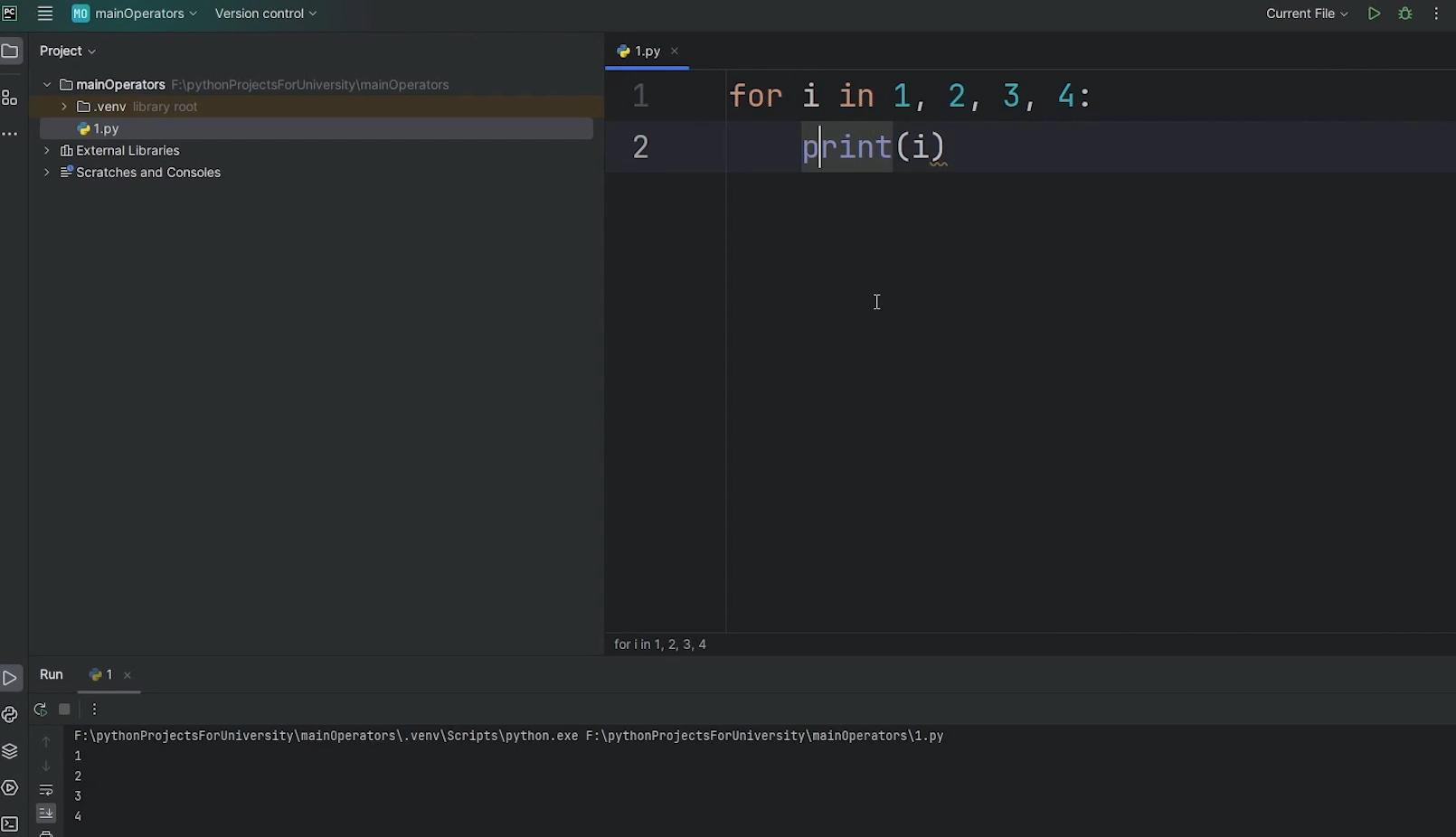
Важно запомнить, что данная переменная существует только в пределах этого цикла. То есть если попытаться где-то в программе обратиться к переменной «i», то выйдет ошибка.

После переменной мы пишем ключевое слово «in», после которого нам нужно передать какую-то последовательность. Для примера зададим какую-нибудь конкретную, допустим последовательность чисел от 1 до 4, и попробуем что-нибудь вывести (Рис.2). Цикл повторит команды столько раз, сколько элементов в последовательности. Таким образом заданная нами команда «print(‘ok’)» повториться 4 раза.



(Рис.2)

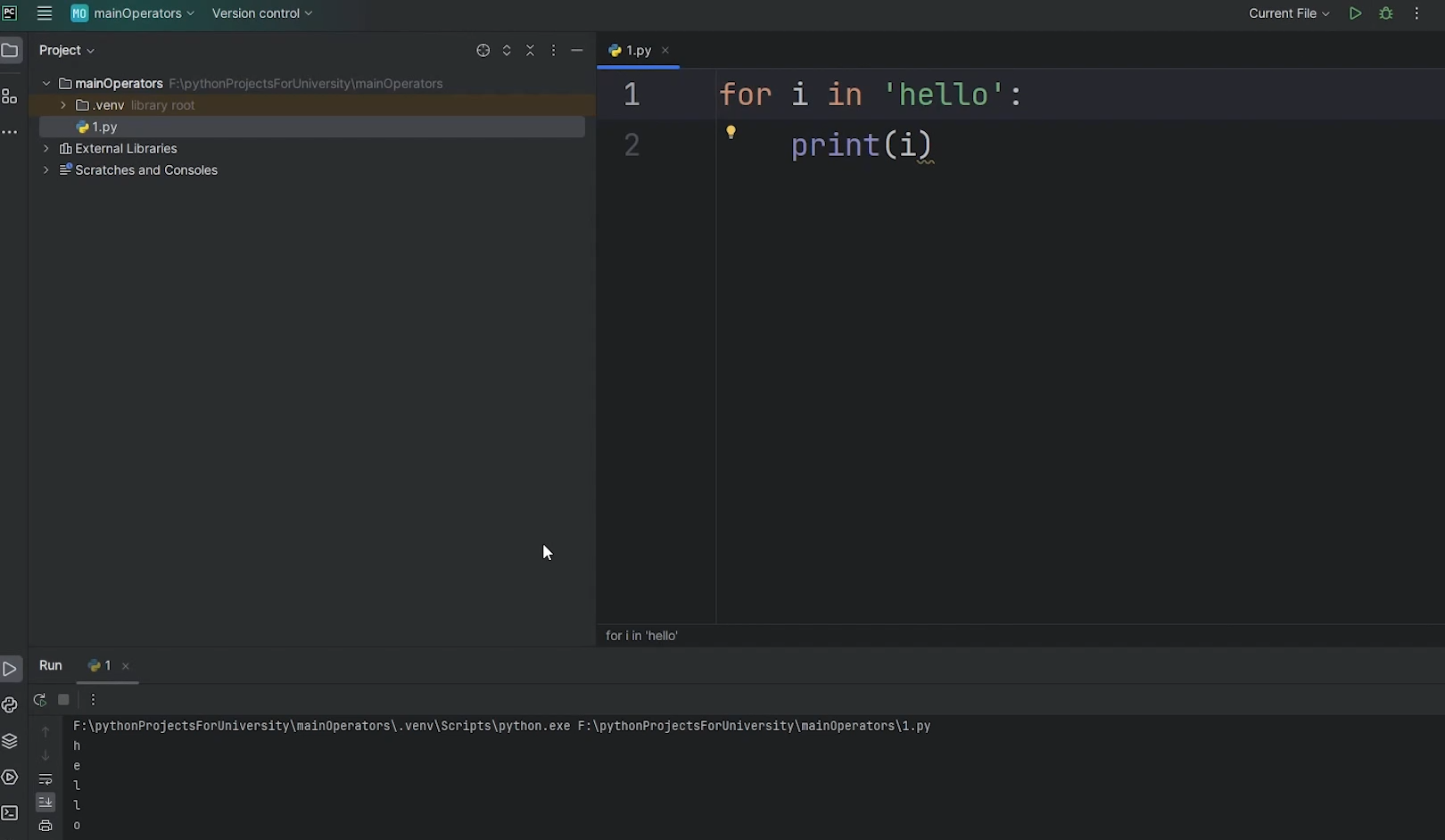
Так для чего же нужна сама переменная? Давайте попробуем ее вывести (Рис.3). Мы увидим такие же значения, как в последовательности, числа от 1 до 4.



(Рис.3)

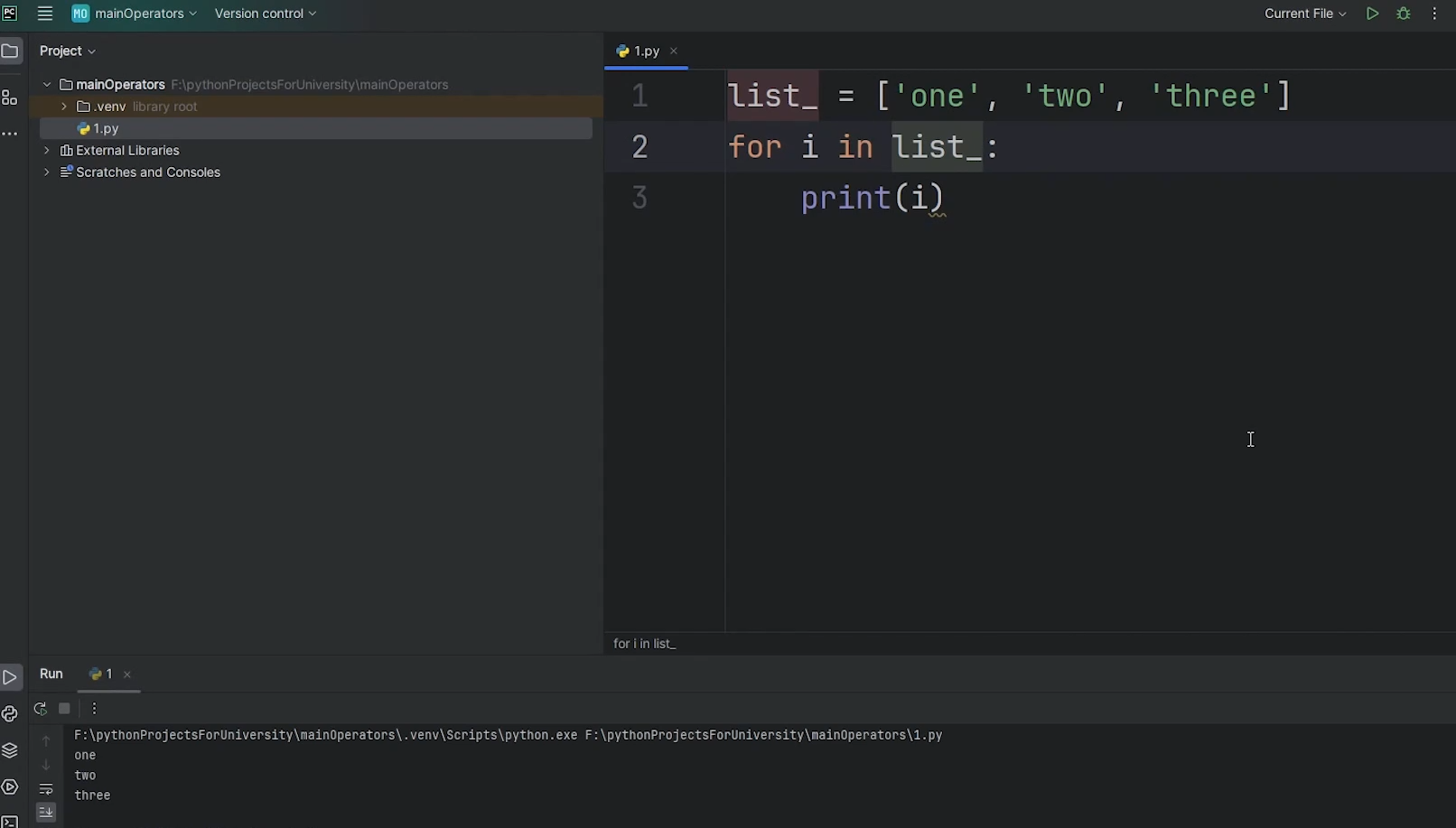
А все дело в том, что переборщиком мы назвали данный цикл не просто так. Переменная «i» на каждом повторении цикла сохраняет в себя значения из последовательности. То есть в нашем случае на первом повторении она сохранила в себя значение «1», на втором «2» и т. д. пока последовательность не закончилась.

С помощью этой переменной мы можем перебрать многое. Например, посмотрим, как это работает со строкой. Строка тоже является последовательностью символов, соответственно, если мы введем строку «‘hello’», то увидим каждую букву строки поочередно (Рис.4).



(Рис.4)

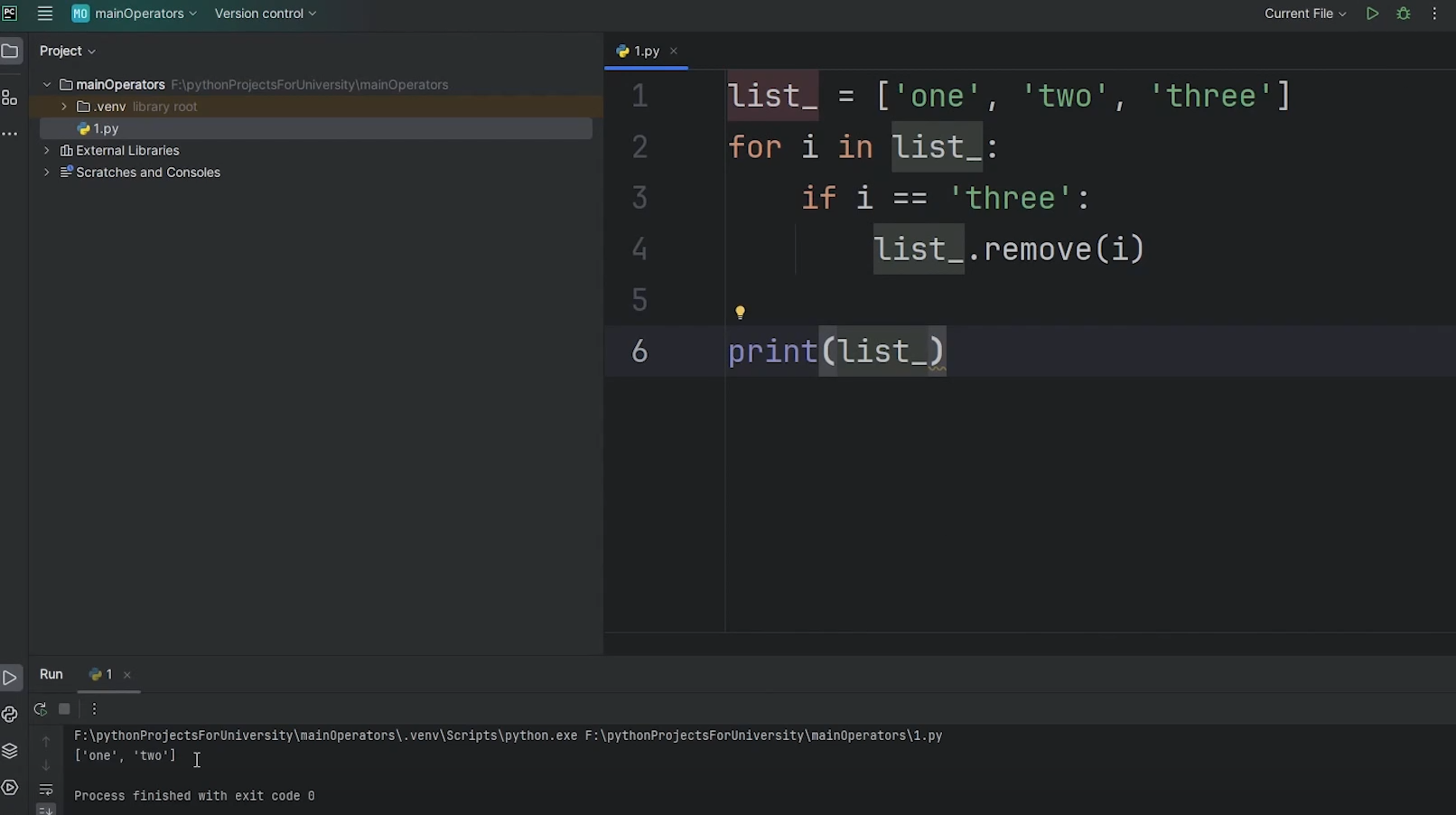
Давайте создадим переменную «list\_» и поместим туда список. А в списке у нас будут три строки, например «‘one’», «‘two’» и «‘three’», вставим этот список в наш цикл (Рис.5). После запуска цикла видим наши строки поочередно. Таким образом, после слова «in» мы можем ставить в том числе и переменные, которые хранят в себе некие последовательности, иными словами, итерируемые.



(Рис.5)

### Цикл for на практике

Например, нам нужно пройтись по списку с какими-то данными и что-то сделать с его элементами. Используем список из предыдущего примера, допустим строка «‘three’» у нас лишняя и нам нужно от нее избавиться. Цикл будет перебирать значения внутри нашего списка. Поэтому запишем такое условие с оператором if, что, если «i» будет равна «‘three’», то этот элемент будет удален из списка с помощью команды **remove**, а затем наш список будет выведен на экран. Выглядеть это будет так (Рис.6). Как видим, строки «‘three’» в ответе нет.



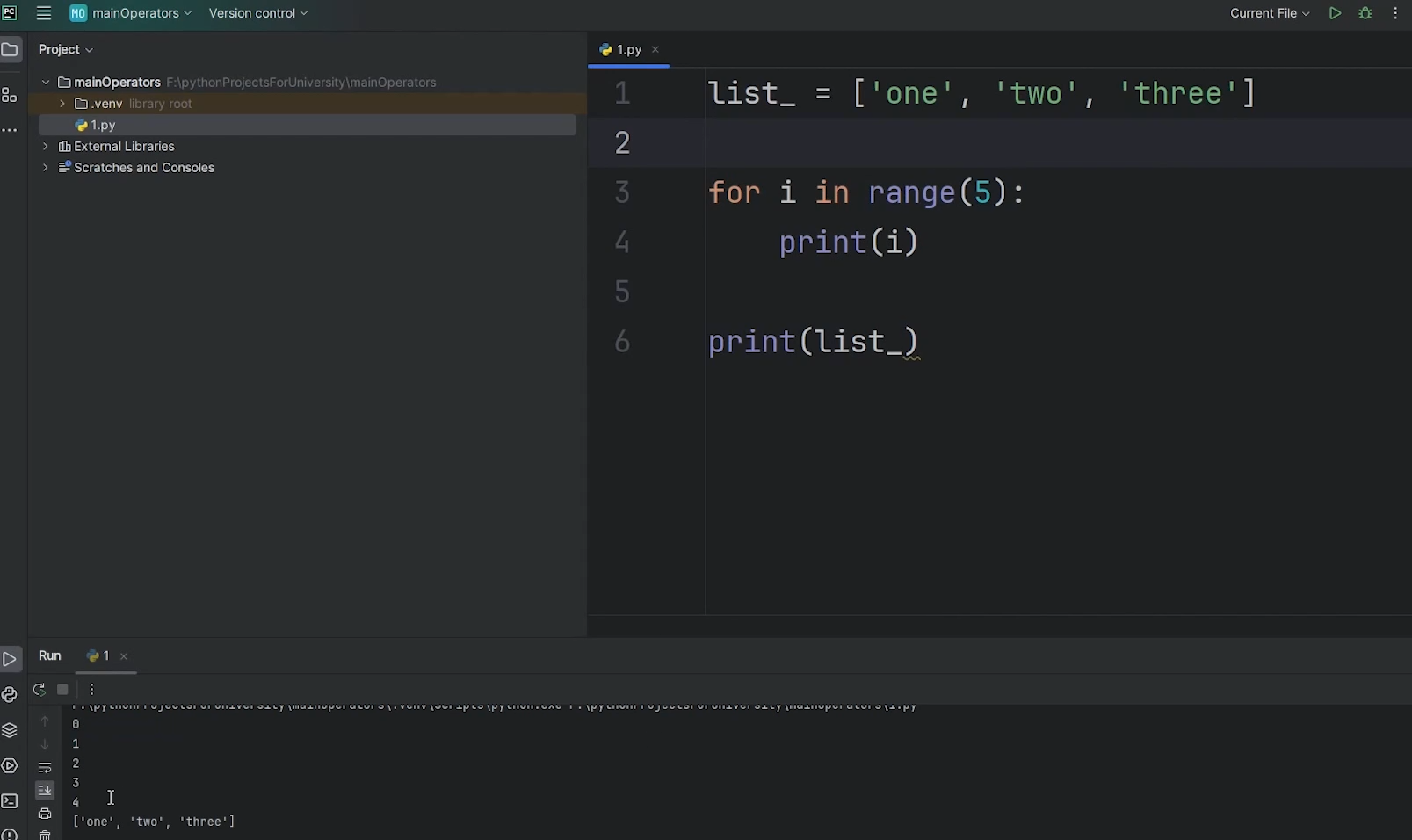
(Рис.6)

Существует еще один вариант работы в цикле for со списком. Сейчас переменная «i» просто сохраняет в себя значения из списка, вернее сказать, она создает в себе подобные элементам из списка значения, но как таковая ими не является.

Другое дело, если мы захотим пройтись по списку и изменить каждый его элемент. Цикл for может помочь нам и здесь. Но перебирать нам будет нужно уже не сам список, а числовую последовательность. Так же, как и у строк, в списках у каждого элемента есть свой порядковый номер, то есть индекс. Индексация, как мы помним, начинается с нуля.

Соответственно, нам нужна числовая последовательность, которая позволит получить индексы, с их помощью мы сможем доставать элементы прямо из списка.

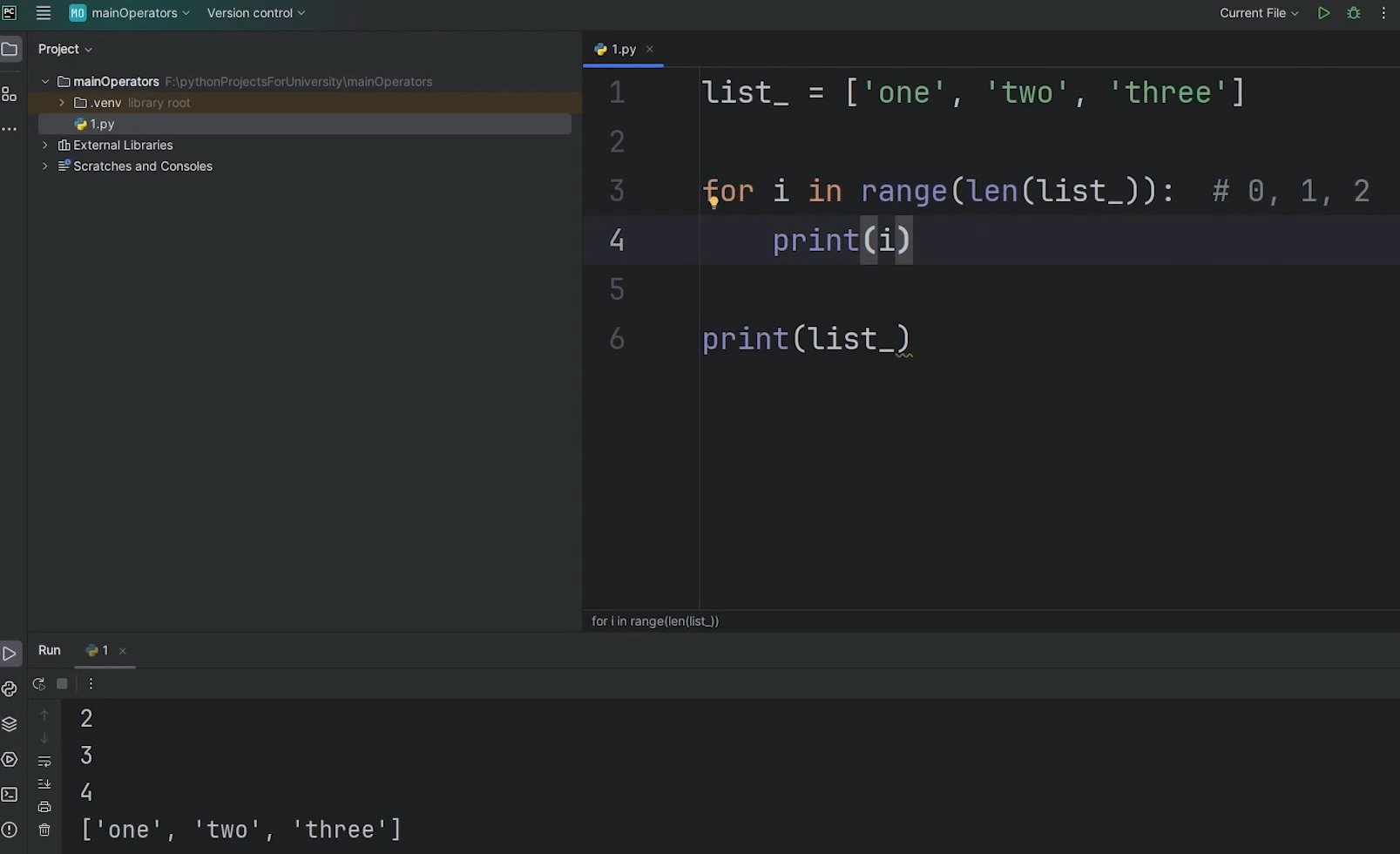
Чтобы это сделать, мы воспользуемся функцией **range**. Она возвращает нам последовательность чисел от 0 до числа, которое мы укажем в скобках. Например, если мы впишем в скобки число 5, то наш цикл повторится 5 раз. Если мы захотим, вывести переменную «i», мы увидим значения от 0 до 4 (Рис.7). Это немного напоминает срез, поскольку здесь последний элемент также не входит в последовательность. Но по факту цикл повторится 5 раз, потому что здесь учитывается 0.



(Рис.7)

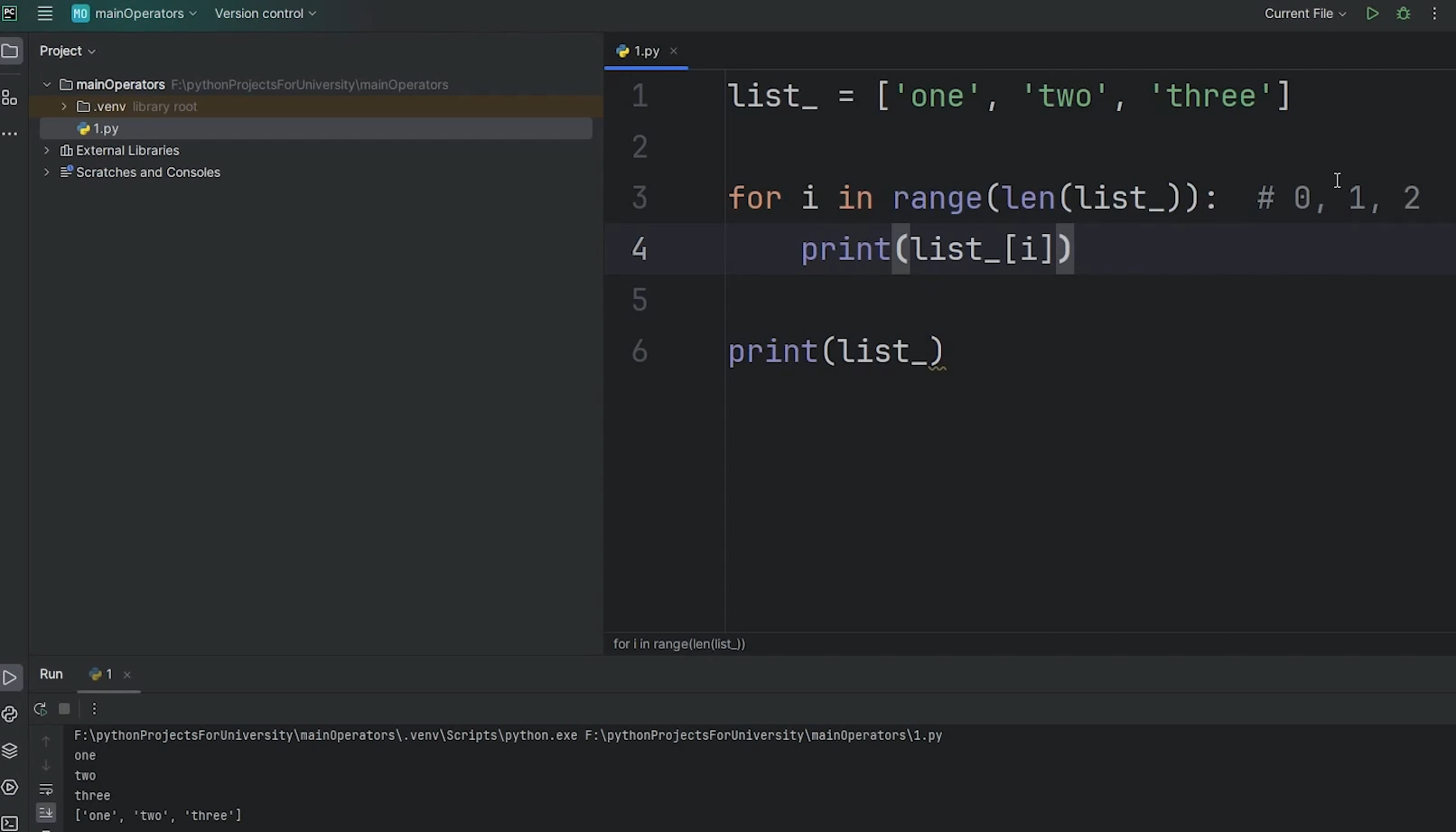
Уже сейчас эта последовательность похожа на нужную нам индексацию, то есть, если мы, например, подставим на место индекса число 0, то сможем достать элемент «‘one’». Однако здесь есть один нюанс. В этой последовательности у нас сейчас больше чисел чем элементов в списке, поэтому на каком-то этапе мы просто наткнемся на ошибку.

Чтобы этого избежать и немного оптимизировать нашу программу под любой размер списка, мы можем написать в скобках вместо числа функцию **len**. Она возвращает нам длину элемента, который мы запишем в прилегающие к ней скобки. Мы впишем туда нашу переменную «list\_» (Рис.8). Таким образом функция len получит от нашей переменной число 3, поскольку в списке 3 элемента, и передаст его функции range, которая выдаст нам последовательность от 0 до 2, что как раз и соответствует индексам списка.



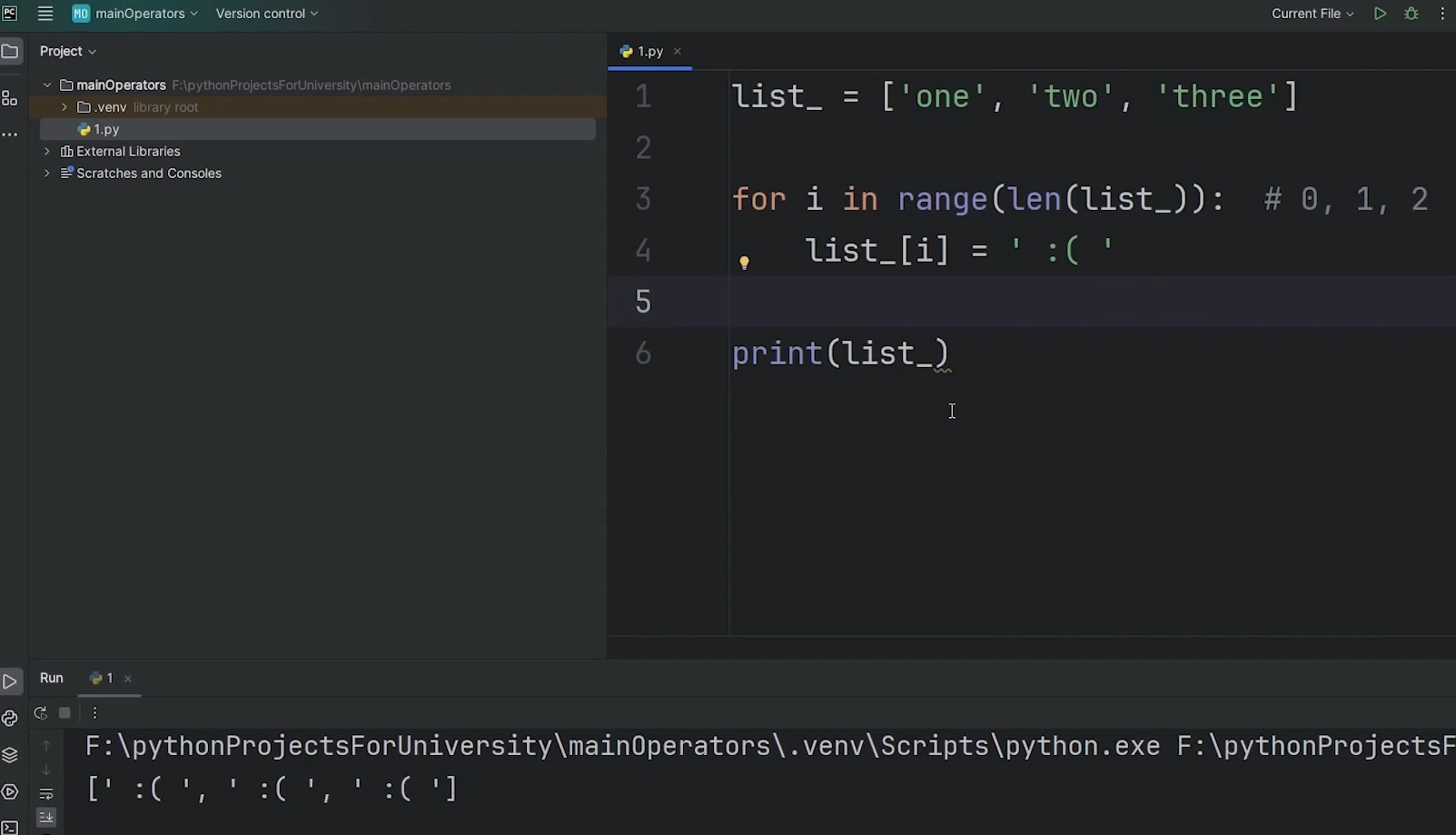
(Рис.8)

Теперь можем достать элементы, как мы помним, нужно вписать в квадратные скобки нужный индекс, но мы не будем писать какой-то конкретный. Впишем туда нашу переменную «i» и увидим, как нам выдало каждый элемент последовательно (Рис.9).



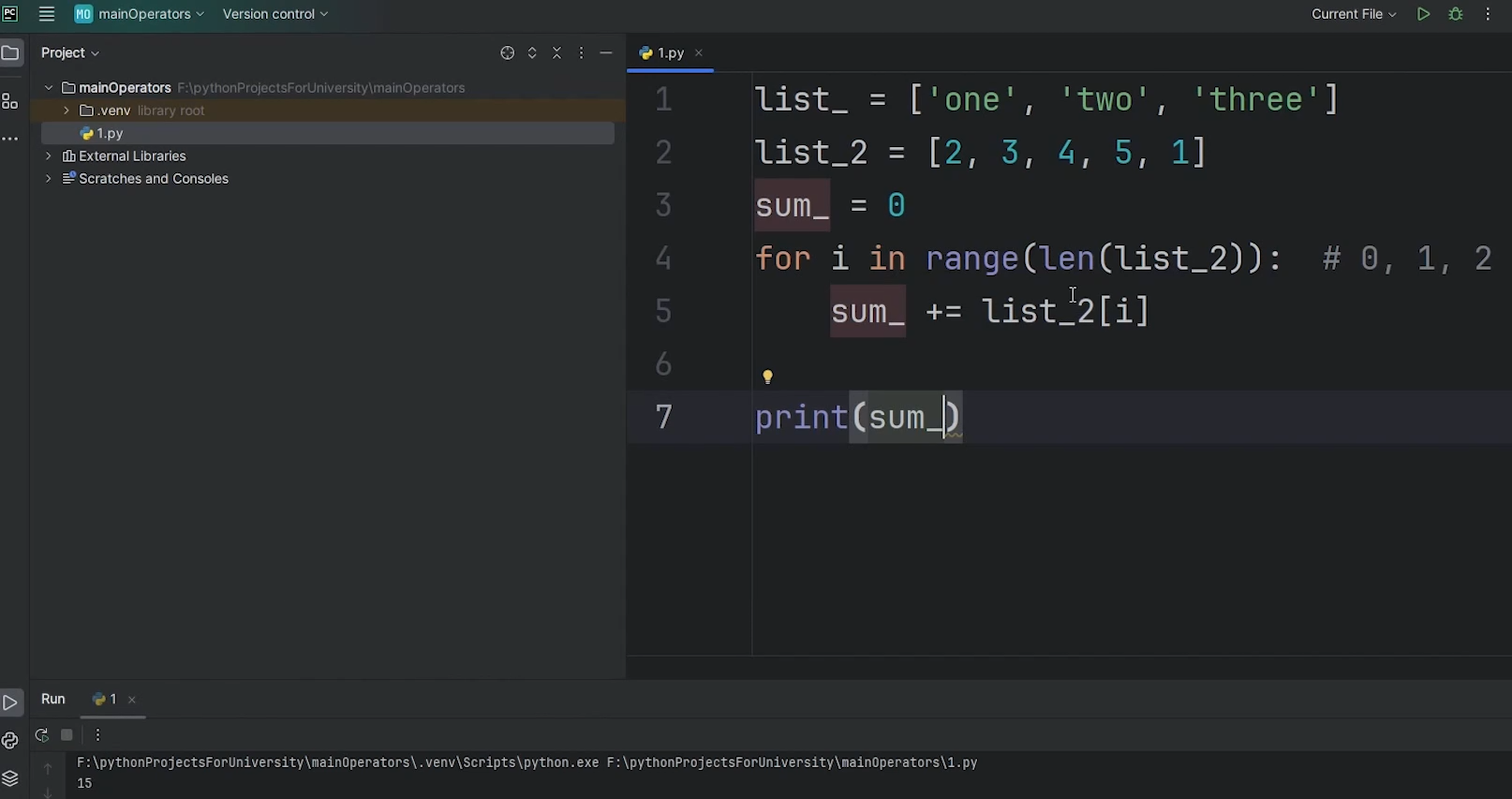
(Рис.9)

Таким образом мы можем доставать элементы из списка и работать непосредственно с ними. Например, заменим элементы нашего списка на грустный смайлик и увидим, что все три элемента теперь смайлики (Рис.10).



(Рис.10)

А теперь рассмотрим пример со сложением элементов в списке. Создадим второй список «list\_2» и добавим в него какие-нибудь числа. Нашей задачей будет посчитать сумму всех его элементов. Здесь нам потребуется переменная, которая обязательно будет существовать вне цикла, например «sum\_», которая равна 0. Если мы запишем ее в цикле, то на каждом повторении она всегда будет перезаписываться. Теперь мы можем брать элементы из нашего списка и прибавлять их к нашей переменной. Запустим программу и получим 15 (Рис.11).

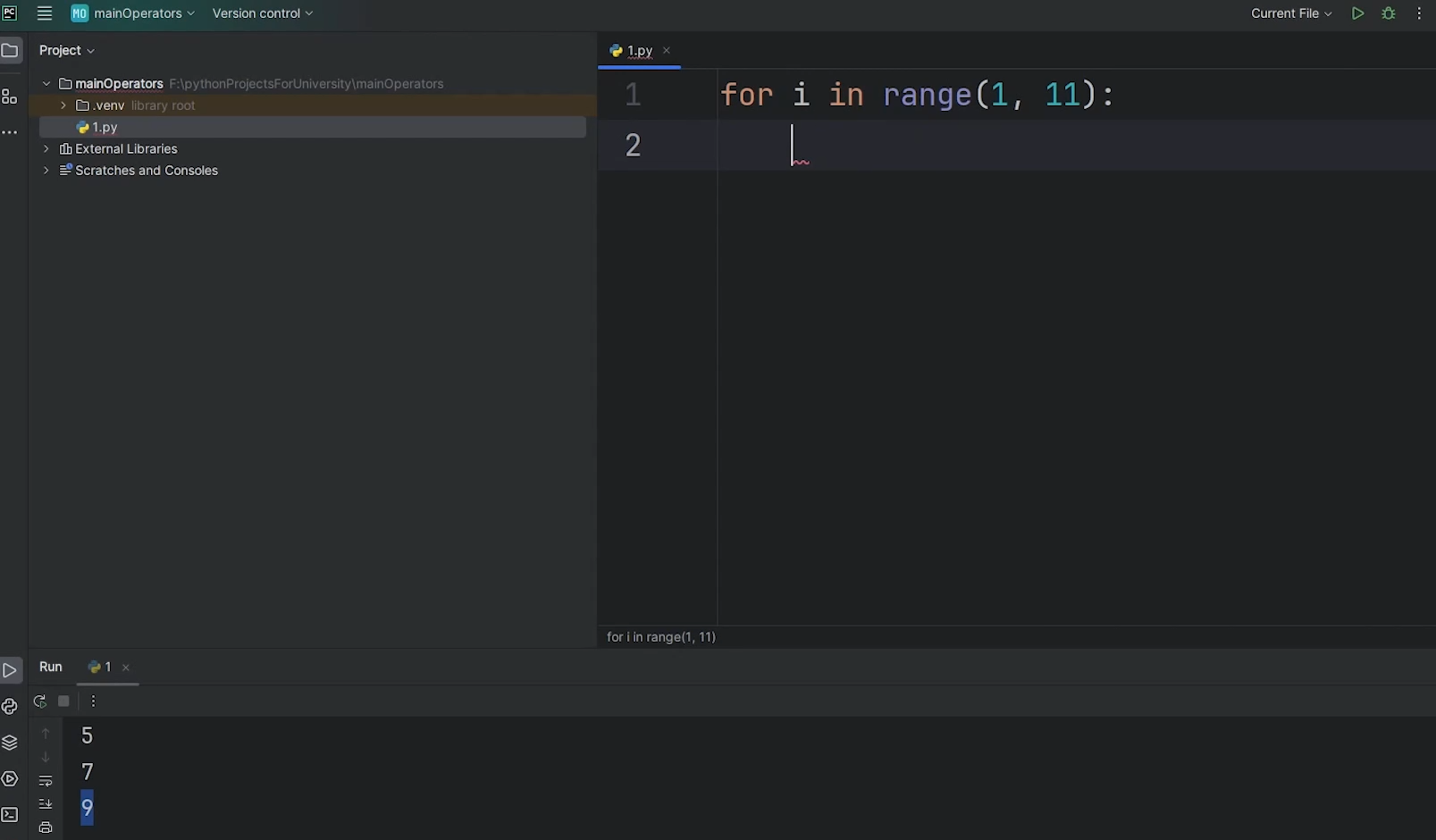


(Рис.11)

### Вложенный цикл

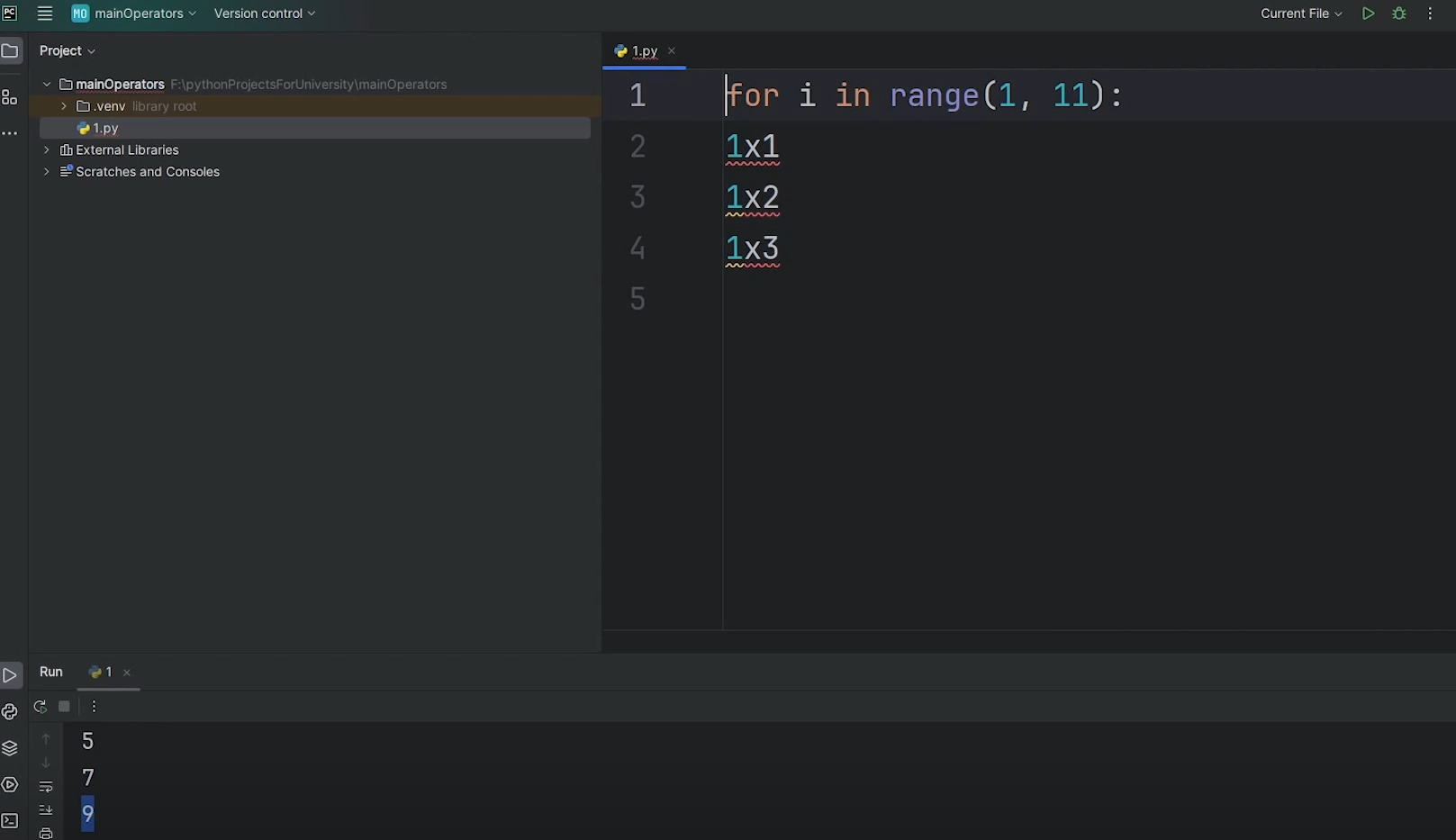
Бывает такое, что есть второй, или же **вложенный цикл**. Рассмотрим наглядно, попробуем с помощью цикла for вывести таблицу умножения.

Для начала создадим цикл, здесь нам также будет нужна команда range, сразу зададим, что наша таблица умножения будет от 1 до 11, т. к. мы помним, что последний элемент не включается. Единицу мы вписали, чтобы наша таблица начиналась не с 0, а с 1 (Рис.12).



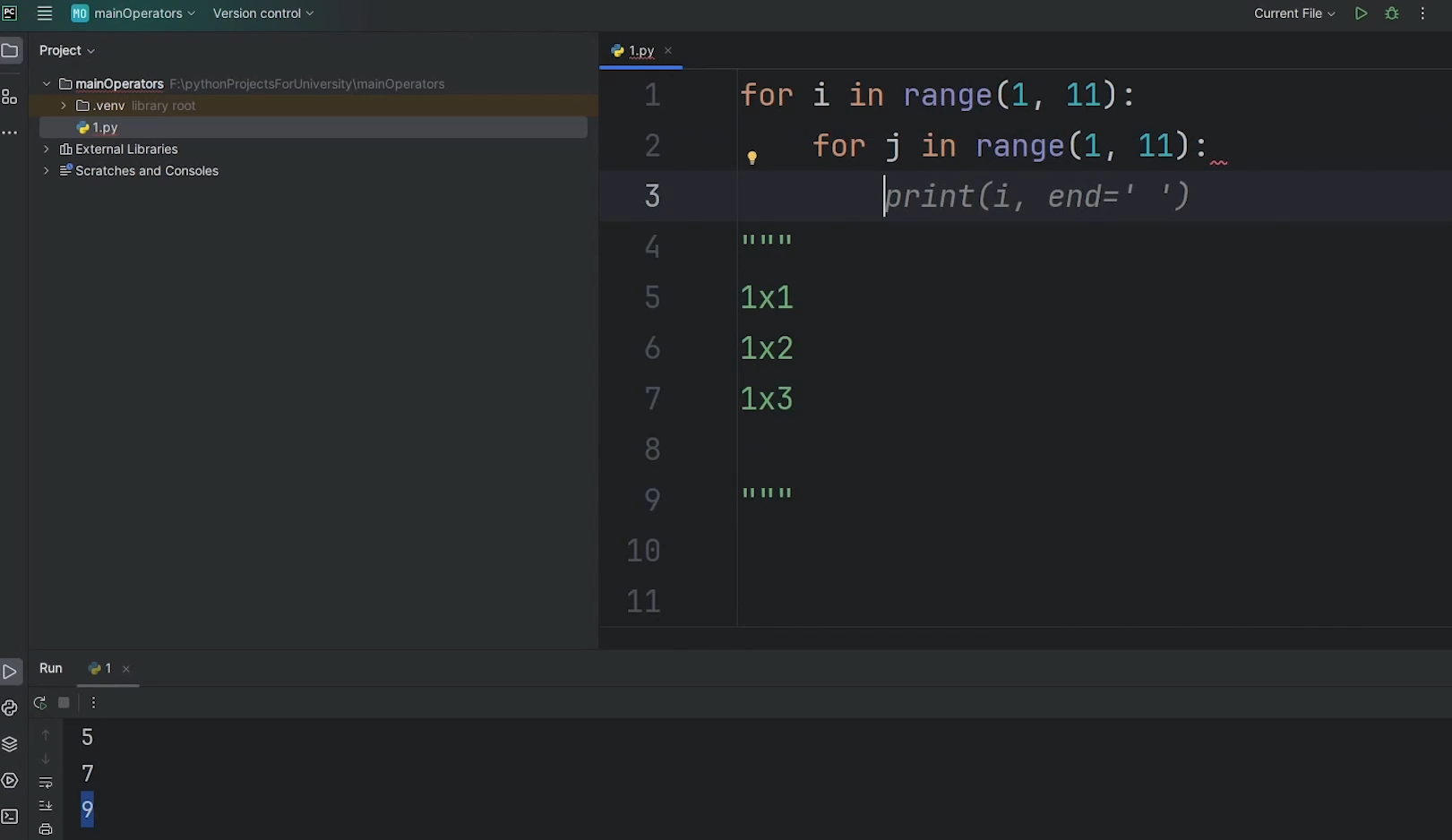
(Рис.12)

Функция range может принимать до 3 значений. Первое – начало отсчета и второе – конец, как и в нашем случае, но существует еще третье – это шаг последовательности. Допустим, если мы укажем шаг два, то нам будет выводиться каждое второе число последовательности.

Итак, в цикле for переменная «i» постоянно меняет значения, но что делать, если нам нужна таблица умножения такого формата? (Рис.13) 

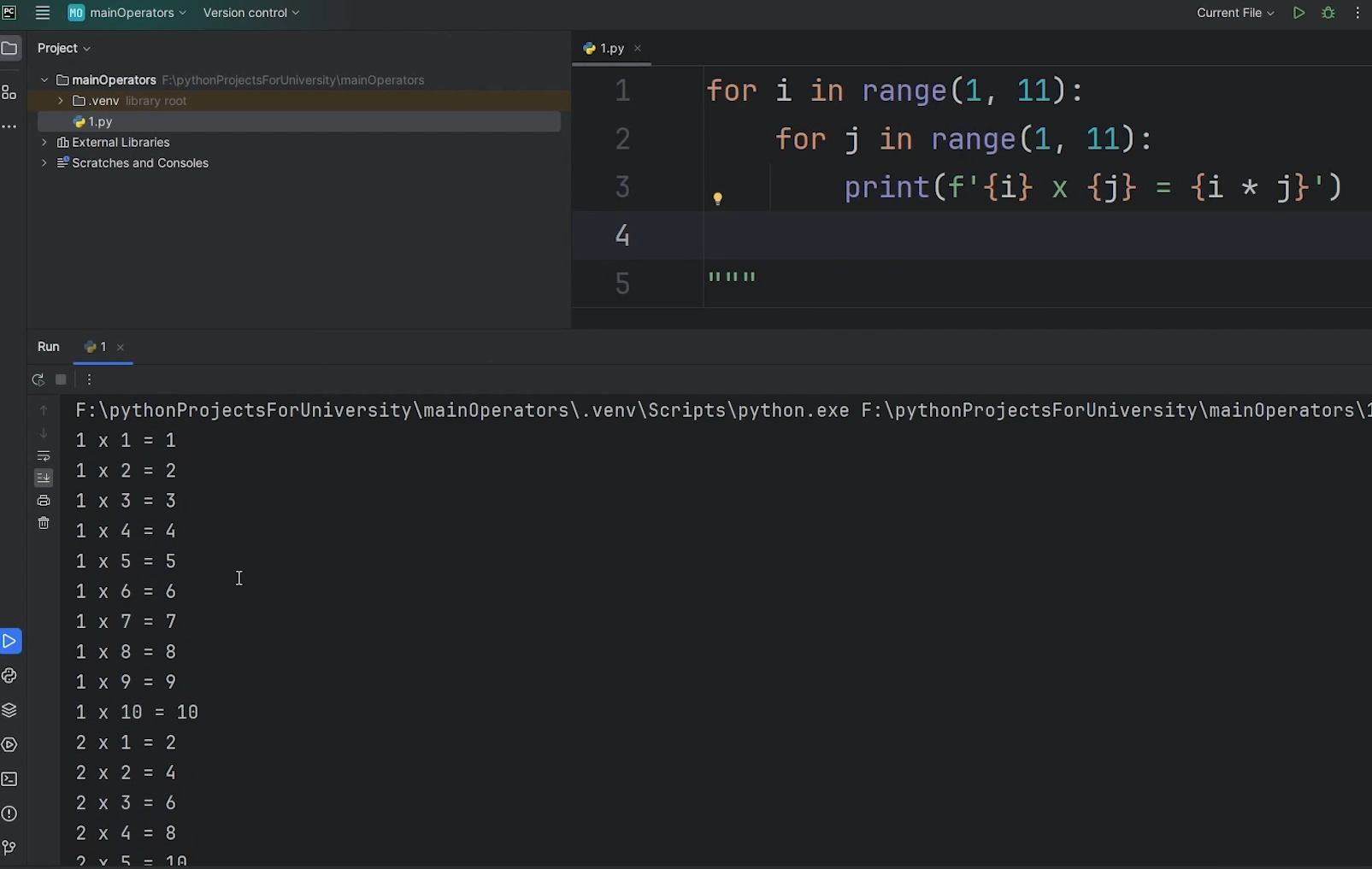
(Рис.13)

Но еще и так, чтобы таблица была для всех чисел от 1 до 9, а не только для единиц. То есть со временем у нас будут меняться значение и слева и справа. Это значит, что нам нужны две постоянно меняющиеся переменные. В нашем цикле одна уже есть – «i», но вот тут нам и понадобится второй цикл со второй переменной «j» (Рис.14). Вот так у нас и получилось два цикла, один внутри другого, то есть вложенный цикл.



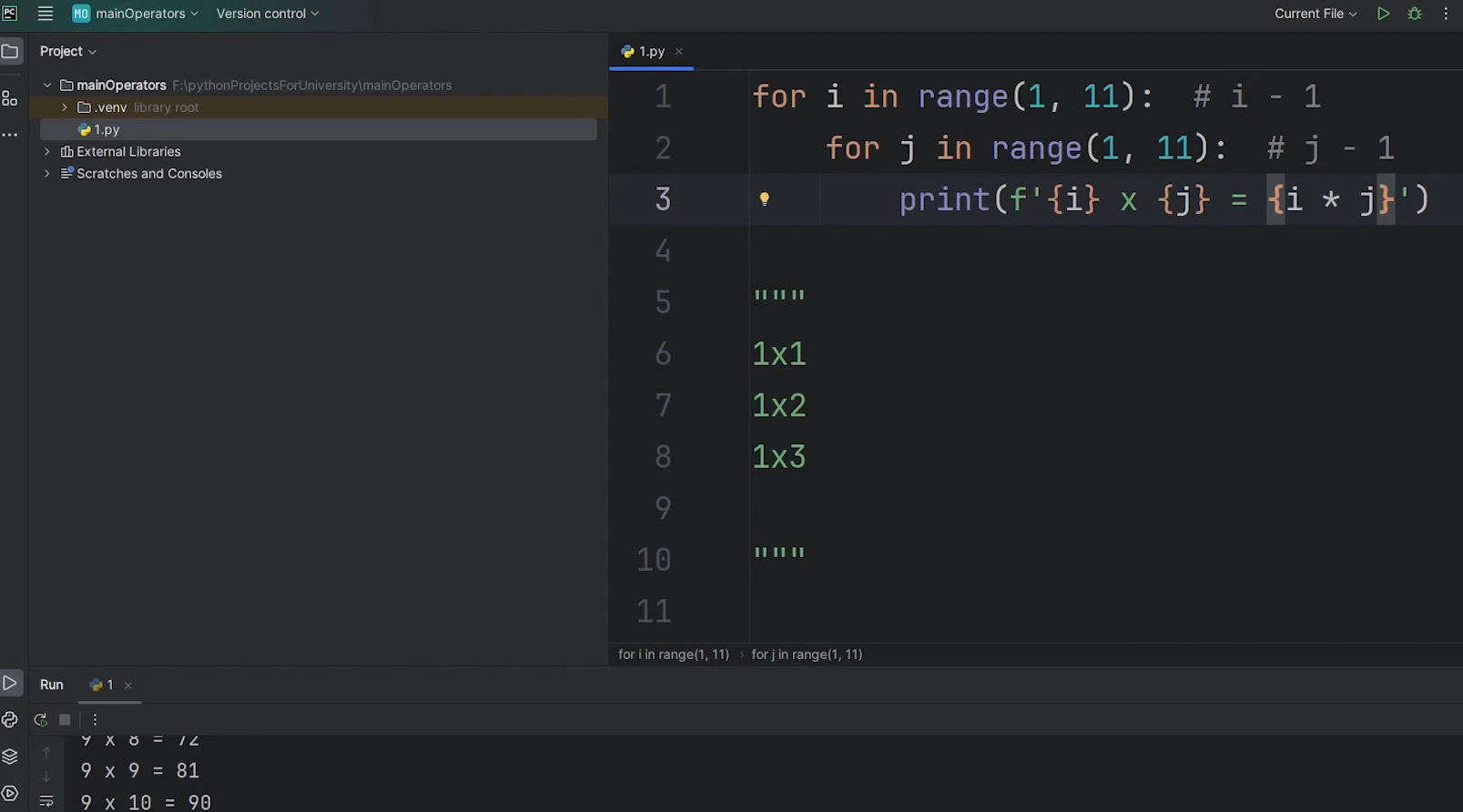
(Рис.14)

Пропишем «print(f ‘{i} x {j} = {i} \* {j}’)», а теперь выведем результат (Рис.15). У нас получилась полноценная таблица умножения.



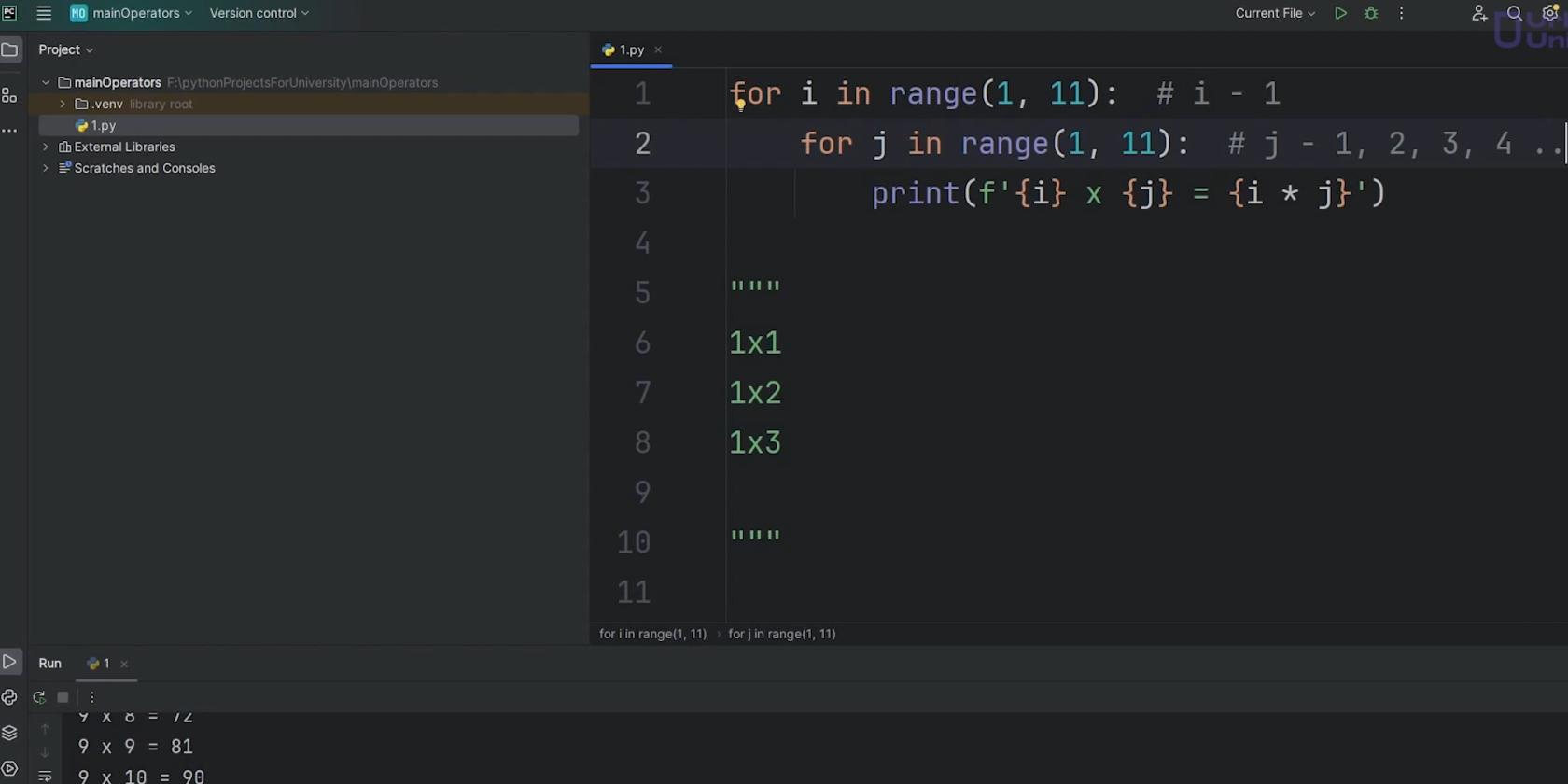
(Рис.15)

А теперь разберемся как это работает. Сначала выполняется первая строчка и создается цикл, а переменная «i» записывает в себя первое значение – 1. Далее создается еще один цикл уже с переменной «j» с таким же первым значением – 1. Потом идет команда print, в выражение подставляются первые значения (Рис.16).



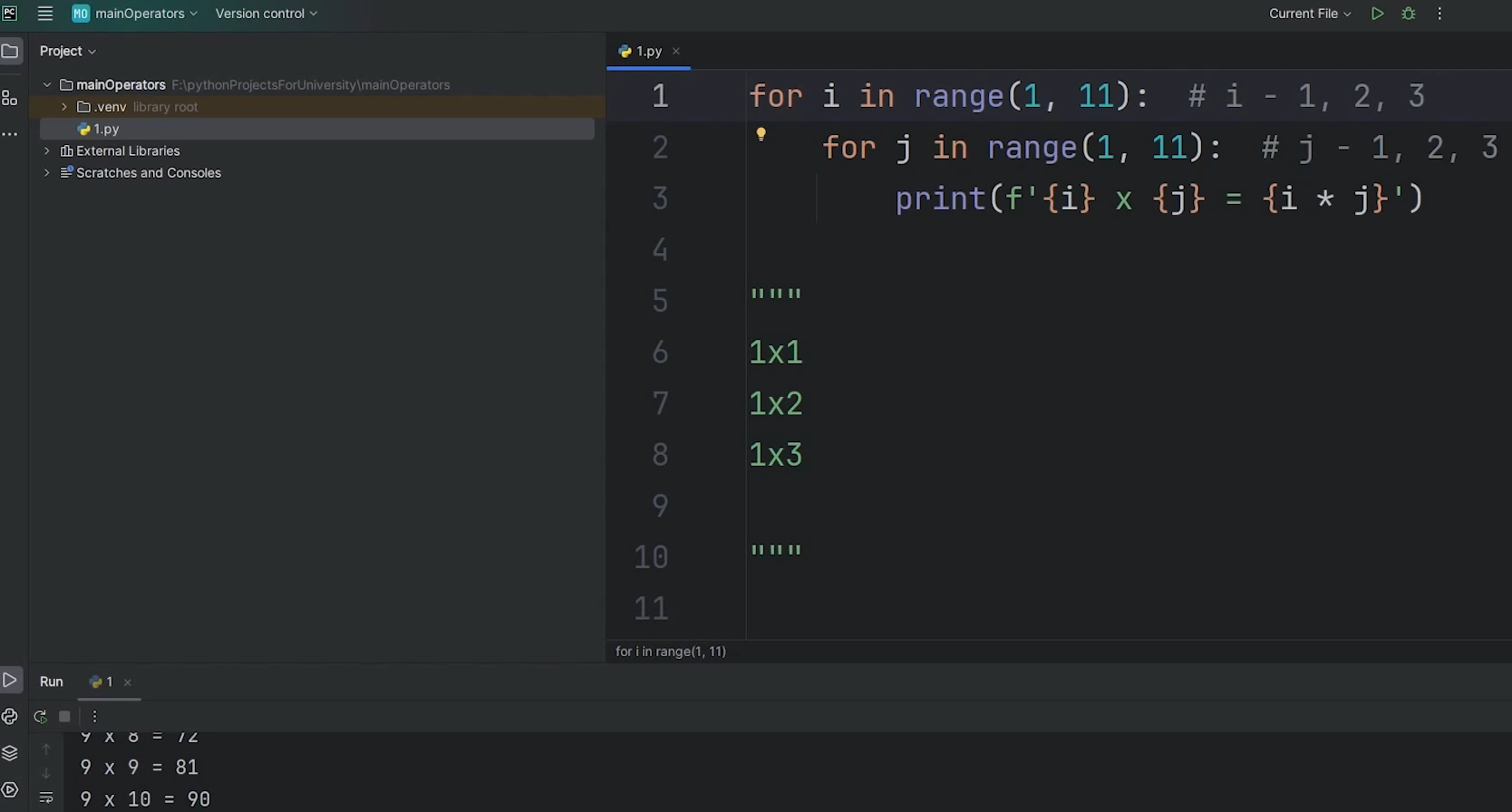
(Рис.16)

И компьютер пошел бы дальше, но наша вторая последовательность, к которой прилегает выражение с умножением, еще не была перебрана переменной «j», а значит он не может просто двигаться дальше. Компьютер продолжает придавать переменной новые значения из последовательности, то есть «j» становится равна 2 и т. д. (Рис.17).



(Рис.17)

После того, как все элементы были перебраны «j» компьютер снова хочет двинуться дальше к следующей команде, но теперь уже во внешнем цикле for переменная «i» еще не перебрала все элементы. Соответственно она меняет свое значение на 2, и снова создается вложенный цикл for, где «j» опять равен 1 и т. д. (Рис.18).

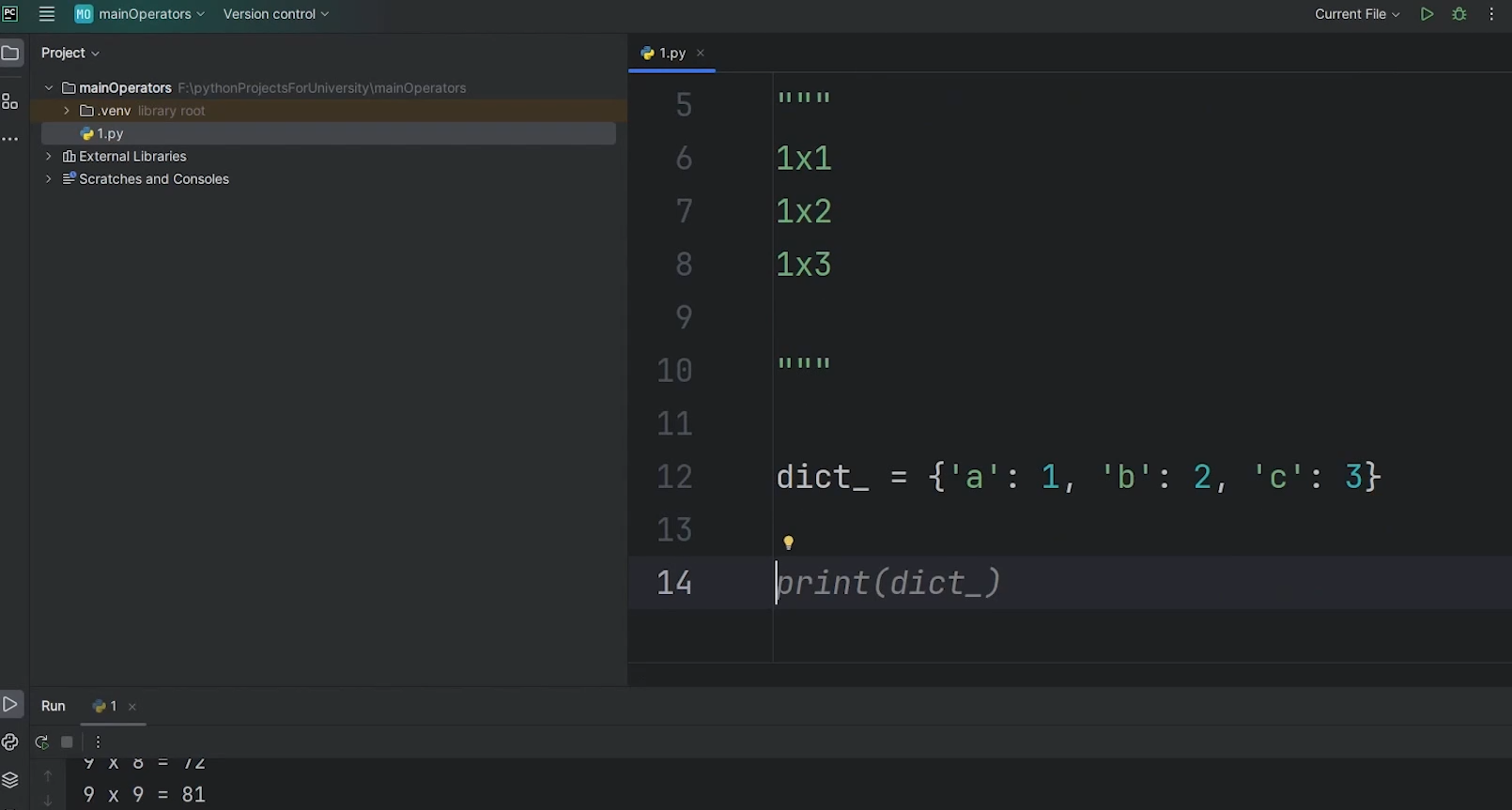


(Рис.18)

Так будет повторяться до тех пор, пока в цикле for не произойдет 10 повторений, на каждом из которых будет по 10 раз повторяться вложенный цикл for.

### Словари в цикле for

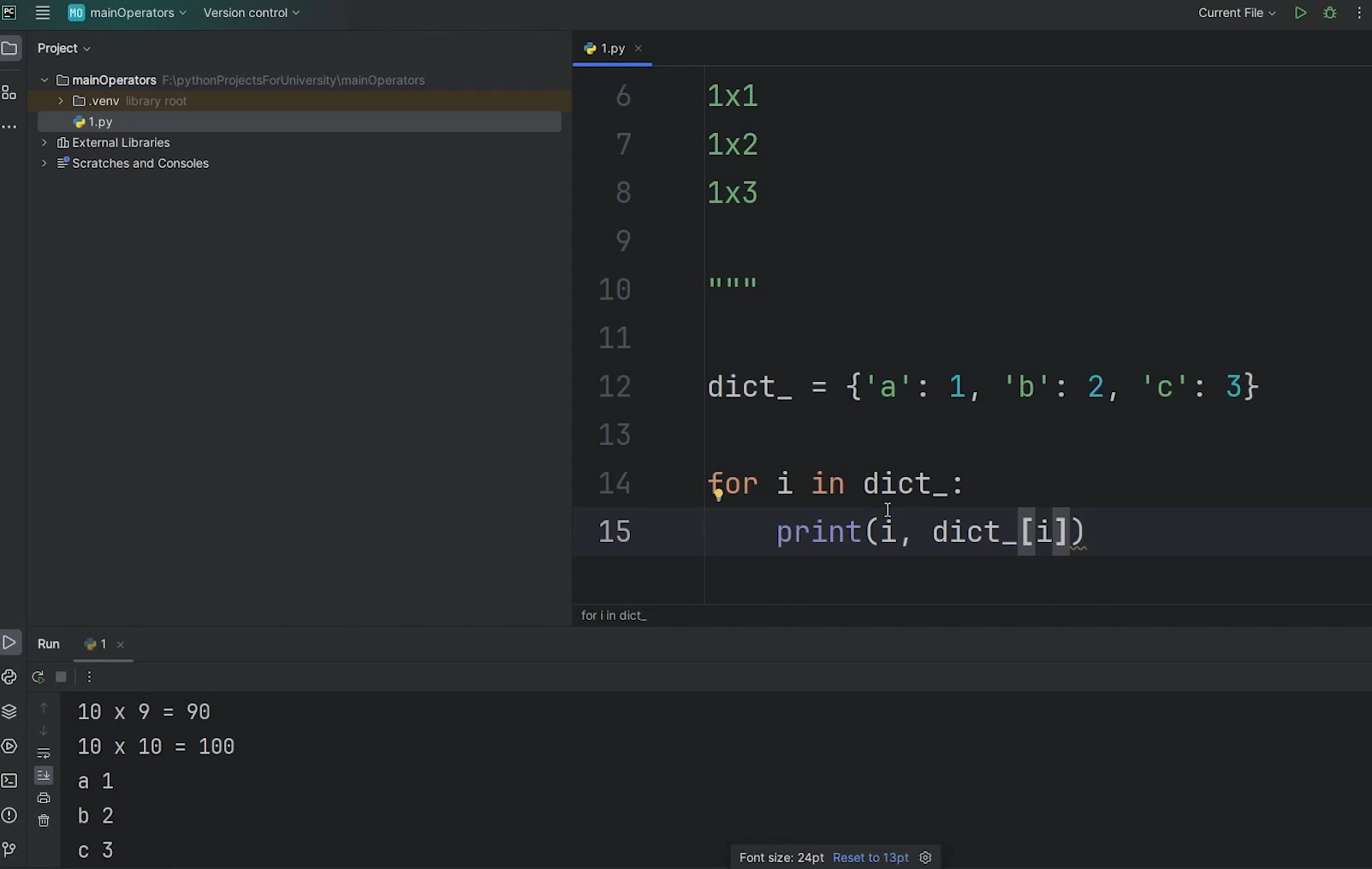
Цикл for может перебирать и словари. Для примера создадим какой-нибудь словарь с тремя значениями и ключами (Рис.19).



(Рис.19)

Словарь тоже является итерируемым объектом, однако он не упорядочен и элементы при переборе не всегда будут расположены так, как мы этого хотим.

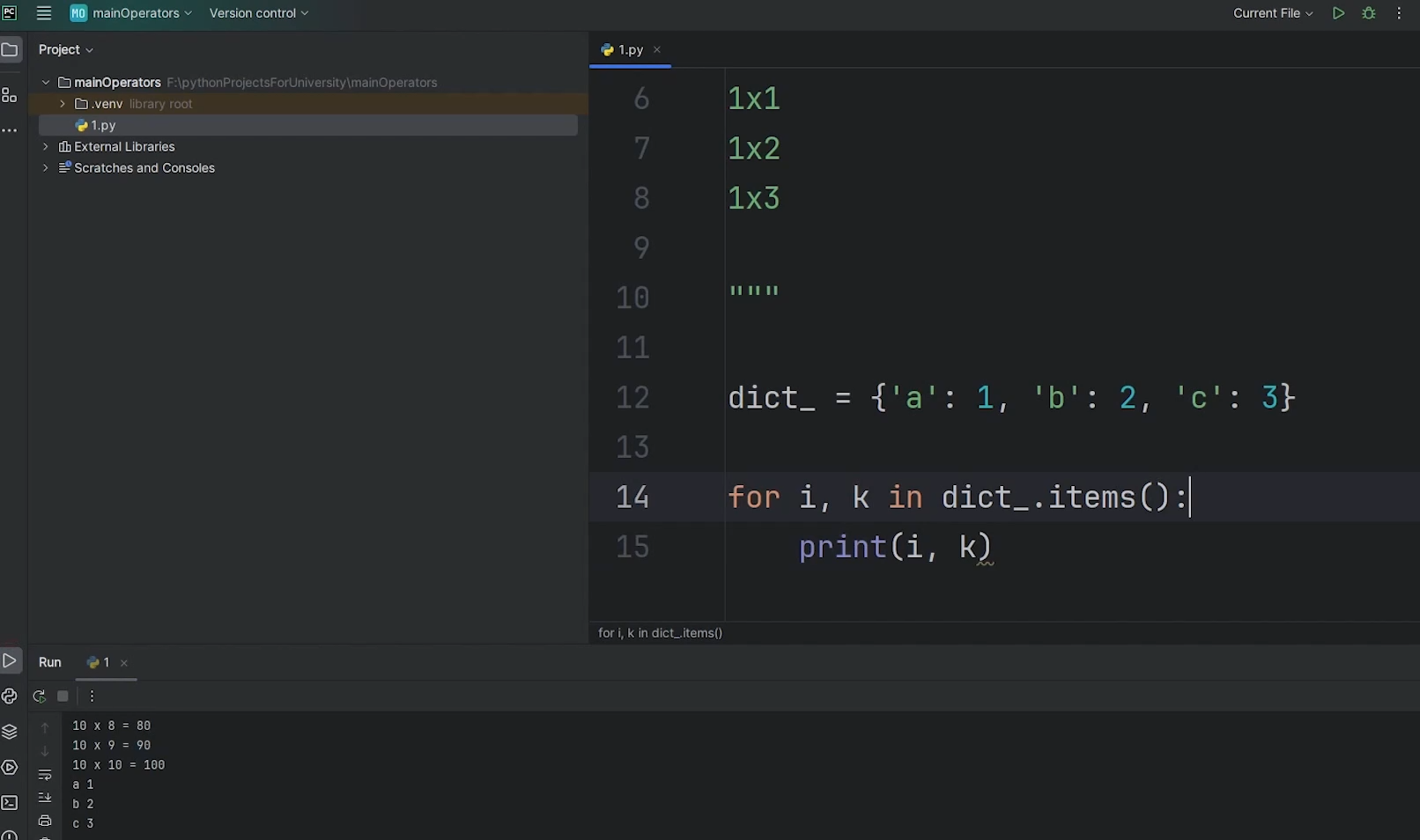
Давайте тоже по нему пройдемся, будем выводить «i» и элемент нашего словаря «dict\_»по ее значению. Мы получим такой результат (Рис.20).



(Рис.20)

Здесь по уже известной нам аналогии мы проходимся по элементам словаря, а переменная «i» вбирает в себя значения ключей, мы можем использовать это, чтобы, например, достать само значение по указанному ключу.

Последнее, что стоит упомянуть, мы можем использовать две переменные в одном цикле, например добавим переменную «k» и используем метод items, чтобы получить все тот же результат (Рис.21).



(Рис.21)

### Интересные ссылки

<https://dzen.ru/a/YfKAlDtRwx5KkobF> - Зачем программисту английский и как его учить