**2023/09/29 00:00|Лекция. Стиль кода. Часть I**

## Стиль кода. Часть I

Сегодня мы с вами разберем что такое стиль кода и почему он важен. Приступим!

### Что такое стиль кода?

На наших предыдущих занятиях можно было заметить, что при написании переменных, перед различными операторами, при написании каких-либо конструкций и т. д. мы оставляли отступы (Рис.1). И на самом деле все эти правила были придуманы не просто так.



(Рис.1)

Сам по себе Python позиционируется как достаточно простой и удобно читаемый язык программирования. Эти простоту и удобство и создает стиль кода. Существует такая вещь, как Дзен Python-a, который содержит в себе так называемые правила написания «красивого» кода. «Дзен Пайтона» — это философия программирования от Тима Петерса, она состоит из 19 «руководящих принципов» написания компьютерных программ, влияющих на структуру языка.  
Открыть его можно, введя в консоль «**import this**» (Рис.2).



(Рис.2)

Эти правила гласят, что, например, красивое лучше чем уродливое или, что простое лучше чем сложное и тому подобное.

### PEP 8

Для того, чтобы заложить основы условно правильного написания кода существует свой свод правил – **PEP 8**. Это определенный документ, то есть стандарт, определяющий стиль написания кода в Python.

В соответствии с этим стилем мы рассмотрим ряд важных моментов, на которые стоит обращать внимание при написании кода. Ведь чем раньше начать писать код красиво и понятно, тем быстрее это станет привычкой и не доставит проблем в будущем. Это не всегда влияет на работу кода, но всегда влияет на степень понимания того, что код несет в себе. Плохо оформленный код спустя время будет сложнее прочитать и сразу разобраться за что он отвечает, а работа программиста по большей части состоит именно в чтении кода. Поэтому при его написании необходимо сделать так, чтобы читать его было удобно и нам, и людям, которым мы его передадим, например по работе.

### Основные правила стиля кода

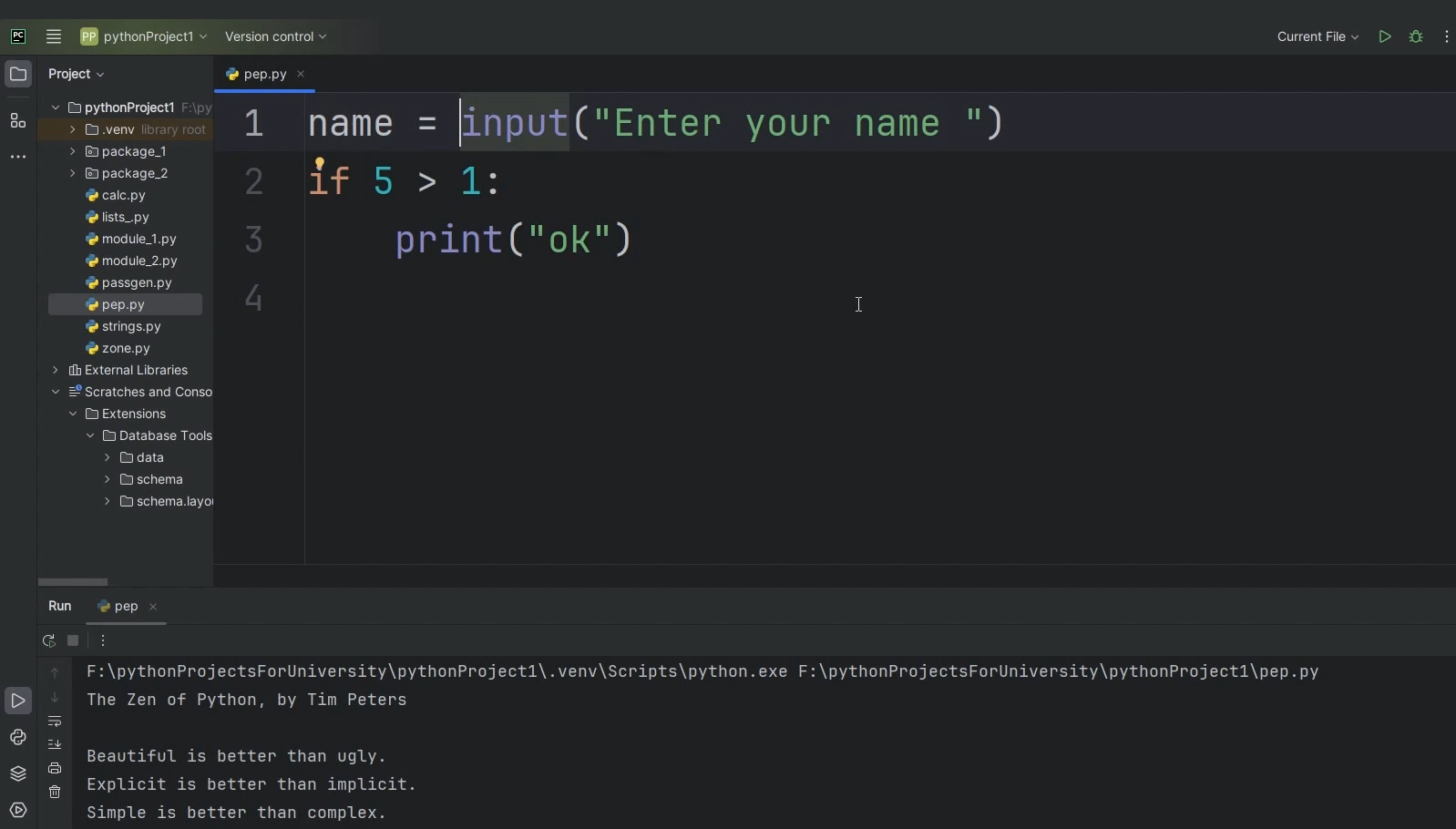
Влиять на хорошее или не очень оформление кода могут разные факторы.

Например, пробелы перед и после основных операторов, таких как равно (=), больше (>), меньше (<) и т. д. Разберем на нашем примере. Мы можем и не ставить пробелы перед и после знака равно, код будет работать, но часто можно заметить, что некоторые вещи сам PyCharm подчеркивает желтой волнистой линией (Рис.3).



(Рис.3)

И если мы нажмем на нее правой кнопкой мыши, чтобы нам показали действия, которые могут быть применимы в данной ситуации, то увидим строчку «Reformat the file», которая приведет подчеркнутую часть к должному виду, в нашем случае вернет пробелы перед и после знака равно (Рис.4).

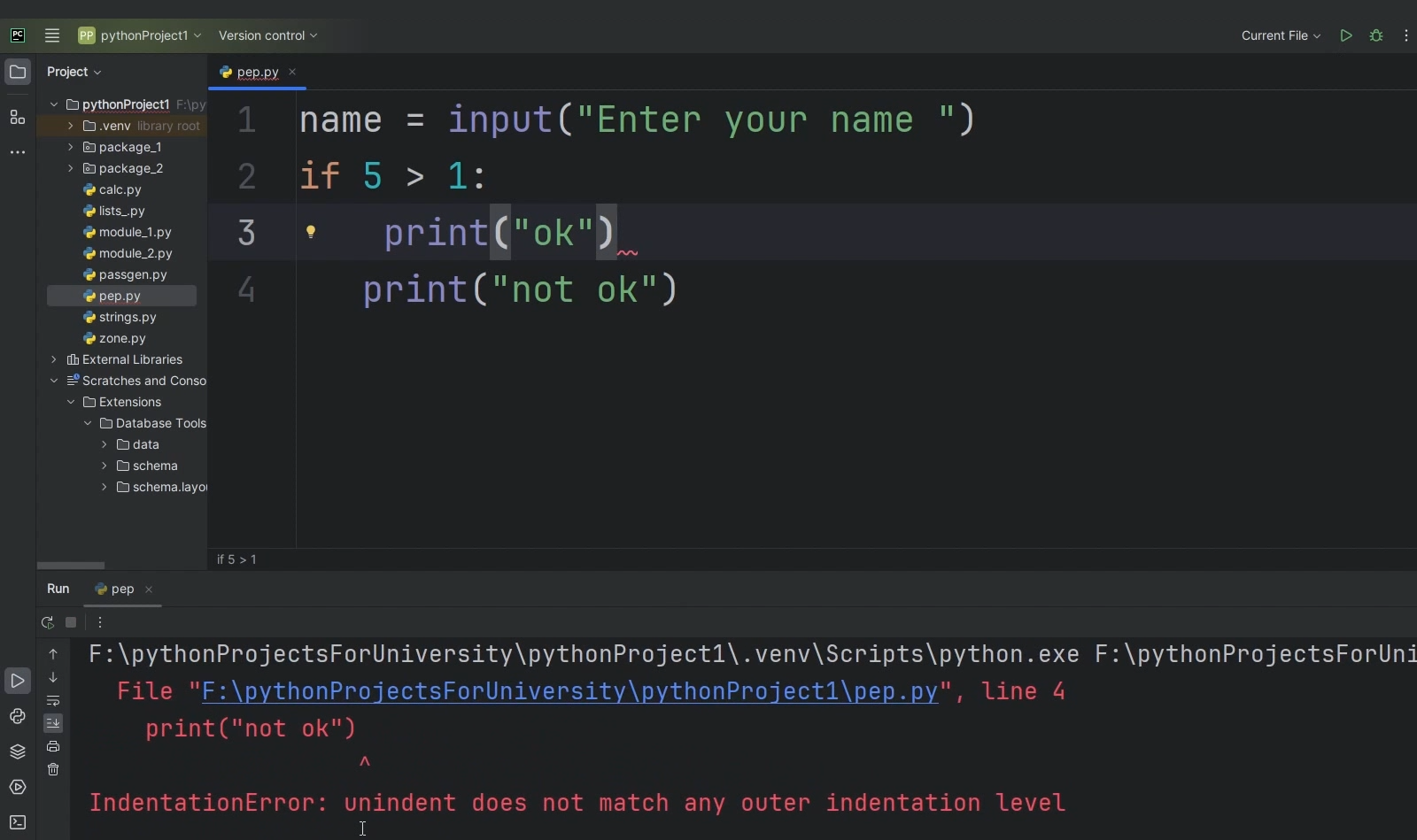


(Рис.4)

Эти правила начали разрабатываться практически с момента создания самого языка и до недавнего времени они постоянно обновлялись и дополнялись. Благодаря чему сейчас у нас есть довольно четкие стандарты, которые делают код понятным и приятным глазу для всех.

Мы также уже не раз сталкивались с отступами после двоеточий. Как мы знаем, после нажатия Enter после двоеточия код начинается с отступом в четыре пробела и так мы понимаем, что эта строка относится непосредственно к предыдущей строке кода. Отступ этот совершается или четырьмя нажатиями пробела или с помощью нажатия Tab, разница лишь в том, что в первом случае отступ засчитывается как четыре символа, а во втором как один. На работу кода это не влияет.

Зато влияет разница отступов. Например, вот наш код с двоеточием, после него идет строчка кода с отступом в четыре пробела, а уже после нее еще одна строчка кода уже с отступом, допустим, в три пробела (Рис.5). И сейчас, если мы запустим наш код, то увидим ошибку.



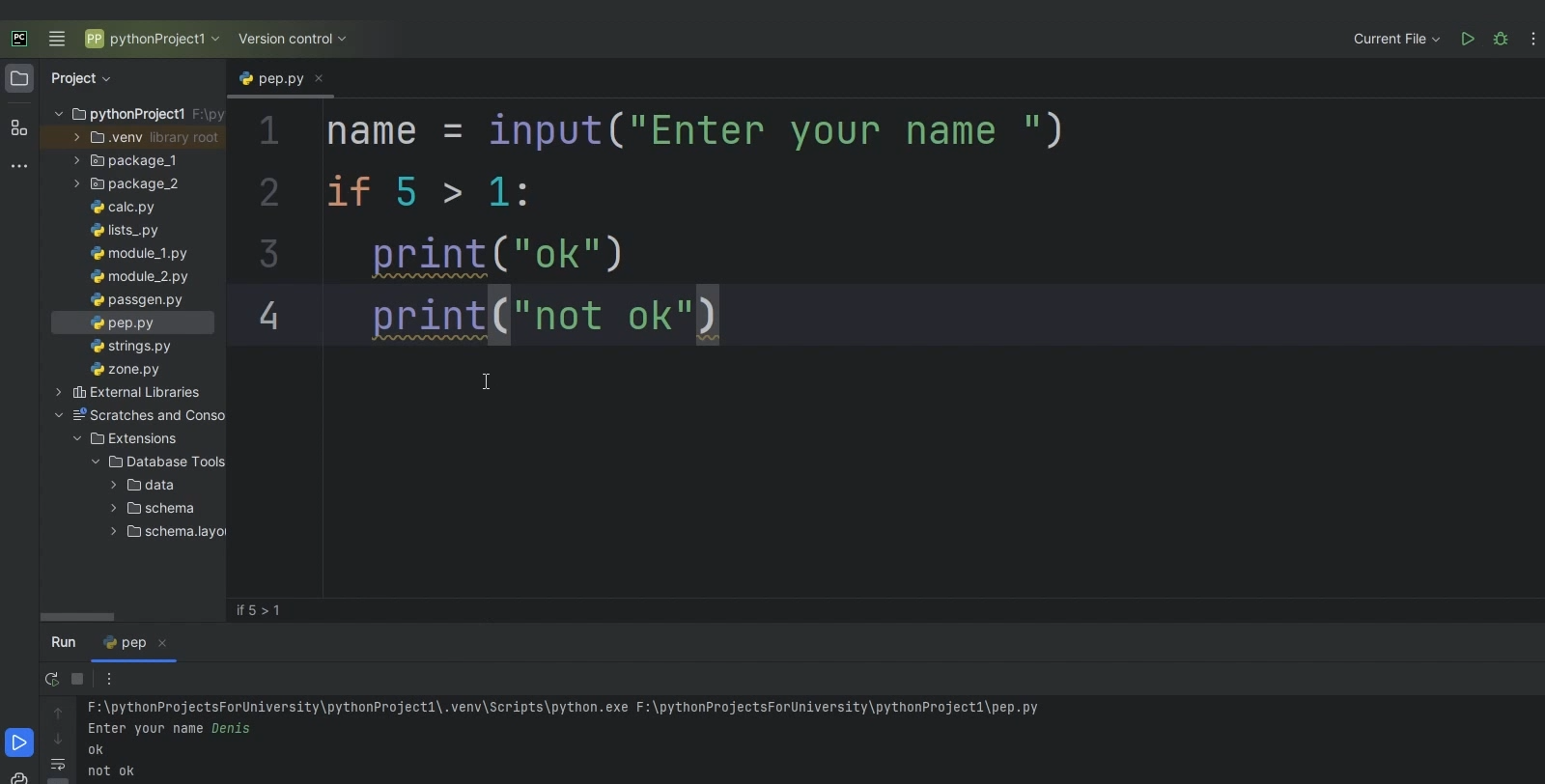
(Рис.5)

Ошибка нам сообщает, что отступы не соответствуют друг другу, потому что строки кода, относящиеся к одной предыдущей строке кода, должны находиться на одной линии. Если мы исправим наш отступ, то все сразу заработает (Рис.6).



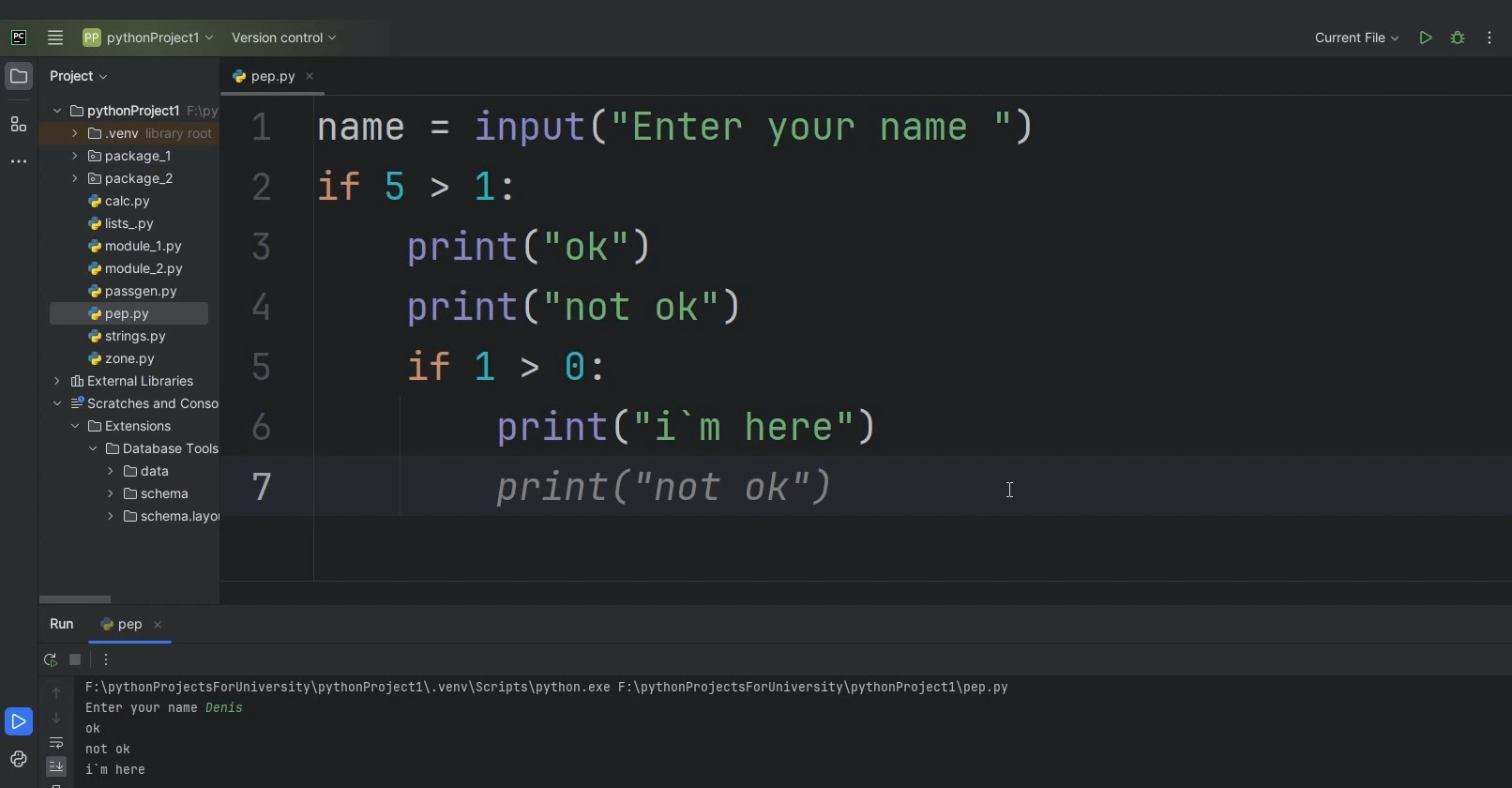
(Рис.6)

Теоретически, мы можем запустить код с отступами в два пробела и все сработает, главное, что они на одной линии (Рис.7). Но в данном случае уже PyCharm подчеркивает наш код, сообщая, что тут какая-то ошибка в соответствии с правилами написания. Код пусть и работает, но принято, что после двоеточия должно быть четыре пробела.



(Рис.7)

Мы можем встретить еще и вложенные конструкции. Например, у нас код с условием, после которого идет отступ. Но внутри этого условия может быть еще одно условие и после него код будет опять-таки идти с отступом. Таким образом у нас получается своеобразная лесенка (Рис.8). Такое может встретиться в разных конструкциях, с которыми мы познакомимся позже.



(Рис.8)

На самом деле таких простых самых базовых правил немного и их довольно просто запомнить. Благодаря чему, можно с самого начала привыкать к тому, что код пишется в соответствии с рядом норм.

Название переменных тоже относится к правилам. Есть общепринятая договоренность, что мы называем переменные таким образом, чтобы позже по названию понять, что находится внутри переменной.

### Форматирование кода

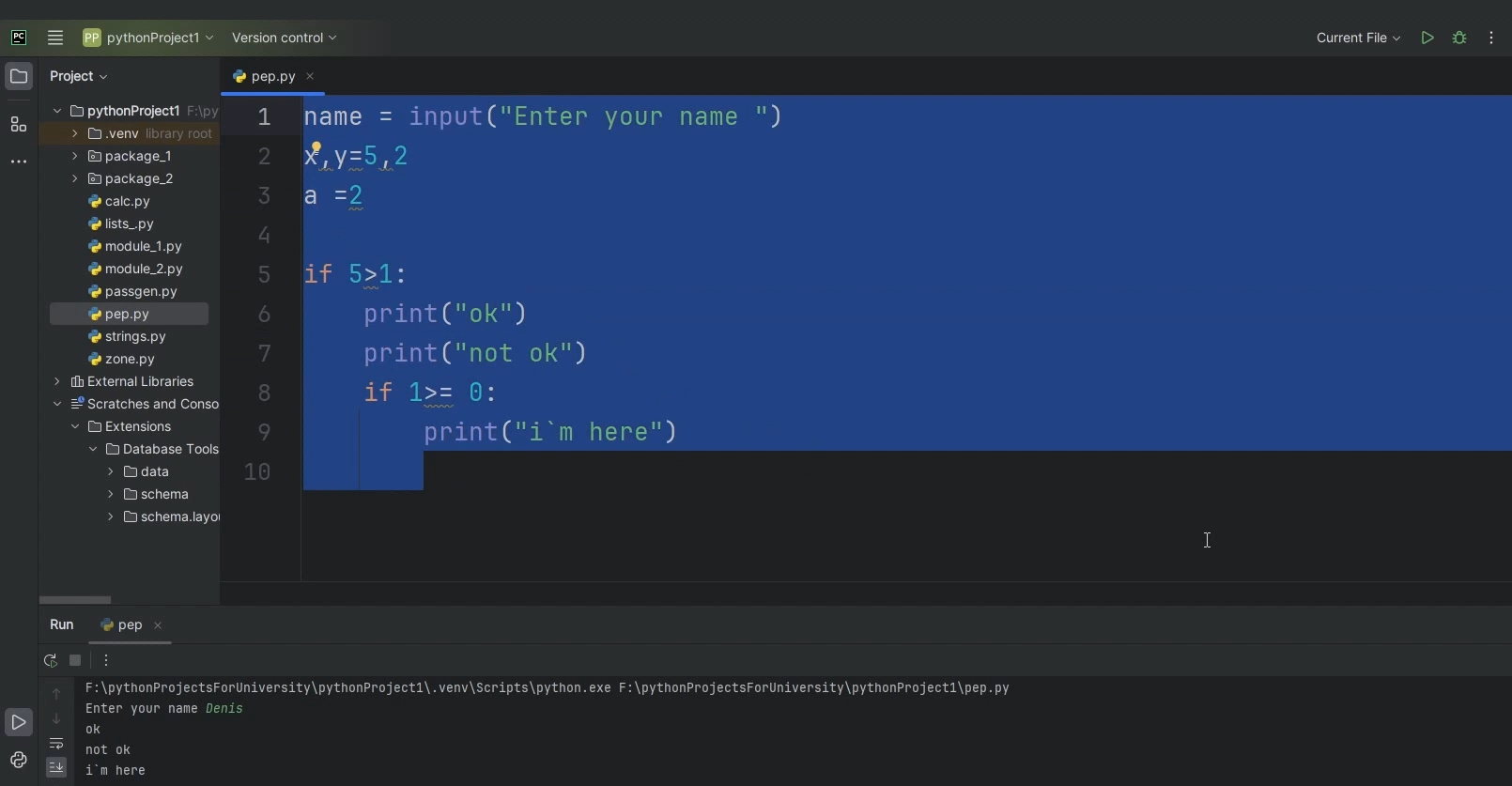
Для примера напишем частичку без соблюдения каких-либо норм (Рис.9). Как видим, читать такой код крайне неудобно. Но, допустим, вот мы в этом убедились, а если у нас код уже написан таким образом, то как быстро привести его к должному виду?



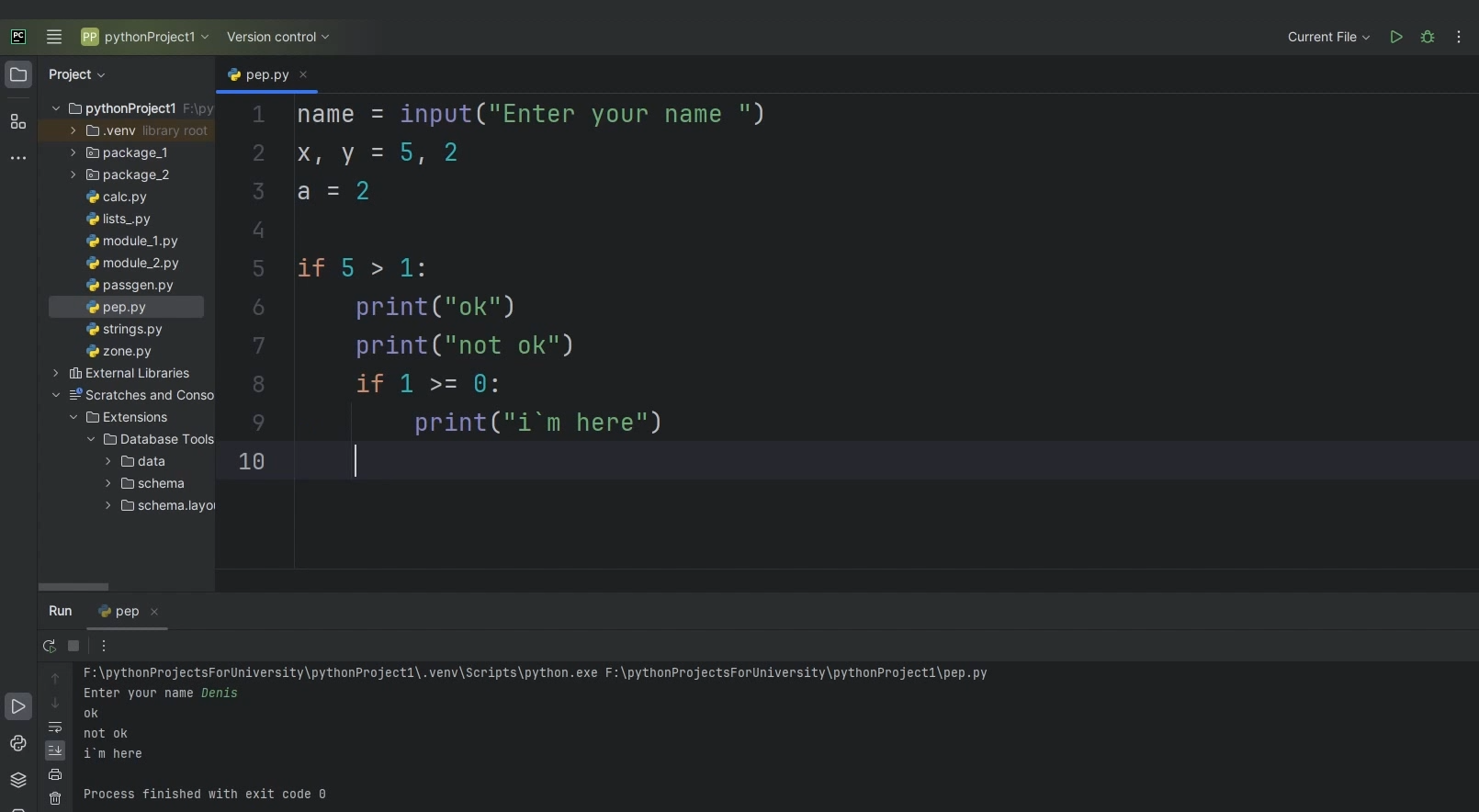
(Рис.9)

Здесь есть два варианта. Первый уже нам известен, нужно нажать правой кнопкой мыши на подчеркнутую часть кода и опять-таки щелкнуть «**Reformat the file**». Файл будет автоматически отформатирован в соответствии со стандартами PEP 8.

А второй вариант, выделить код, который нам нужно отформатировать (Рис.10) и нажать **Ctrl + Alt + L**. После этого стиль кода будет исправлен (Рис.11).



(Рис.10)



(Рис.11)

### Зачем нужен стиль кода?

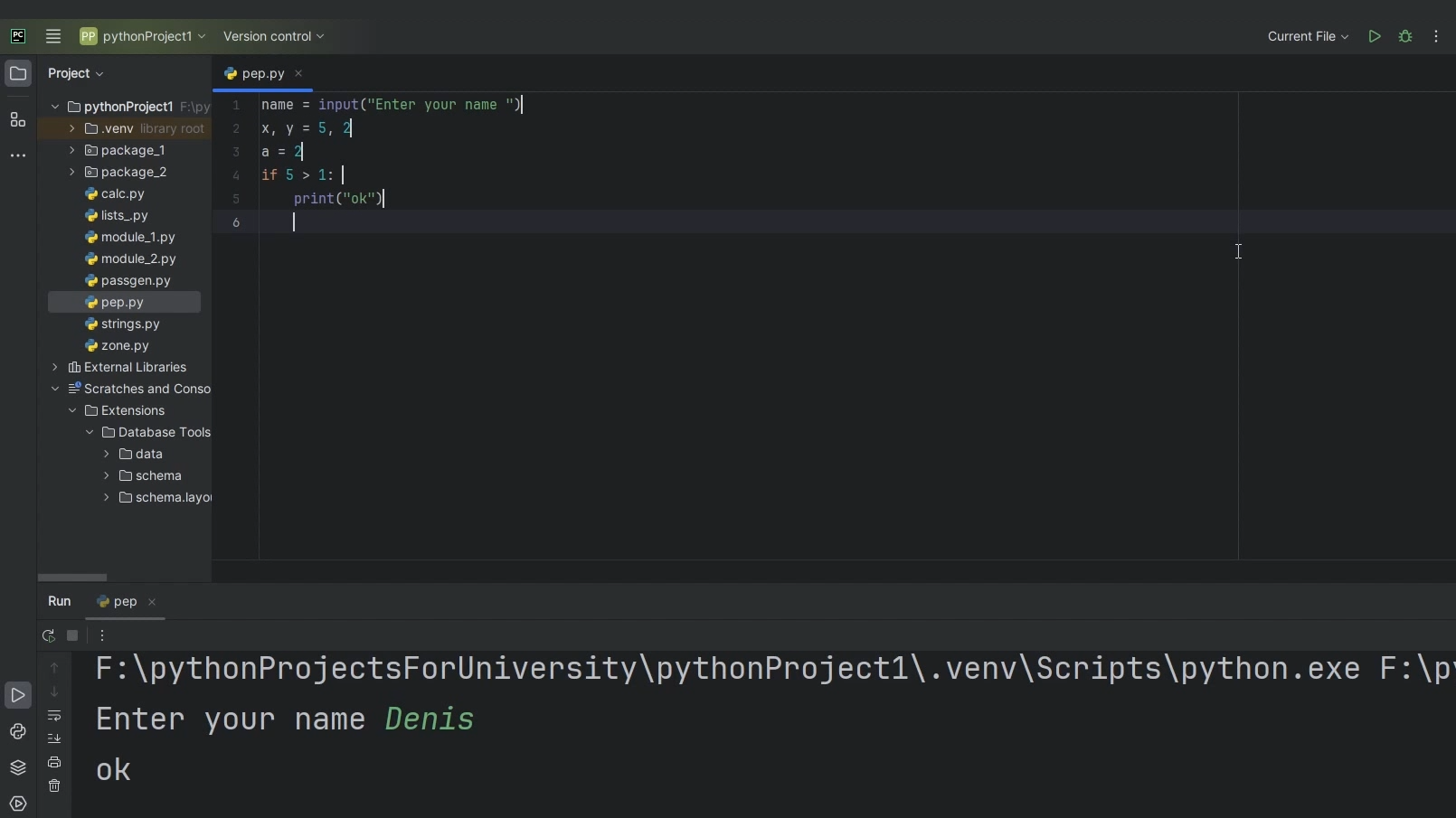
Зачем же нужна вся эта «красота» написания, действительно ли это так важно? Возьмем для примера C++, при его создании не слишком учитывалась какая-то эстетическая составляющая, упор был справедливо сделан на эффективность и функциональность. Поэтому там можно часто встретить конструкции, которые пишутся в одну строку, то есть код может быть очень и очень длинным. И когда мы читаем этот код, нам сложно уловить, что происходит в этой строке, то есть помимо смысла, нам нужно искать какие-то разделяющие символы для обозначения границ той или иной команды.

В Python технически мы можем также писать код, например, сразу после двоеточия (Рис.12), код будет работать, но PyCharm cнова будет «недоволен».



(Рис.12)

Но, помимо этого, чтобы предотвратить появление чрезмерно длинных строчек кода, в PyCharm есть вертикальная линия, за которую мы не должны переходить (Рис.13). Если код пересечет ее, то ничего страшного конечно же не произойдет, и он все равно будет работать. Но опять же читать его будет уже весьма неудобно, а PyCharm обязательно сообщит нам через подчеркивание, что здесь нарушен стиль.



(Рис.13)

Казалось бы, все это такие простые правила, которые не оказывают влияние на работоспособность нашего кода. Однако они очень важны, ведь именно благодаря им нам проще ориентироваться в коде и понимать его.

### Интересные ссылки

<https://peps.python.org/pep-0008/> - PEP 8 - руководство по стилю кода в Python