**2024/01/10 00:00|Лекция. Систематизация тестов**

## Систематизация тестов

Перед нами стоит одна задачка, которая связана с масштабированием тестов и удобством их использования. И для этого сейчас сделаем для себя такой валаунчер наших тестов, который будет собирать их и запускать. Назовём его «test\_suite». Создадим его (Рис.1, Рис.2).

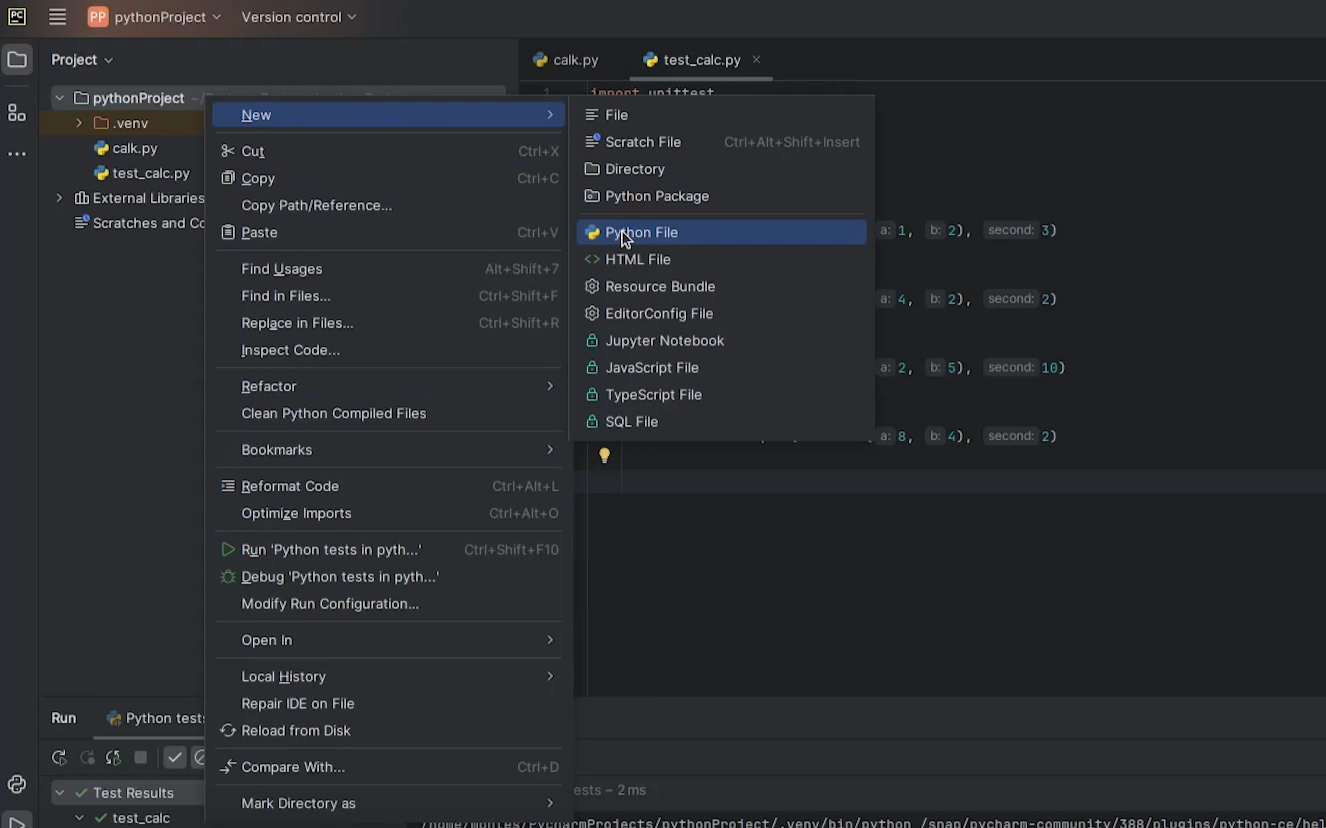


Рис.1



Рис.2

Что для него нужно? Ему прежде всего нужен импорт самого «unittest» и импорт наших файлов с тестами. В нашем случае это «test\_calc» (Рис.3).

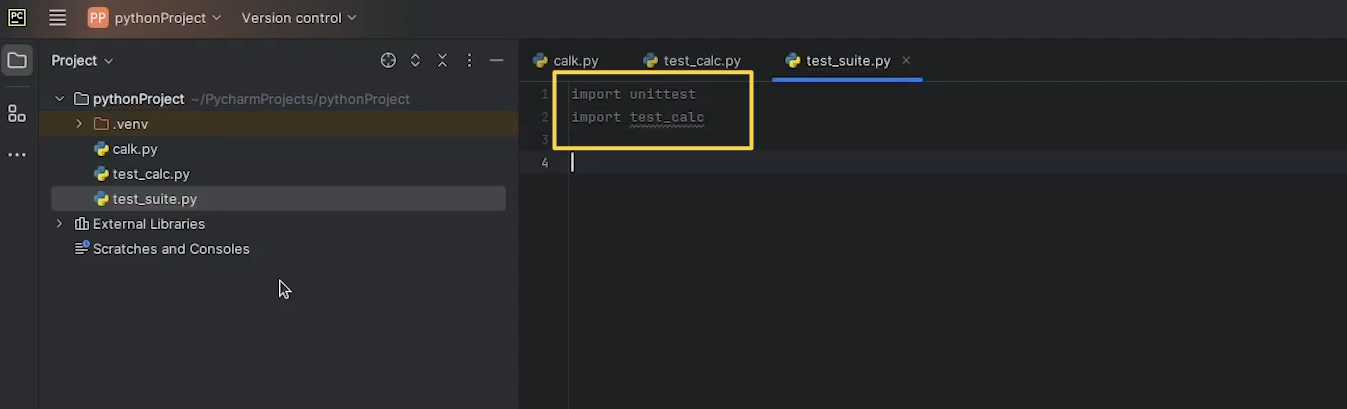


Рис.3

Создадим этот тест «calcST». Он будет наследоваться от «unittest.TestSuite» (Рис.4).

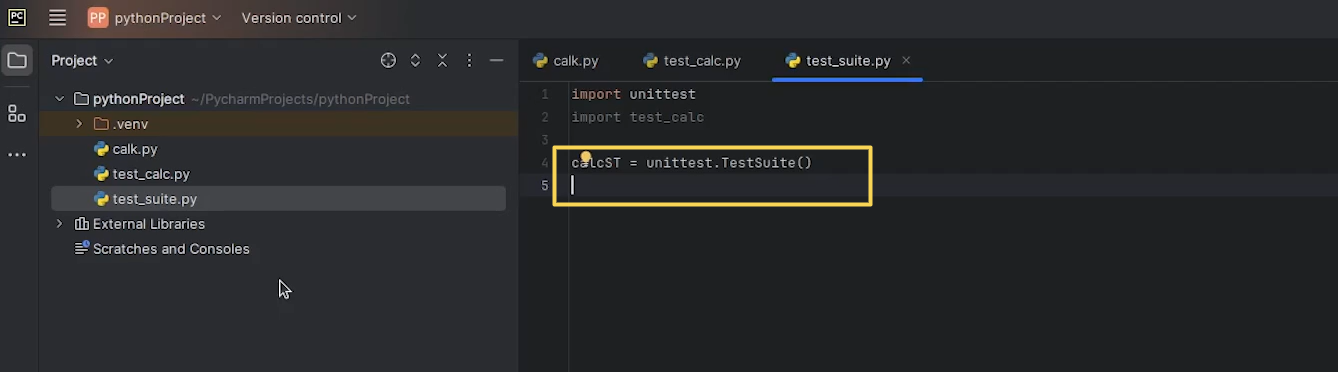


Рис.4

Мы получили некий объект «test\_suite». Но в него нужно как-то запихнуть наши тесты. Делается это достаточно легко «calcST.addtest(unittest.makeSuite(test\_calc.CalcTest))» (Рис.5). Здесь используется функция «unittest.makeSuite», затем мы обращаемся к нашему файлу с тестами и подгружаем нужный класс с тестами. Раньше этот метод использовался часто, но сейчас он устаревает.

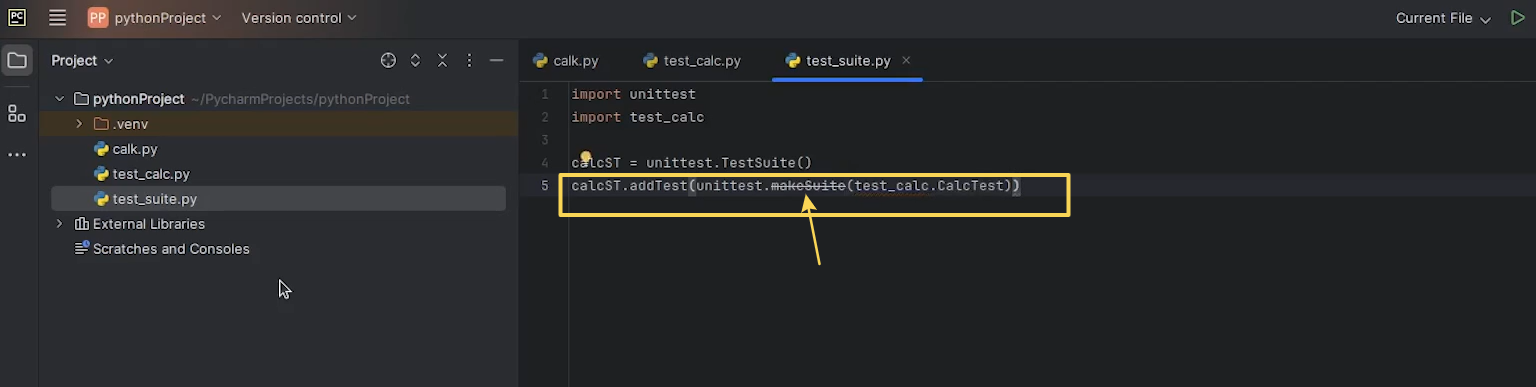


Рис.5

Если заметили, у нас вычеркнут «makeSuite». И говорится о том, что он в версии Python 3.13 перестанет работать (Рис.6).

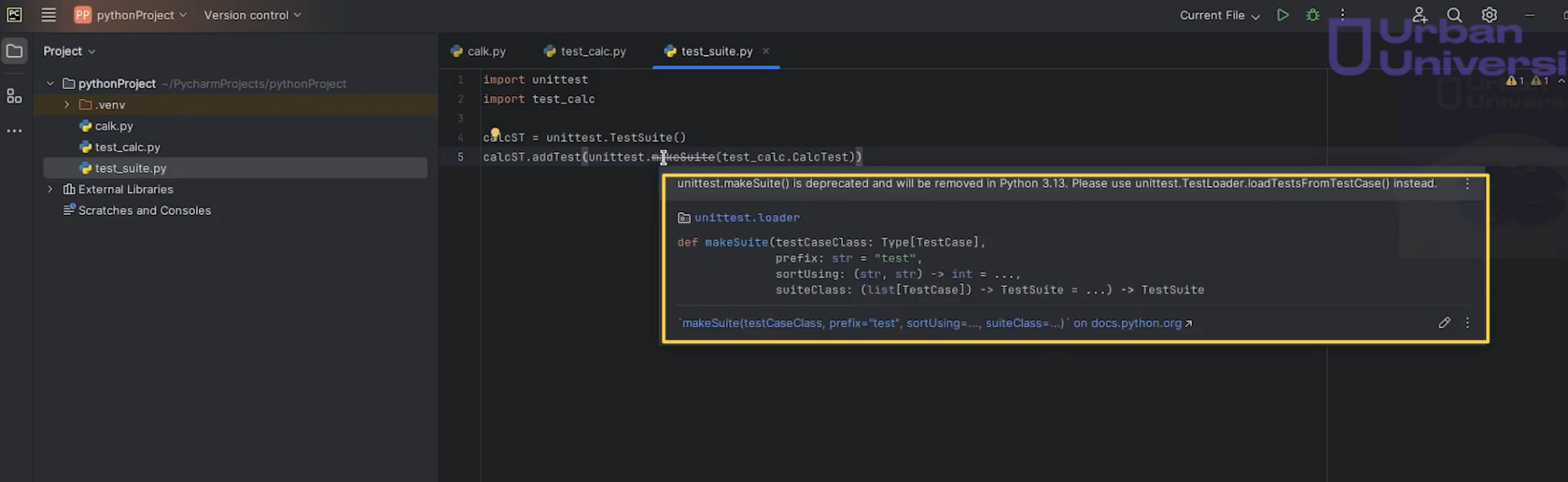


Рис.6

Есть более современный метод, который нам позволит решить эту проблему (Рис.7). Он также добавляет тесты, но уже через метод «TestLoader» и функцию «LoadTestsFromTestCase». Дальше также указываем путь до нашего «TestCase», то есть модуль откуда мы берём это и название класса.

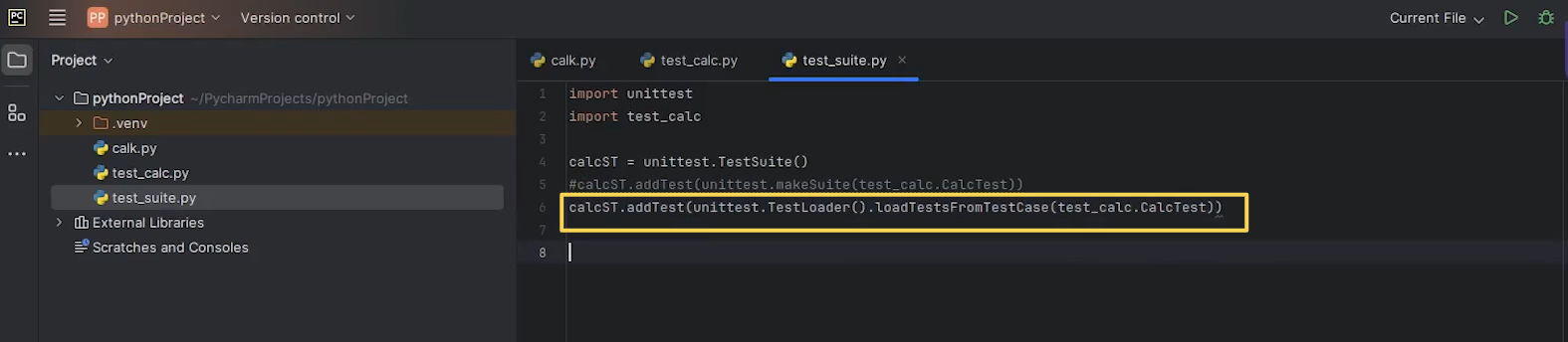


Рис.7

Дальше мы можем это дело сконфигурировать по запуску. Можно сделать отдельный файл «runner». Но можно сделать прямо здесь: также обращаемся к «unittest» и берём «TextTestRunner» (Рис.8).



Рис.8

Также мы его можем запускать прямо здесь. Надо объяснить, по каким «TestSuite» мы его будем запускать. В нашем случае это «calcST». Если мы его запустим, то увидим, что все тесты пройдены и все прекрасно работает (Рис.9).

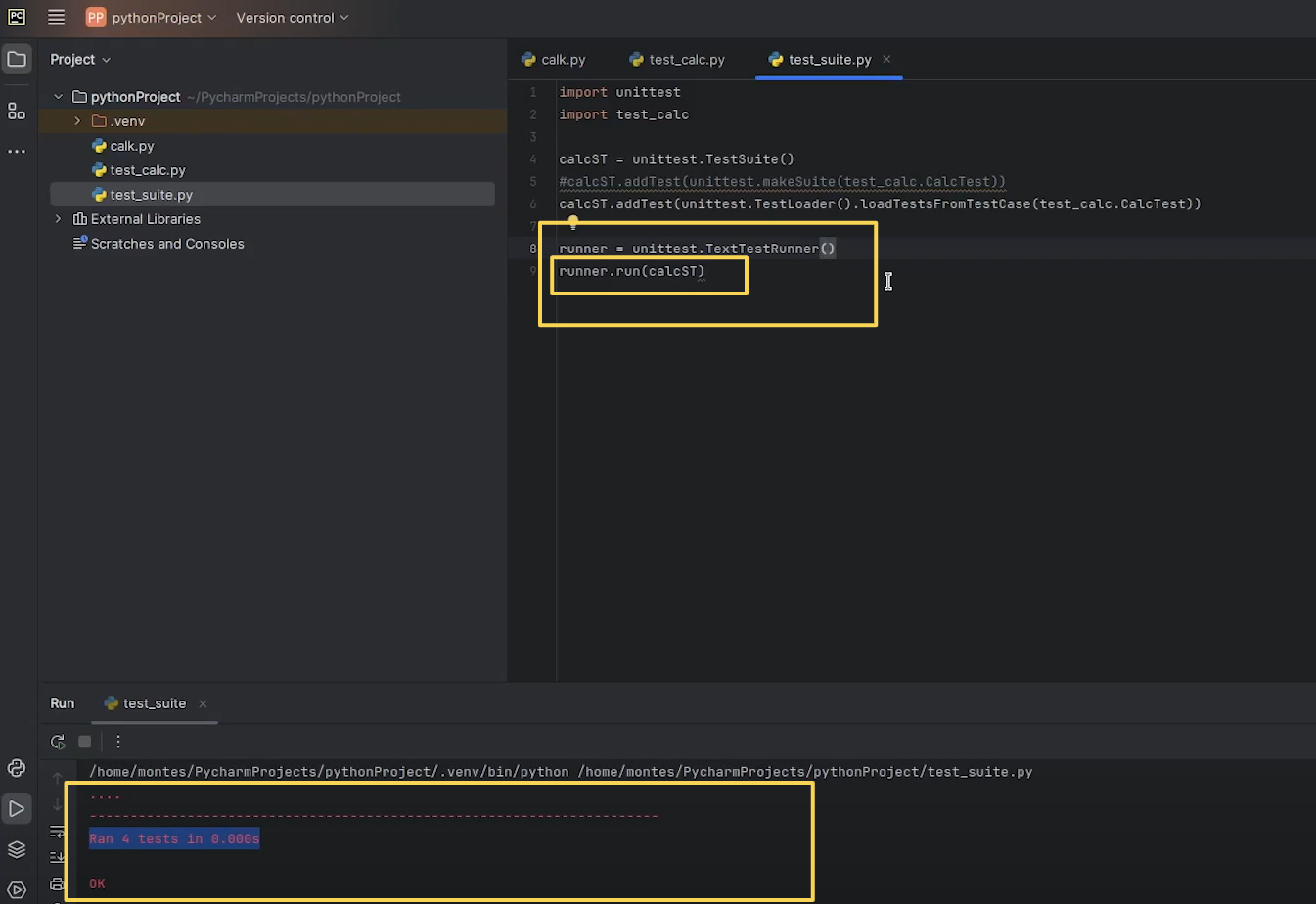


Рис.9

Есть ещё параметр «verbosity». В нашем случае он позволяет менять детализацию. К примеру, как видите, выявилось, какие тесты конкретно мы прошли (Рис.10).

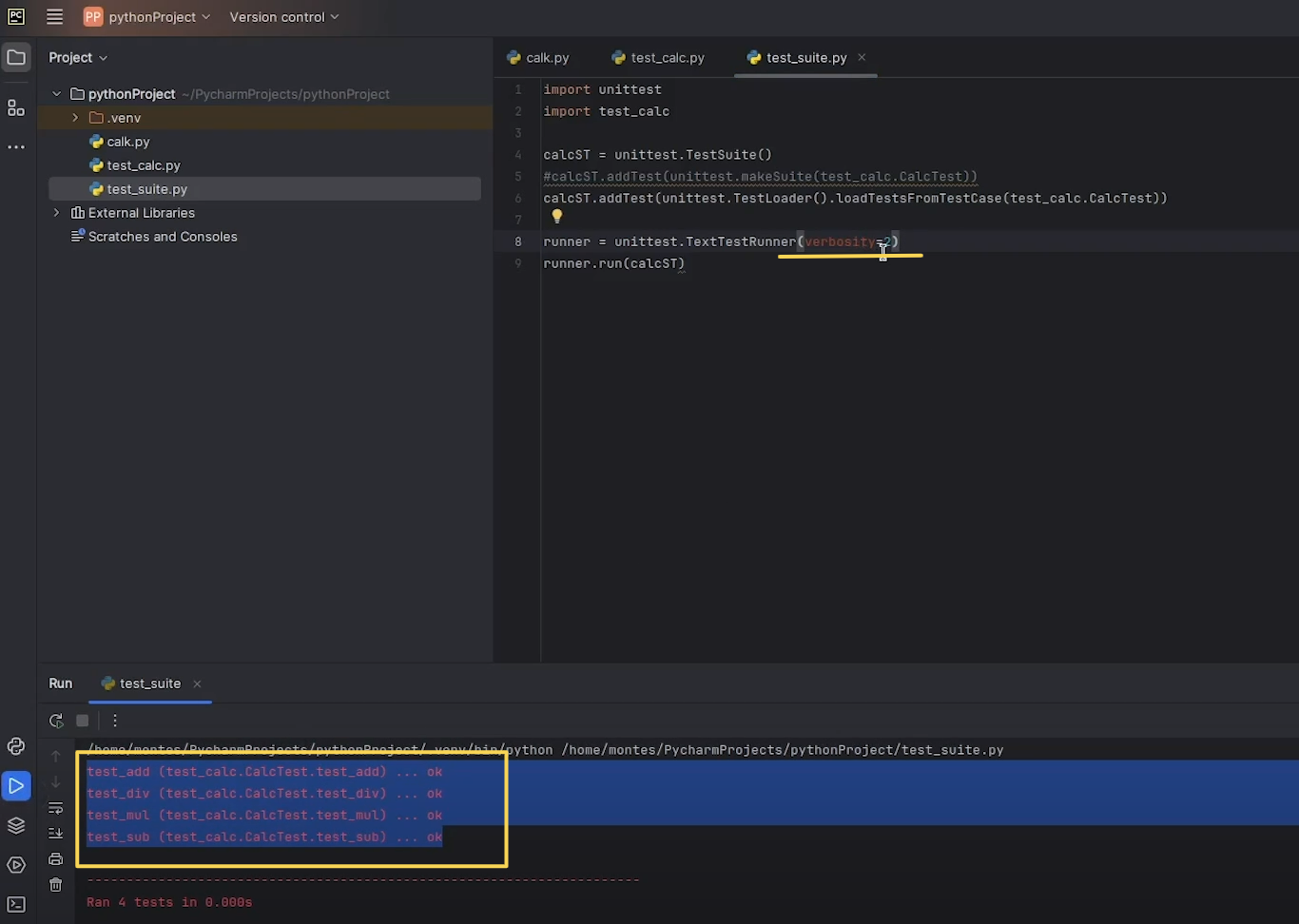


Рис.10

И на двоечке он выводит нужные нам параметры, какие конкретно тесты прошли (Рис.11, Рис.12)

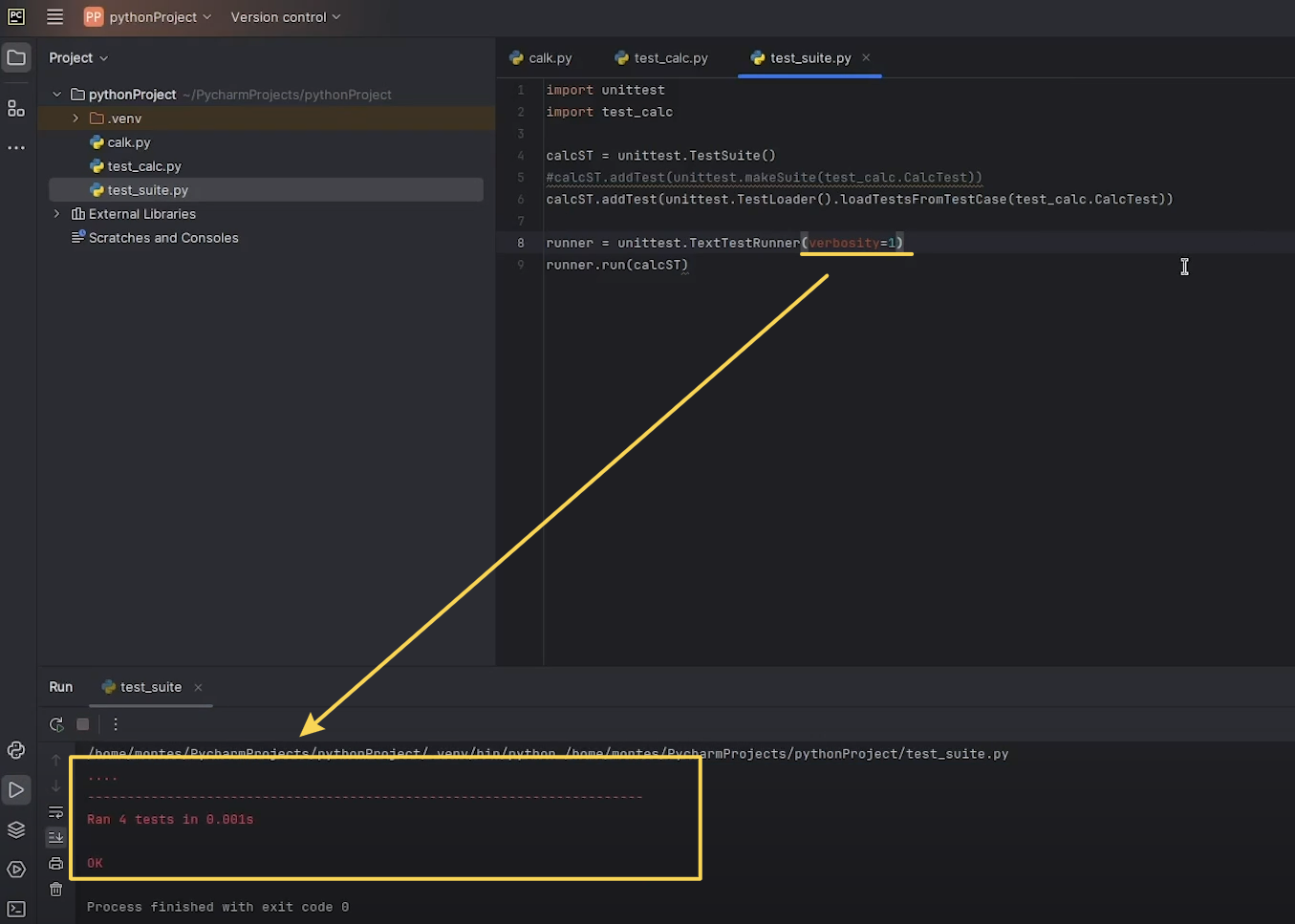


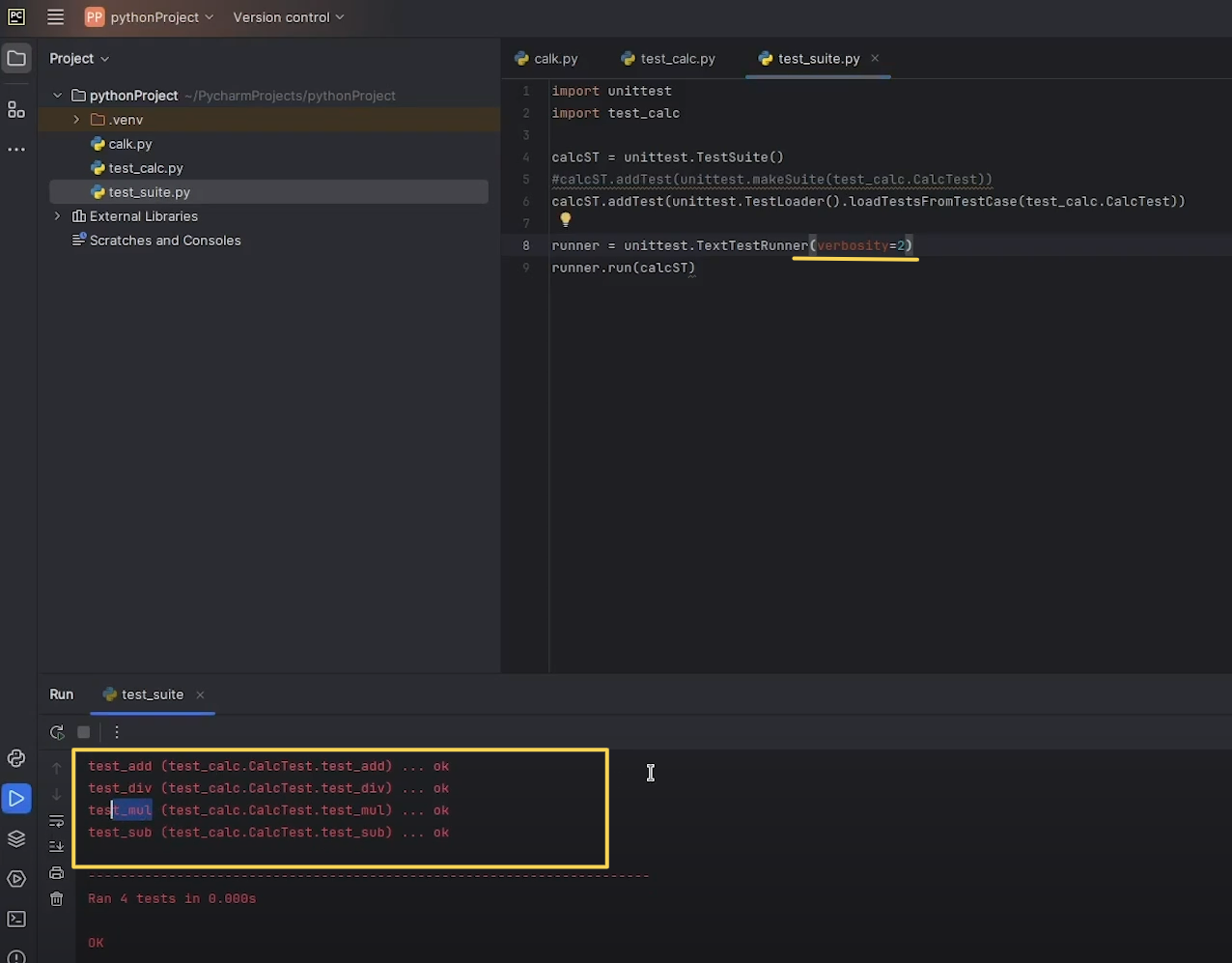
Рис.11

Рис.12

А зачем мы, собственно, это делали? Мы же до этого все могли запускать. Ну, на самом деле ситуация очень простая. Представим, что вы являлись разработчиком, который сделал эти 4 функции (Рис.13).

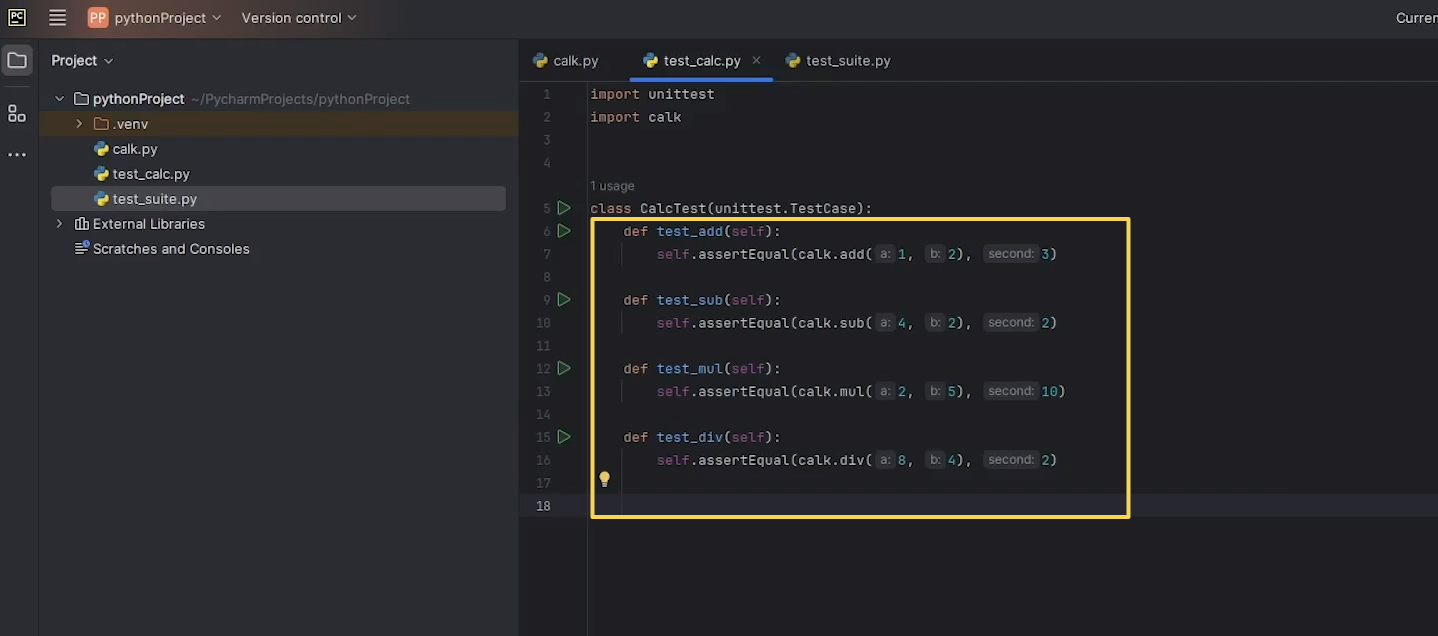


Рис.13

Потом пришёл ваш друг и не знает, что вы делали раньше и делаете сейчас. И он добавил новые функции. Например, функцию корня, которая возвращает значение арифметического корня. И функцию возведения в степень «pow», которая будет возводить «a» в степень «b» (Рис.14). На самом деле неважно какие функции. Суть в том, что у нас появились новые функции.



Рис.14

И он, естественно, для себя сделал свой набор тестов. Сделаем для него файл «test\_new\_calc». И он себе тоже импортировал «unitest», калькулятор (Рис.15).

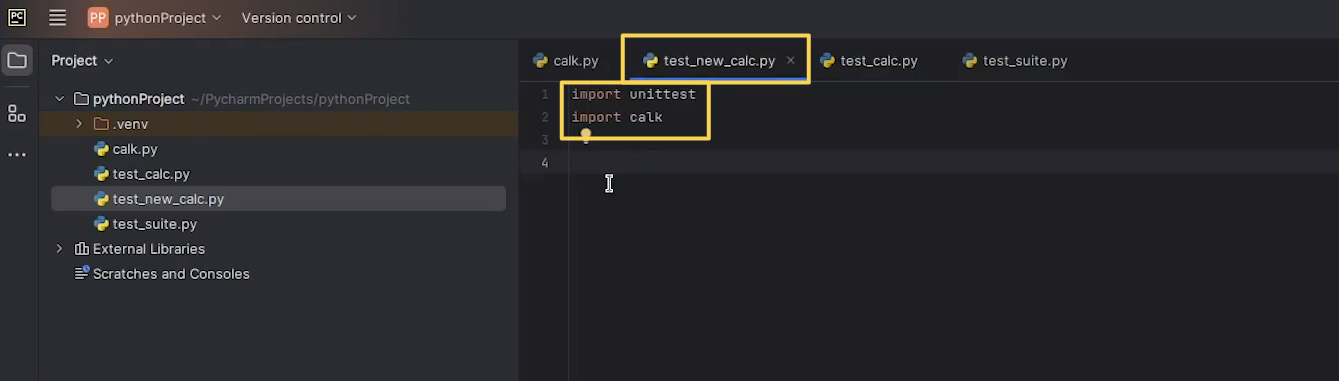


Рис.15

И скорее всего он повторил структуру файла «test\_calc». Давайте мы с вами это и сделаем: напишем «class NewCalcTest», и наследоваться он будет от «unittest.TestCase» (Рис.16).

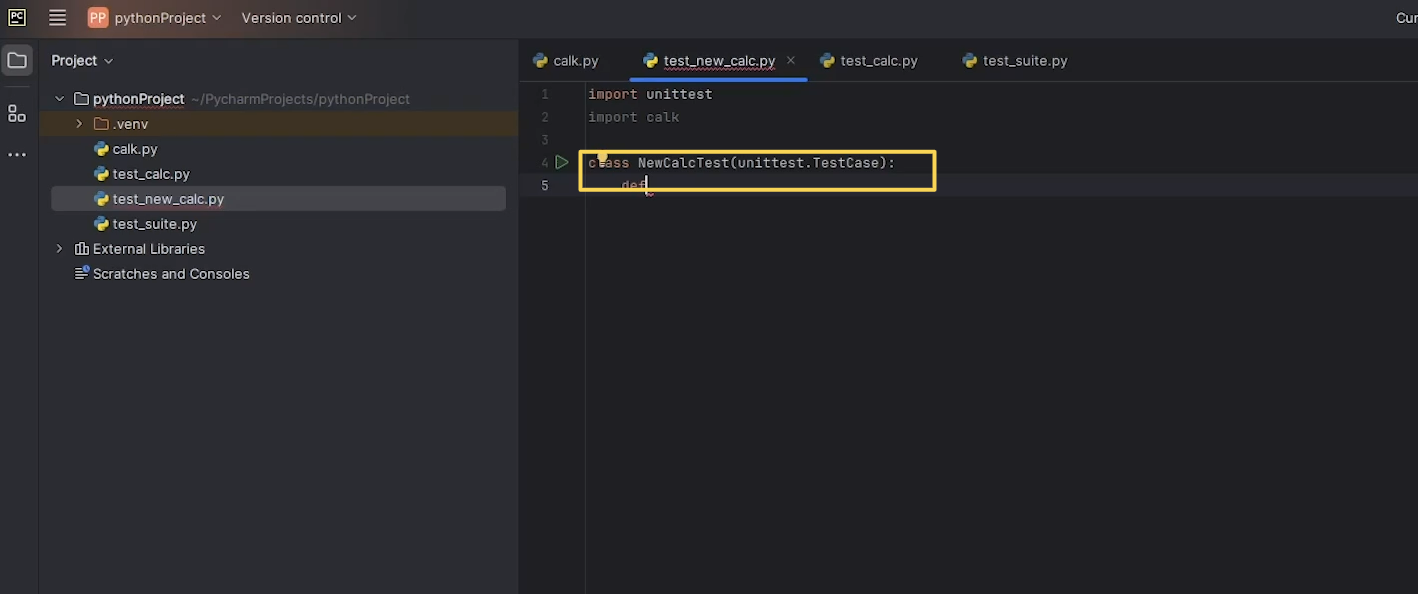


Рис.16

И сделаем ему даже набор функций, набор тестов. К примеру добавим ему тест на корень «test\_sqrt». И пусть он проверяет также на равенство «self.assertEqual». И чтобы не пытаться что-то выдумывать, очень сложное, обращаемся к калькулятору функции «sqrt» от 4. От 4 это будет 2 (Рис.17).

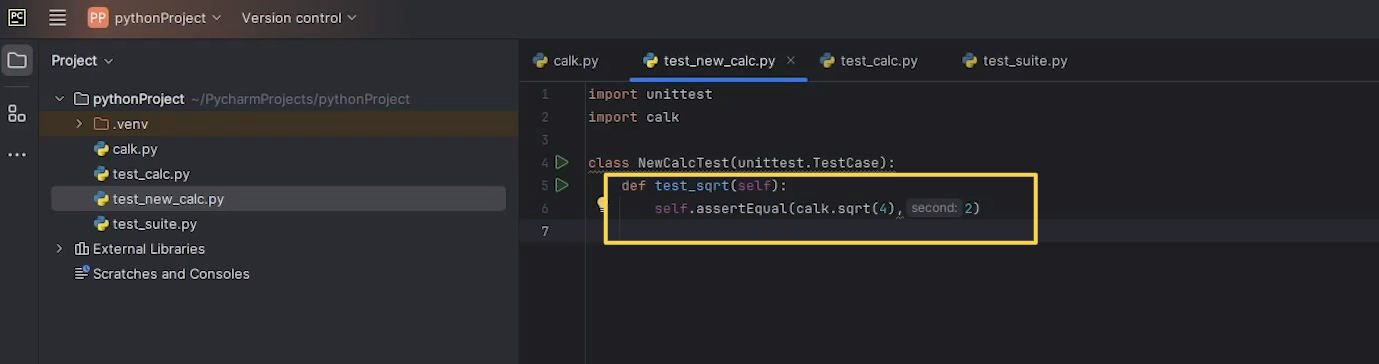


Рис.17

Дальше добавим тест на возведение в степень тест «test\_pow». Пишем проверку на равенство «self.assertEqual», обращаемся к калькулятору «pow». Возьмём 3 в 3 степени. Это должно получиться 27 (Рис.18).

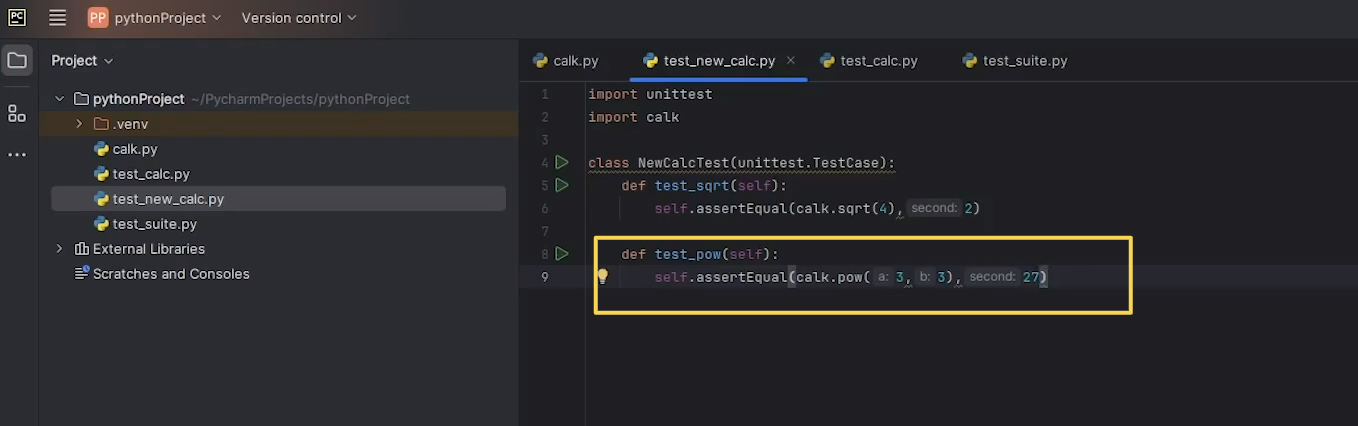


Рис.18

Ваш друг написал себе эти прекрасные тесты и залил их на ГИТы. Если бы мы действовали как раньше, нам приходилось бы каждый файл по отдельности добавлять. А теперь нам нужно просто добавить новый блок тестов. Давайте мы его импортируем и подключим (Рис.19).

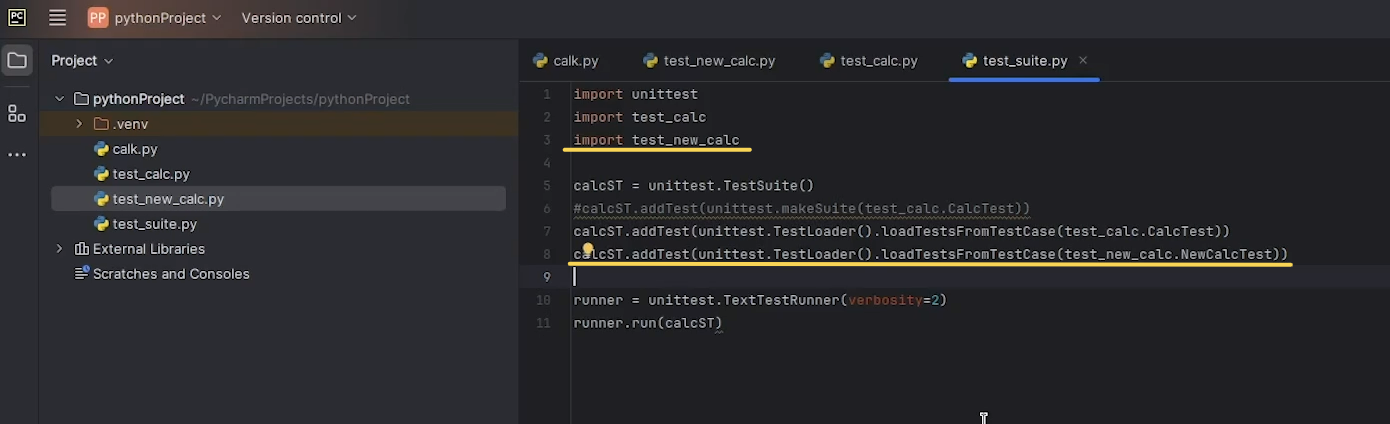


Рис.19

Мы сейчас добавляем все в один «test\_suite». А мы можем собрать их как разные и запускать ещё их по отдельности. Мы сейчас все единым блоком запустим и видим, что все прошло. И с «test\_calc» прошли тесты и с «test\_new\_сalc» (Рис.20).

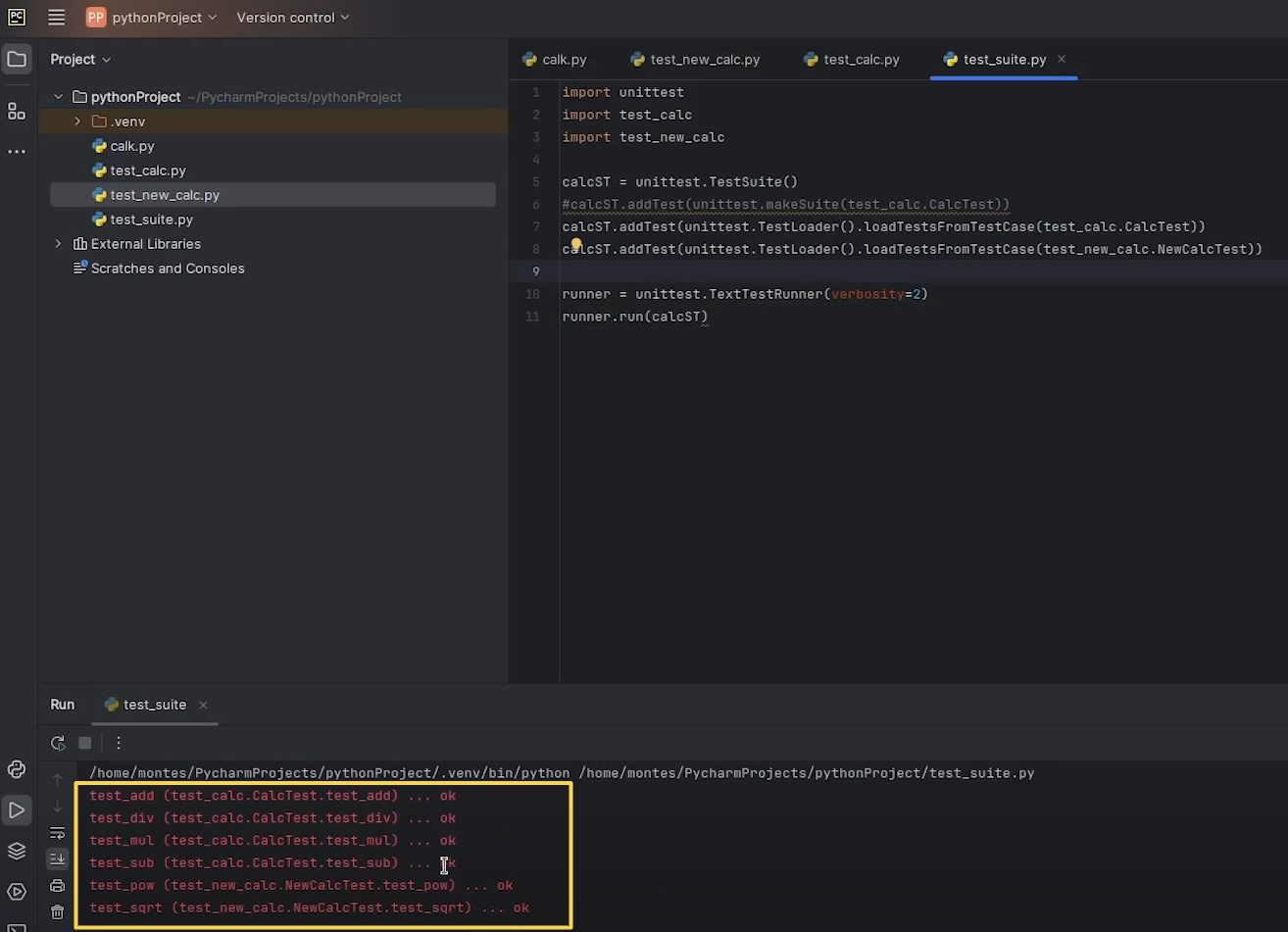


Рис.20

Это действительно очень удобно. Почему? Потому что мы можем сделать себе папочку «tests» (Рис.21, Рис.22, Рис.23).

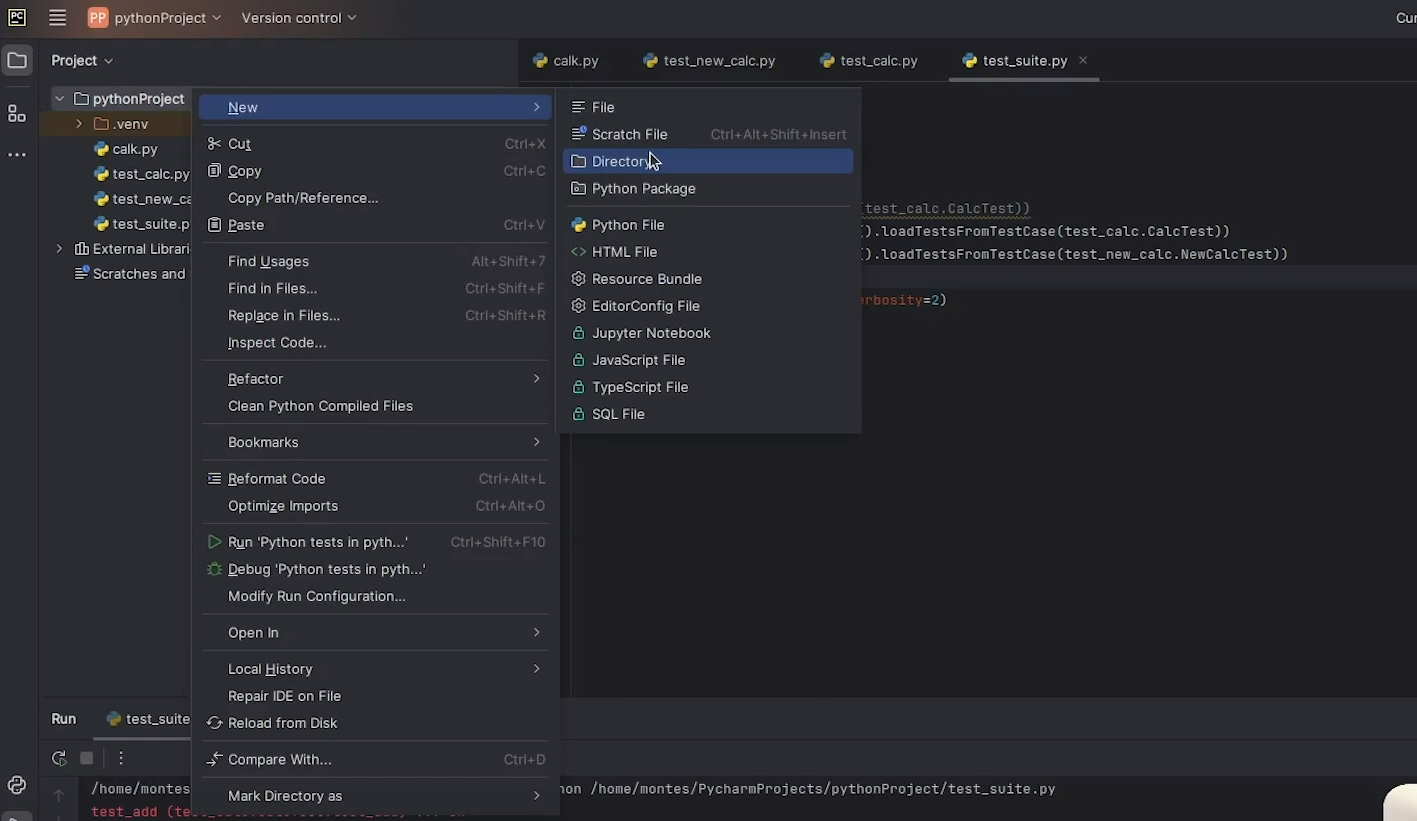


Рис.21

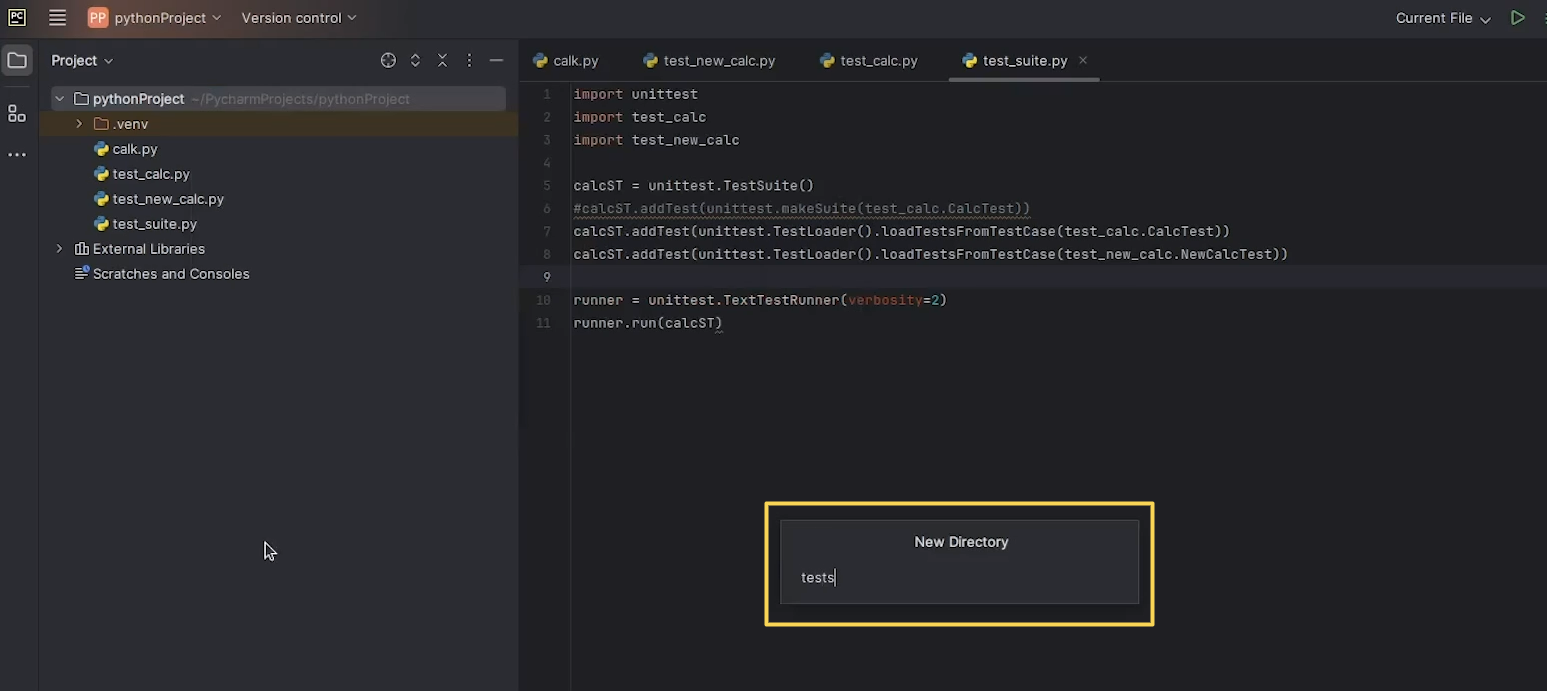


Рис.22

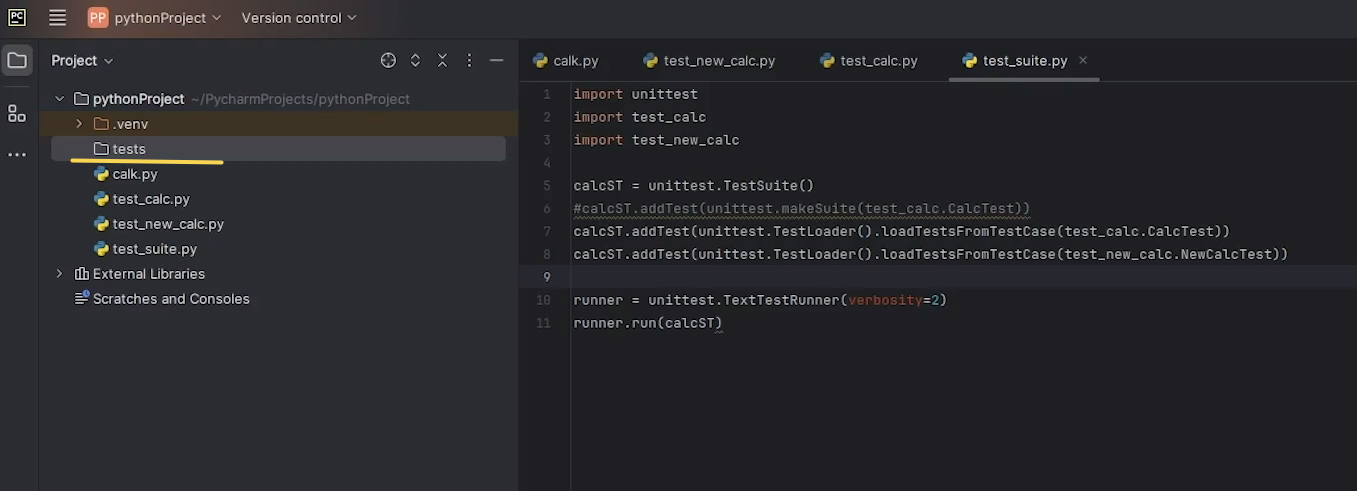


Рис.23

И переносим в неё все наши файлы и получаем удобную корректную структуру для будущего проекта. Для этого эта система нужна дальше. Уже я, Как мы и говорила, можно, к примеру, «runner» вытащить тоже в отдельные файлы, если он у вас обрастёт большим количеством настроек, для того, чтобы было ещё удобнее работать.

Важно! «TestSuite»-это наборы, которые мы с вами собираем, добавляя в них тесты. И если нам необходимо, можем добавлять разные «test\_suite» и запускать их по отдельности, что будет идеально для нас. И такая структура подходит под серьёзный промышленный уровень разработки.