**به نام خدای رنگین کمان**

**تست نویسی :**

کلا چرا تست مینویسیم برای برنامه هامون ؟؟ دوتا دلیل کلی داره یکی اینکه خب ببینیم این کدی که نوشتیم درست کار میکنه یا نه و دلیل مهم ترش هم این هستش که کدی جدیدی که مینویسی برای کد های قبلیت مشکلی ایجاد نکنه ، و ما میتونیم این رو با تست نویسی امتحان کنیم.

تو بقیه صنعت ها هم این تست نویسی هستش دیگه ، هم میتونیم به صورت دستی تست کنیم (که خب همه میتونن و اون دلیل مهم تر دوم رو نداره) و هم به صورت اتوماتیک.

برای کسایی هستش که یکم پایتون رو بلدن و زمینه دارن.

با چیا آشنا میشی ؟

1. Doctest

همون داک استرینگ نویسی خودمون هستش. (""" Example:int""") که تو پروژه های بزرگ زیاد استفاده نمیشنه و کد رو شلوغ میکنه.

1. Unittest

مهم ترین ابزار تست نویسی پایتونیک هست و خیلی جاها استفاده میشه ازش. به قدری که توی جنگو میگن با این بیاید و تست بنویسید.

1. Pytest

این هم بر اساس یونیت تست هست و اومده یه سری متود هاشو برای کمتر کرده که ساده تر باشه.

ابزارهای دیگه ای هم هستش مثل ماک (mock) که اینجا چیزی راجبشون نمیگم. خودتون برید بخونید اگه دوست داشتید.

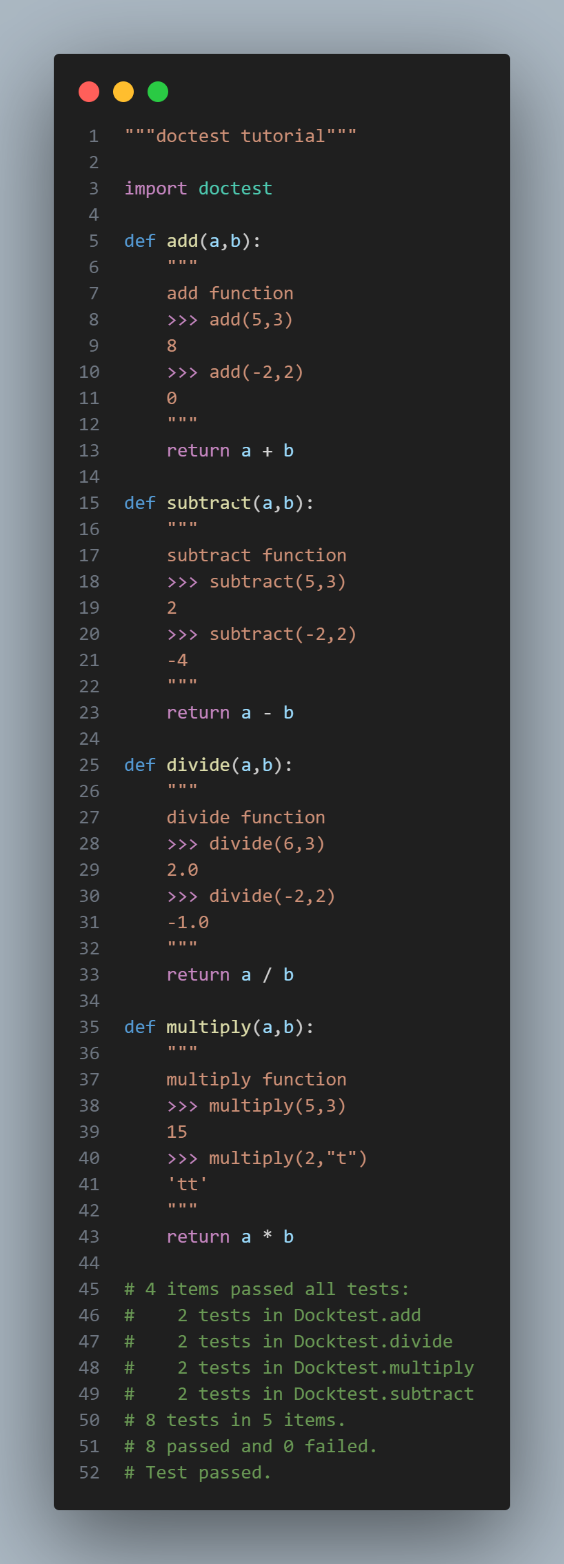
**Doctest :**

میایم برای هر فانکشنمون یا کلاسمون تست مینویسیم که مشخص بشه نتیجه چی باید بشه :

1. از داک استرینگ ها استفاده میکنیم
2. از <<< برای نوشتن دستوراتمون توی همون داک استرینگ استفاده میکنیم.
3. از پاسخ دستور **python -m doctest example.py** متوجه میشیم که خطایی نداشته یا داشته، اگه جوابی نداد یعنی اوکی بوده.

3.1 – اگه دستور **python -m doctest -v example.py** رو بزنی جزییات بهت میده.

در این دستورات -m یعنی ماژول و -v هم یعنی verbose



**Unittest :**

قبل از ورود به خوده یونیت تست ها باید با assertآشنا بشیم، و بدونیم که چی و چجوری هستند. اسرت ها برای تست کردن برنامه توسط خوده برنامه نویس به کار گرفته میشوند و شما نباید از اونها توی برنامه اصلی که کاربر قراره ببینه استفاده بکنید(اینجا از شرط ها استفاده کنید.)

اسرت ها برای ما میتونن مثل شرط ها عمل کنند ولی محدودت تر ، به مثال زیر دقت کنید.

میخوایم که یک تخفیف اعمال کنیم ، مشکل اینجاست که اگر تخفیف زیاد باشه ، فروشنده یه چیزی بدهکار میشه به خریدار و ما باید این رو اول با اسرت ها چک کنیم و بعدش با شرط هندلش کنیم ، به شکل زیر :

****

که در اینجا اگر کد را ران کنیم به این ارور میخوریم :

AssertionError: Sorry

چون مقدار تخفیف بیشتر از قیمت محصول بوده برای همین باید با شرط اوکیش کنیم.

نکته : هیچ وقت از اسرت برای اعتبار سنجی ها استفاده نکنید چون با کامند زیر ، کلا اسرت ها غیر فعال میشوند:

x = 4

assert x==5

print("x is equal 5")

مثلا اینجا باید به ما خطا بده ، که خب اگه عادی رانش کنی ، درسته و خطا میده ولی اگه من این دستور python -O example.pyرو بزنم خطایی نمیده :



یه حالت دیگه هم داره که همیشه درست برات برمیگردونه اونم وقتیه که شرطت رو توی تاپل بزاری مثلا تو مثال بالا بنویسی: assert (x==5 ,)

نکته دوم : اگه از اسرت ها استفاده کردی، همیشه حالتی رو در نظر بگیری که یه جوری به ارور بخوره و ازش بیاد بیرون، وگرنه اگه Trueباشه کلا نوشتنش فایده ای نداره.

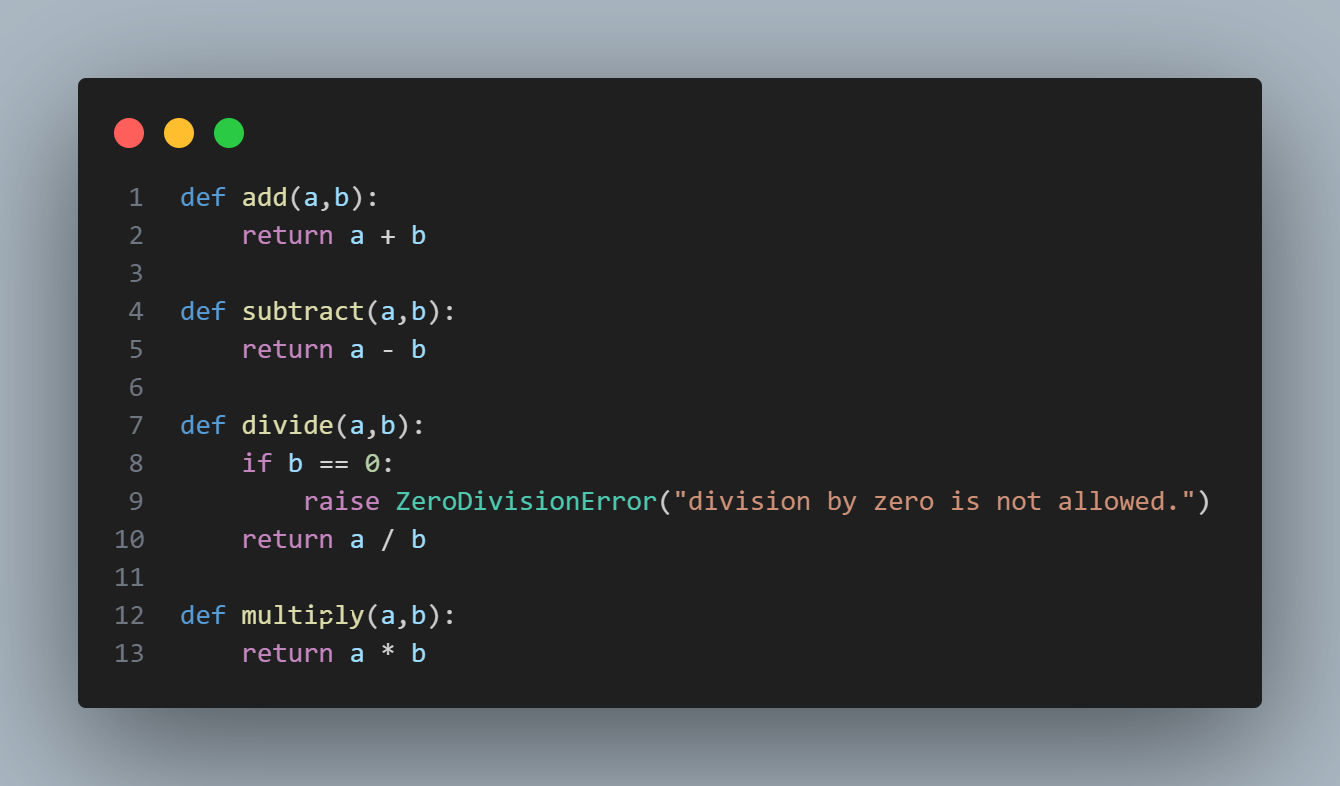
پس از اسرت ها برای وقتی استفاده میکنیم که دنبال یک باگی توی برنامتی نه جای دیگهههههه.

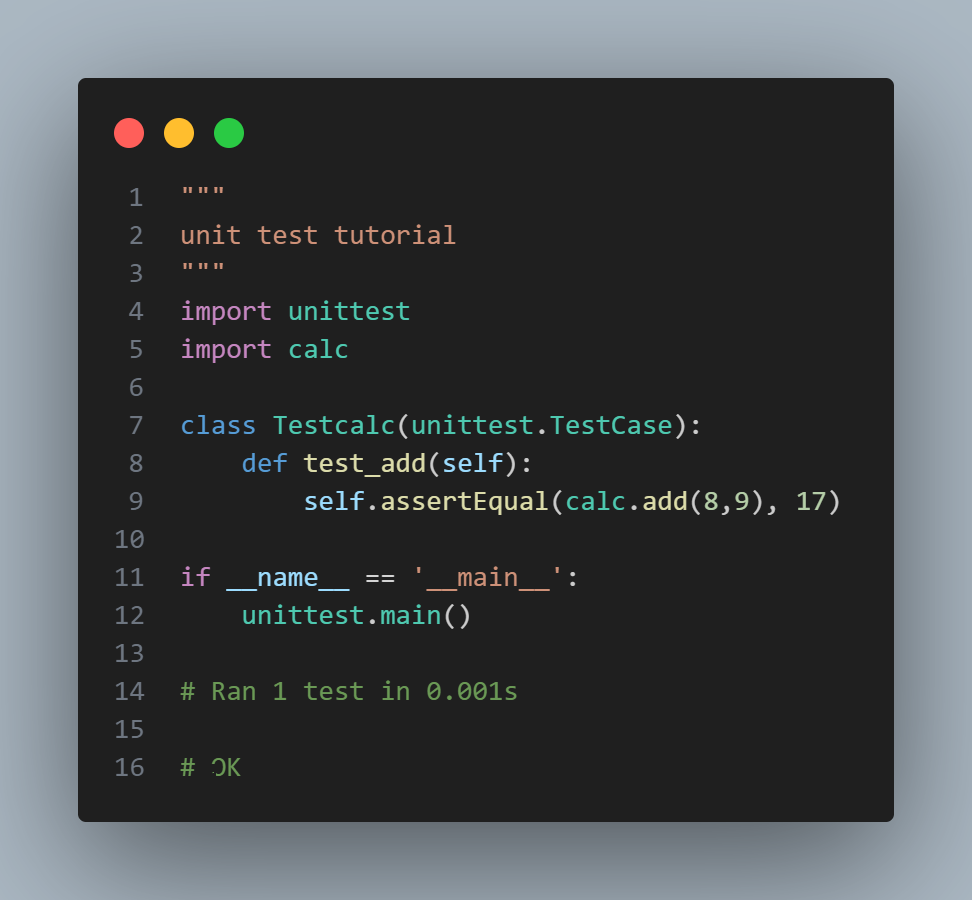
**خب حالا میخوایم با unittest آشنا بشیم ، این ماژول اولین بار با تاثیر از JUnit زبان جاوا ساخته شد. ماژول unittest مجموعه ای وسیع از ابزارها را برای ساخت و اجرای تست ها فراهم می کند.**

خیلی بهتر از قبلیه و کلا تست هارو توی یک فایل جدید میسازیم. جزء ماژول های داخلی پایتون هستش و ما کلاس Testcaseرو صدا میزنیم و ازش استفاده میکنیم.

برای اسم فایل جدیدت به این صورت عمل میکنیم : test\_name.py که بجای بایدname اسم فایلی که میخای تستش کنی باید باشه.

مثال : عکس صفحه بعد برای فایل test\_calc.pyهست و پایینی هم خوده فایل calc.py





اسم فانکشن ما حتما باید رو داشته باشه و سعی کنید مثل همین عکس اسم گذاری کنید چون قرارداده.

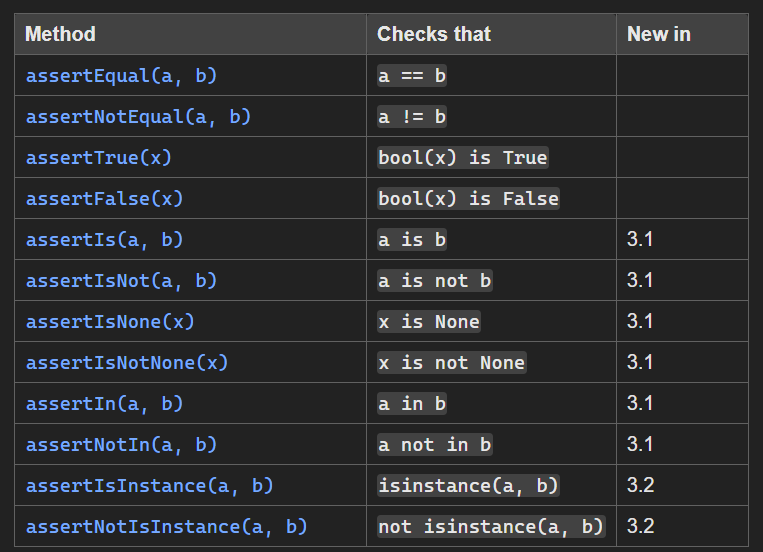
هم میتونی رانش کنی تو خوده ادیتورت هم با استفاده از کامند های زیر :

python -m unittest -v test\_module

یا

python -m unittest discover

این یه سری از متود های مهم یونیت تست هستش ، ولی بازم هست مثلا برای مدیریت اکسپشن ها از متودش فرق میکنه...



و در آخر تستی که برای calc.py فایل نوشتیم :



Fixtures : با استفاده از fixtureها میتوانید قبل و بعد از اجرا شدن تست‌ها کارهایی خاصی مثل ساخت آبجکت را برنامه ریزی کنید.

مثلا تو این مثال ما دوبار یک کد رو تکرار کردیم که خب هیچوقت توی برنامه نویسی ما نباید کد تکراری بنویسیم توی برنامه هامون.

اینجا برای تست برناممون باید instanceمیساختیم برای تست هر تابع که اصلا بهینه نیست، پس از کمک میگیریم :

خوده فایلی که میخوایم تستش کنیم: (یکم ریزه ، یه یونیت تست معذرتتت)



قبل :



بعد : اینجا چنتا متود داریم ، که مهم ترین هاش setup, teardown, setUpClass, tearDownClass, setUpModule, tearDownModule

: Setupقبل از هر **واحد تست** ران میشه. حالا توی بدنه این متود میخای کدهایی که قبل از هر واحد تست ران بشه مینویسی.(باید پراپرتی طورش کنی با استفاده از self که تو کله کلاس در دسترس باشه.)

: tearDownبعد از هر واحد تست ران میشه.

: setUpClassقبل از هر **کلاس تست** ران میشه که باید @classmethodهم باشه.

: tearDownClassبعد از هر کلاس ران میشه که باید @classmethodهم باشه.

برای ماژول هم خودت فهمیدی چجوری ران میشه دیگه.....



یادت باشه که تست هاتو میای مینویسی به صورت منطقی و بعدش اون جاهایی که faliedشد رو میری توی کد اصلیت درست میکنی، نه اینکه بیای تستات رو با کدت منطبق کنی و دست به تستت بزنی، و باید بدونی که این واحد تست هایی که مینویسی نباید اصلا بهم ربطی داشته باشند( isolate باشند).

نکته : یه مبحثی هستش به نام nosetest ، که اومده کاره تست نویسی مارو راحت تر کنه. مثلا شما نیازی نداری چیزی ایمپورت کنی به جز فایلی که میخای تستش کنی، و دیگه اون متود ها مختلف اسرت رو نداری و با کلمه کلیدی assert خالی تست میکنی. دوست داشتید بخونید [راجبش.](https://nose.readthedocs.io/en/latest/usage.html) ورژن 2 هم داره.

**: pytest**

این از یونیت تست هم بهتره و باهم سازگارند، یعنی اگه یونیت تستی نوشتی باشه ولی [pytest](https://docs.pytest.org/en/7.1.x/contents.html) میتونه اون هارو هم ران بکنه ، اولش نصبش میکنی با پیپ.

pip install -U pytest

بعدش با دستور pytestمیاد هر فایلی که با شروع شده رو انتخاب میکنه و توش دنباله متود هاو کلاس هایی با میگرده و رانش میکنه، (هم یونیت تستی هم nose)

خوبیش اینه خیلی خوب و تروتمیز بهت میگه چیکار داشته میکرده و خوب لاگ میندازه برات دستورات دیگه ای هم داره برای ران کردنش مثله :

pytest -v .\test\_fixtures.py

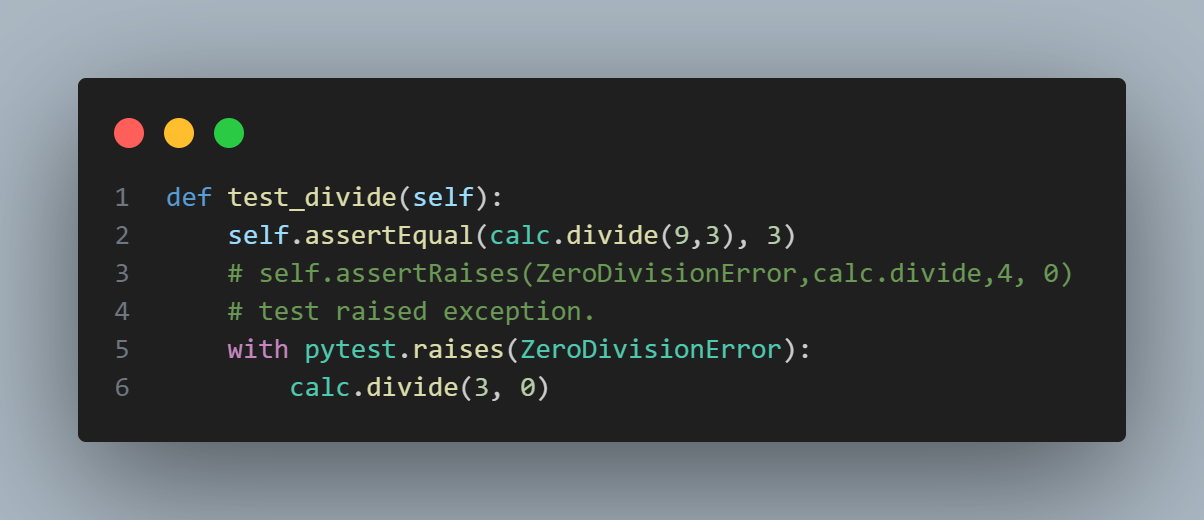
pytest .\test\_fixtures.py

Fixtures در pytest : نسبت به یونیت تست خیلی راحت تره، اینجا اول ایمپورتش میکنی بعدش به صورت دیکراتور اون تابعی که میخای رو مینویسی،و به عنوان یه آرگومان به اون تابع اصلیت پاسش میدی . اینجا با یه تابع هم کارت راه میوفته و لازم نیست دوتا داشته باشیم مثل قبلی(set , tear)



بعد ازyield هرچی بنویسی به عنوان teardown استفاده میشه و دیگه اینجا نیازی به انواع متود های مختلف یونیت تست نداری و فقط از کلمه کلیدی assert استفاده میکنی.

و برای raise errors ها هم به شکل زیر عمل میکنیم: (به صورت context manger)



با دستور زیر هم میتونیم نتیجه این تست هارو هم به صورت فایلی ذخیره کنیم (از نسخه 6 به بالا دیگه نمیشه)

pytest test\_example.py –resultlog=result.log