Szoftvertesztelés Pythonban

Számológép projekt

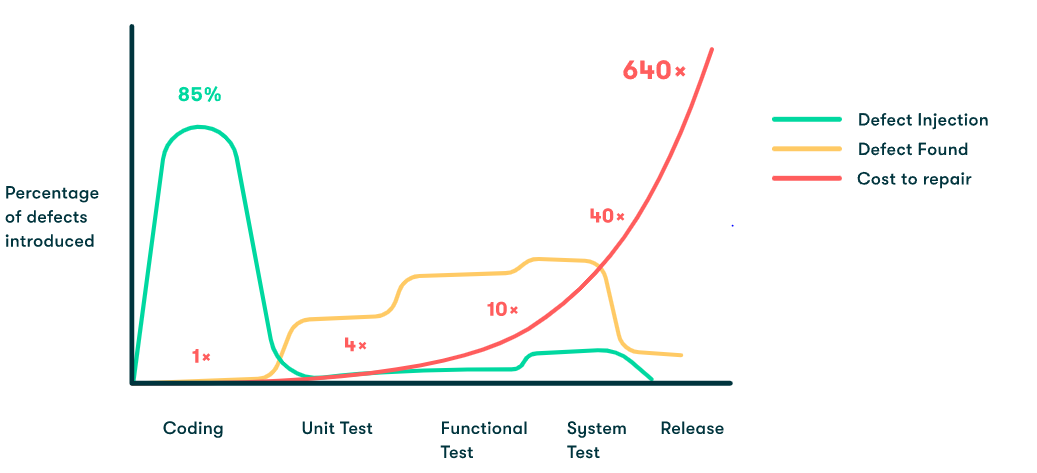
Szabó Huba

***Bevezetés:***

Napjainkban a világ fejlődése következtében a digitális eszközök világa is elért egy magas szintet. Ezen gyors fejlődés miatt az emberek egyre igényesebbek lettek melykövetkeztében azok a termékek(mi esetünkben szoftverek) amelyeket megvásárolnak minél jobb, magasabb elvárásoknak kell megfelelnie. A jó működés érdekében bevezették a szoftverek tesztelését.

A tesztelések bevezetésére azért volt szükség, mivel egy megírt szoftver sosem biztonságos, rengeteg hiba található benne. A hibák emberi szemmel nem mindig felismerhetőek, viszont a tesztelés segítségével több hibát ki lehet kiszűrni.

Azt tudni kell, hogy egyetlen szoftver sem lesz tökéletes. Attól mert a fejlesztők, teszterek véghez visznek teszteket nem feltétlen lesz biztonságos a szoftver, mert egyszerűen annyi esetet kellene letesztelni, hogy effektív idő és pénz sincs rá. Viszont ha egyáltalán nem használják a teszteléseket, és kikerül az emberiség kezébe, akkor a szoftver valószínűleg nem megfelelő működést fog mutatni ami által az emberek nem fogják használni.

A szofvertesztelés nagyon fontos egy termék gyártásában, hiszen a hibák kiküszöbölésében segít. Ez azért fontos mert ezek időben való felfedezésével, csökkenthetjük azok kijavításának költségeit.

1.abra

***Mi is a szoftvertesztelés?***

A **szoftvertesztelés** egy rendszer vagy program kontrollált körülmények melletti futtatása, és az eredmények kiértékelése. A hagyományos megközelítés szerint a tesztelés célja az, hogy a fejlesztés során létrejövő hibákat minél korábban felfedezze. A tesztelői munka egyre inkább eltolódik a fejlesztők és a döntéshozók információkkal való támogatásának irányába.

***Szoftvertesztelés definiciója röviden?***

* Felhasználhatósági szemszög: "Alkalmasság a felhasználó által szánt célra."
* Tesztelői szemszög: "Egyezés a specifikációval."

***Szoftvertesztelés célja?***

A klasszikus szoftvertesztelés célja a **szoftverhibák felfedezése**. A fejlesztésnek minél korábbi szakaszában derül fény egy hibára, annál olcsóbb annak korrigálása. Újabb keletű elvárás a **szoftverminőség mérése**.

***Nincs 100%-ban hibátlan szoftver!!***

***Tesztelési módszerek, technikák***

A tesztelési módszereket csoportosítani tudjuk több féle szinten annak érdekében, hogy milyen szinten használjuk ezeket, valamint.

A megfelelő minőséget két szempont biztosítja:

* A validációs tesztelés (validation) a felhasználhatóságot vizsgálja: alkalmas-e a

rendszer a felhasználó által szánt célokra. A megfelelő terméket készítjük el?

* A hiányosság tesztelés (verification) a tesztelői szemszöget képviseli és azt vizsgálja,

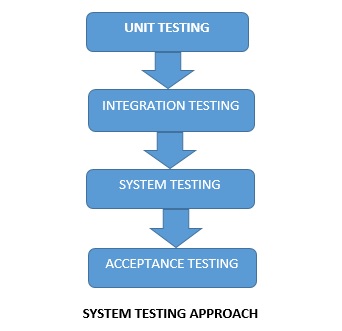
hogy minden a specifikáció szerint működik-e. Ennek a tesztelésnek az a célja, hogy

felfedezzük azokat a hibákat, amelyek a szoftver helytelen viselkedését okozzák, azaz

a működése nem felel meg a specifikációjának. A terméket jól készítjük el?

***Tesztelési szintek:***

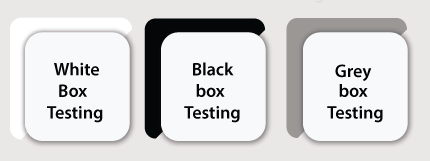
* ***Komponensteszt:***
  + Ezen a szinten a rendszer egy komponensét teszteljük önmagában.
  + Az egységteszt vagy unit-teszt a metódusokat teszteli. Megvizsgálja, ha adott paraméterekre az elvárt eredményeket kapjuk. Fontos, hogy magának az egységtesztnek ne legyen mellékhatása.
* ***Integrációs teszt:***
  + A komponensek együttműködésének tesztjeit foglalja magába, az összeillesztése során keletkező hibákat keresi.
* ***Rendszerteszt:***
  + A rendszerteszt a már egységeé egészként, szoftverterméket (összes komponens) teszteli.
  + Megvizsgálja, ha a rendszer megfelel a rendszertervnek és a követelmény, illetve funkcionális specifikációnak.
* ***Felhasználói elfogadási tesztelés:***
  + A felhasználói (alfa, béta, felhasználói átvétel, üzemeltetői átvétel) tesztek során a felhasználók a kész rendszert tesztelik, meggyőződve, hogy a termék megfelel az elvárásoknak, hibamentesen és biztonságosan használható éles körülmények között is.



2. ábra

***Tesztelési technikák:***

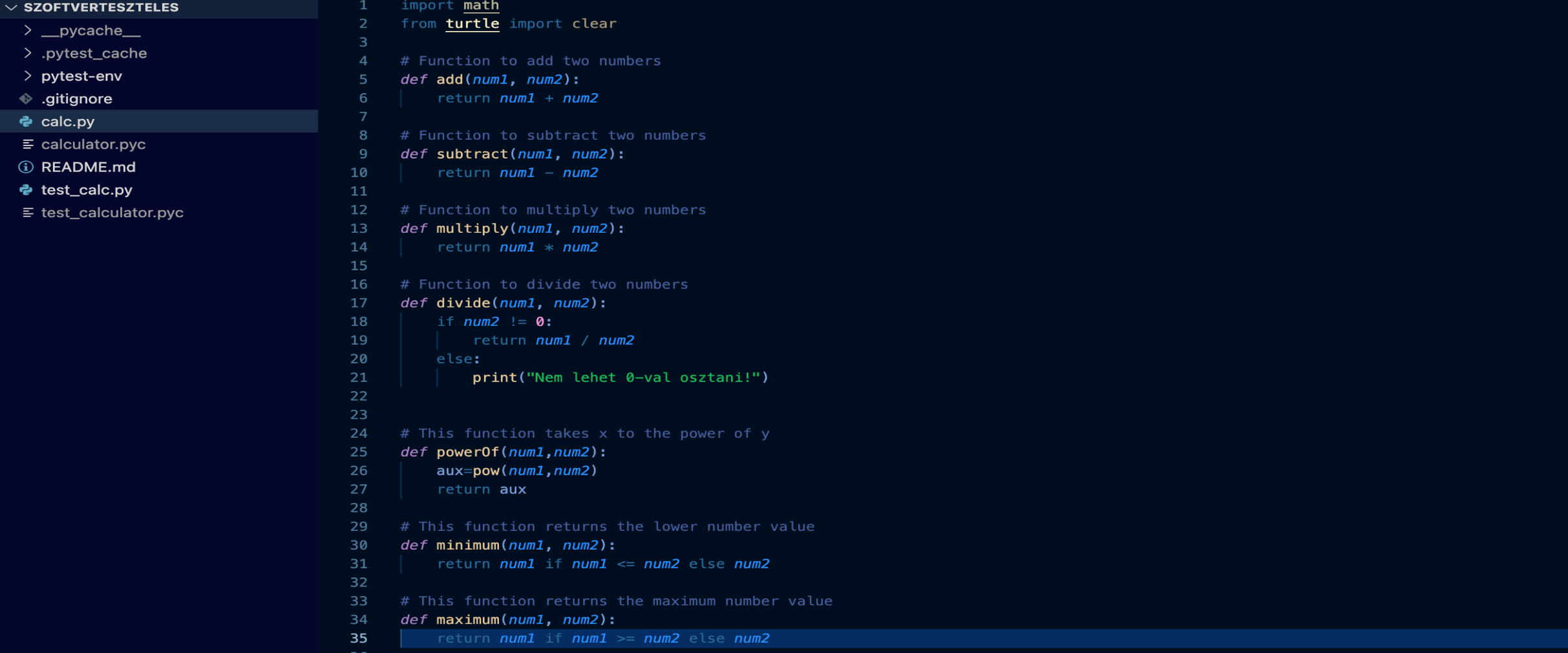
* ***Fekete doboz tesztelés:***
  + A szoftver teszteli a rendszer felépítésének és a belső modulok szerkezetének ismerete nélkül.
  + Célja a hiányzó funkcionalitások, teljesítmény és inicializálási problémák megtalálása.
* ***Szürke doboz tesztelés:***
  + Átmenet a fekete illetve a fehér doboz tesztelés között, a kettő kombinációja
  + Az alkalmazás belső felépítését csak részben ismerjük.
* ***Fehér doboz tesztelés:***
  + A szoftver belső felépítését teszteli.
  + A fekete dobozos tesztelés ellentétje.



3. ábra

***Szoftver projekt bemutatása:***

A dokumnetáció további részében a az általam előkészített kicsi python szoftver projekt pár metódusának a tsztelését szeretném bemutatni.

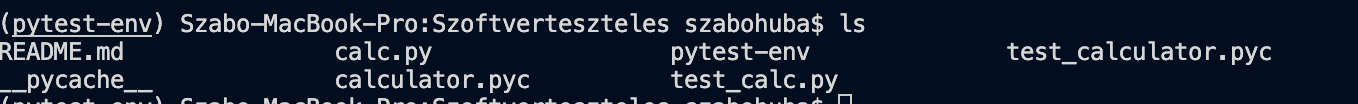
A projekt megvalósítására Visual Studio Code fejlesztői környezetet haszáltam. E fejleszői környezetben a szoftvertesztelés alapjait sajátítottam el, unit teszteket irtam egy számológép függvénymüveleteire.(4. ábra/5. ábra)

4. ábra

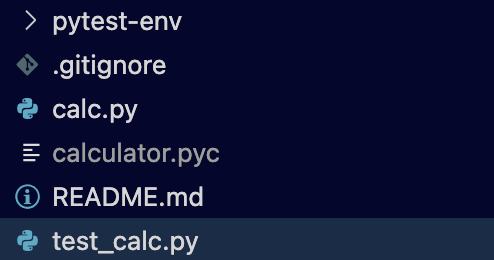


5. ábra

A tesztelés egyszerűsége, valamint a jó átláthatósága érdekében Python programozási nyelvet választottam az implementációhoz, ennek pedig a legujabb verzióját(3.7.4). A tesztek megvalósítására pedig az ajánlott pytest(7.1.1)-re esett a választásom.

A rendezettség, illetve a biztonság kedvéért egy virtuális környezetben valósítottam meg e kis projektemet.(6,7. ábrák)

6. ábra



7. ábra

A másik ok amiért a vituális környezetet vákasztottam az az, hogy a Python nem a legjobb a függőségek kezelésében. Ha nem vagyunk specifikusak akkor a pip az összes külső csomagot amit instalálni akarunk a site-packages mappaba instalálja a fő Python mappába.

* $ mkdir pytest\_demo
* $ cd pytest\_demo
* $ python3 -m venv pytest-env

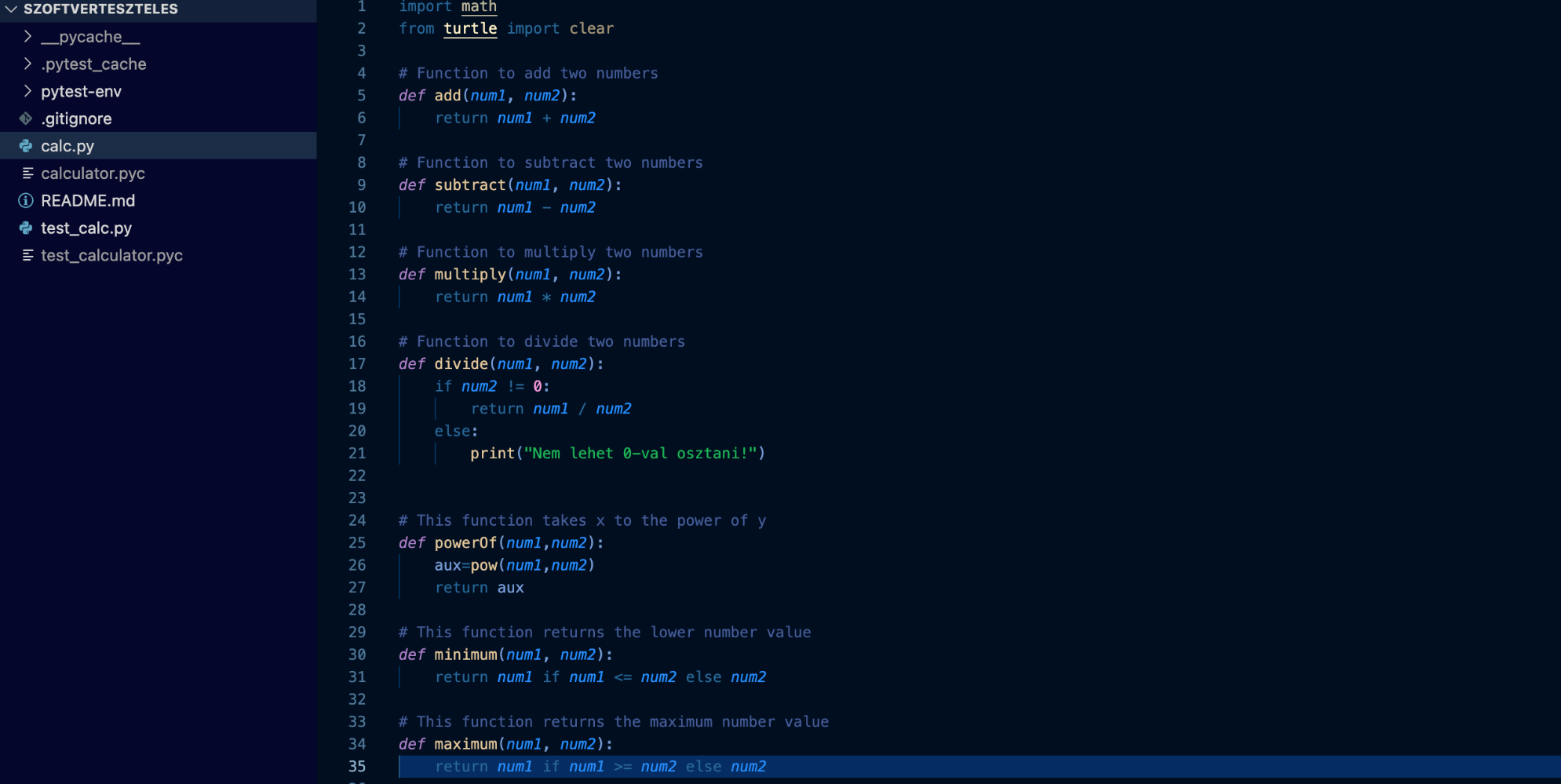
Virtuális környezet aktiálása és haszálata:

* $ source pytest-env/bin/activate

Második lépésben instaláltam a pytestet a projektben lévő függvények tesztelésére. A pytest az egy tesztelési framwork ami a python specifikus, az az python könyvtárcsomagra alapszik.(megjegyzés-a pip csomag menedsszer 22.0.4 es verzióját használtam.)

* pip3 install pytest

Miután minden kellék teleőítve lett, illetve létrehoztuk a szükséges python fájlokat amiket tesztelni akarunk, nekiállhattunk a tesztfájlok létrehozásának.



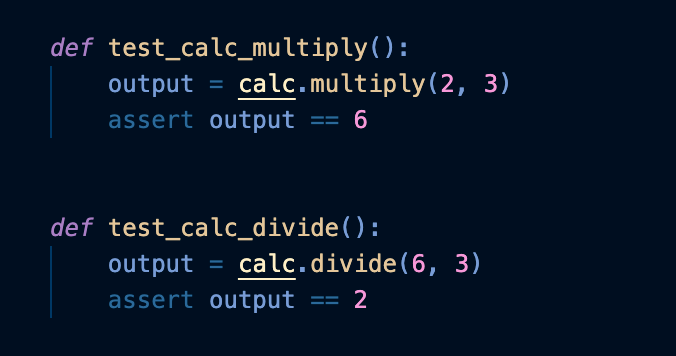
8. ábra

A calc.py python fájl egy számológép müveleteit tartalmazza:

* add(összead)
* subtract(kivon)
* multiply(szoroz)
* divide(osztás)
* powerOf(hatványozás)
* minimum, maximum(két szám közül a kisebb, illetve nagyobb)

A fentebb felsorolt függvényeket szerettük volna tessztelni pytest segitségével. Ennek megfelelően létre kellett hoznunk egy python jájlt aminek a neve a „calc.py” – t kellett tartalmaznia, kiegészitve a „test” – kulcsszóval. Erre azét volt szükség, mivel a környezetünk így automatikusn felismeri, hogy ebben a fájlban a calc.py függvényeit teszteljük.

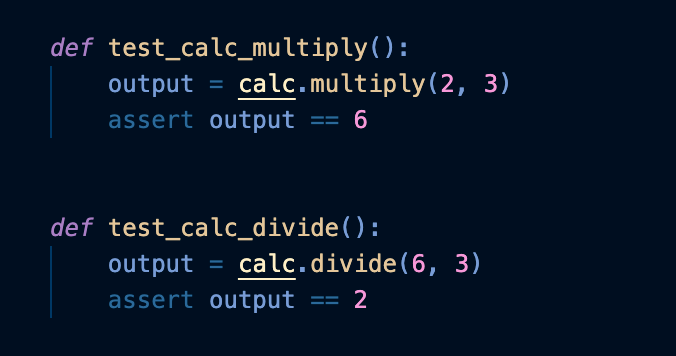
Miután létrehoztam a test\_calc.py fájlt különböző teszteket írtam minden függvényre amit a szamológép fájlba implementáltam. (9. ábra)



9. ábra

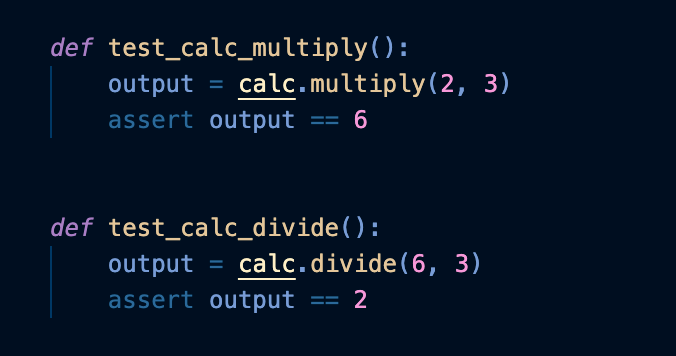
A továbbiakban a „test\_calc.py” nevű mappába beimportoljuk az eredeti „calc.py”-t, hogy elérjük a tesztelésre szánt függvényeinket. Ezek után a 9. ábrán láthatjuk a tesztelésre megirt függvények szerkezetét. A fentiekben(9. ábra) két tesztmetódust láthatunk implementálva(szorzás, osztás). Ezeknek a szerkezete nagyon fontos.

A metódus nevéből ki kell derülnie (a mi esetünkben), hogy melyik müveletet fogjuk tesztelni, valamint, hogy teszteljük-e. A következökben egy változóban elmentjük a kivánt függvény visszatéritési értékét.(10. ábra)

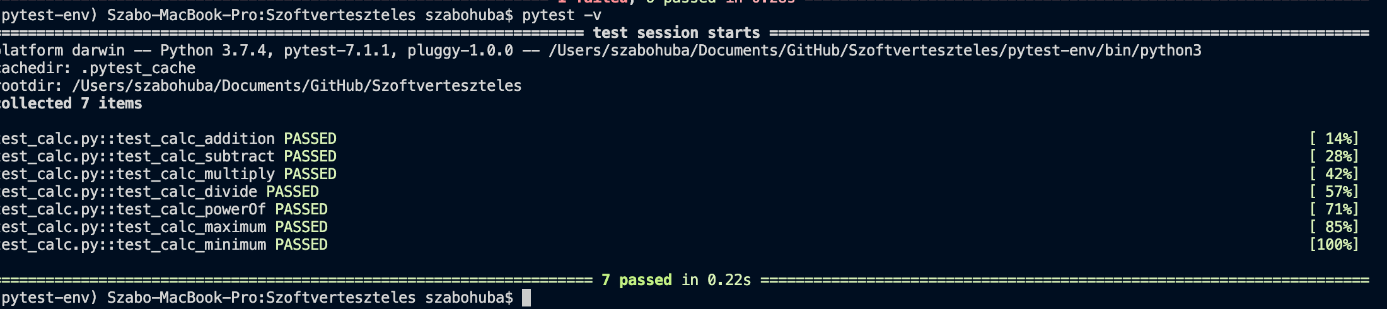


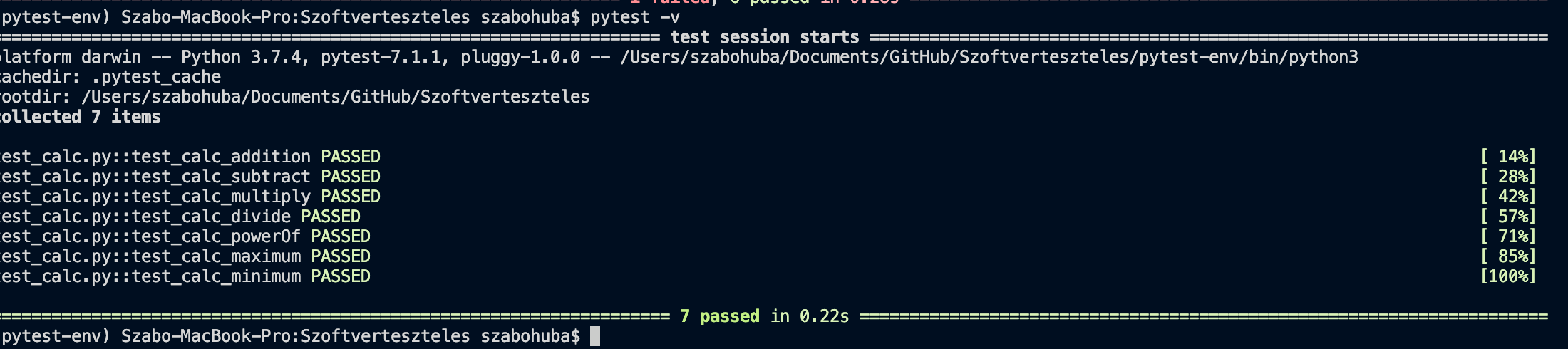
10. ábra

A továbbiakban talán a fentebb sor jön a megírt tesztfüggvényünknek.(11. ábra)



11. ábra

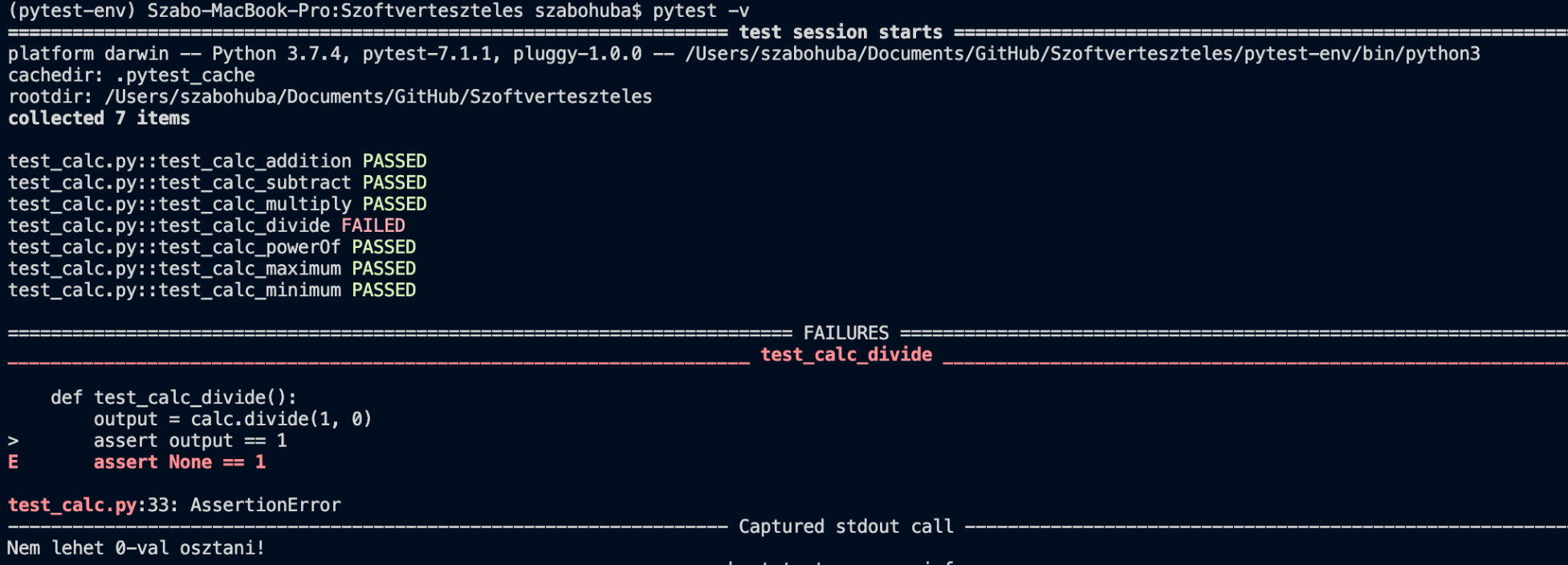
Ebben a sorban történik meg a kiértékelése az eredményünknek. Az outputban találjató az álytalunk kívánt számítás eredménye, amit egyenlővé teszünk az elvárt kimenettel a fenti képen (11. ábra) ez a „6”. Visual Studio Code-ban „pytest -v” paranccsal(12. ábra) tudjuk lefuttatni a tesztelést. Ha a 11. ábrán látható egyenlőség teljesül akkor a 12. ábrán látható eredmény kapjuk terminélra.

 12. ábra

13. ábra

A 12,13 as ábrákon egy sikeresen lefutott, hibaüzenetek nélküli terminálra küldött eredményt kaptunk. Látható, hogy melyek azok a tesztek amelyek helyesek, valamint az is, hogy mennyi idő alatt futott ez le. A jobb oldalon a százalékok pedig felosszák a 100 % feladatot, kisebb részekre, és minden teszt után kiíródik, hogy a teljes munkának hány százaléka az a függvény.

Ha esetleg valamelyik tesztünk nem teljesül akkor hibaüzenetet kapunk a terminálunkra. Futtatás után a függveny neve mellé odakerül, hogy „FAILED”, valamint lentebb a hibaüzenetet láthatjuk, hogy melyik sorban van. (14. ábra)

 14. ábra

A 14. ábrán megfigyelhetjük, hogy a beépített hibaüzeneten kívül az is megjelenik ha valami le van kezelve a tesztelt függvényben „0- val nem lehet osztani” (ez az általam irt lekezelés).

Könyvészet:

* <https://docs.google.com/document/d/1FUUzWrYopCJinb09hKW8EGxBSl7KeYKaA3f8zN8kQDY/edit#>
* <https://docs.pytest.org/en/6.2.x/>
* <https://realpython.com/tutorials/testing/>