Szoftver tesztelés

A pytest

(dokumentáció)

Tartalomjegyzék

[1. Bevezető 4](#_Toc98405483)

[2. Az adatról 5](#_Toc98405484)

[3. Tesztelési módszer 6](#_Toc98405485)

[4. Implementálás 7](#_Toc98405486)

[5. Összefoglalás 8](#_Toc98405487)

# Bevezető

--általánosan a tesztelésekről

# Működés

Ahhoz, hogy eljussak a teszteléshez előbb meg kellett ismerkednem a kódbázissal. Rendelkezésemre állt employee\_manager és egy relations\_manager forráskód. A következőkben ezeknek a funkcionalitásait fogom ismertetni.

A relations\_manager tartalmaz egy osztályt (RelationsManager) és ebben van néhány függvény, az init fügvényben található egy employee lista benne 6 darab employee-val (alkalmazottal). Ahhoz, hogy ez megvalósítható legyen, szükség van egy employee osztályra, ami tartalmazza egy alkalmazott adatait, és ezt az osztályt deklarálni kell a relations\_manager forráskód elejére. Ebben a függvényben még található egy struktúra ami reprezentálja az employee listában levő csapatokat. Minden csapatnak van egy vezetője (leader) és ebben az esetben 2-2 csapat tag. Ezen kívül még található a RelationsManager osztályban 4 függvény, amelyek a következők:

* is\_leader(): amely visszatéríti, hogy egy alkalmazott vezetője-e egy csapatnak vagy sem; egy bool típust térít vissza.
* get\_all\_employees(): visszatérít egy listát, ami az alkalmazottakat tartalmazza.
* get\_team\_members(): megnézi, ha egy alkalmazott vezető pozicióban van és ha igen akkor vissza adja a csapat tagjait.
* get\_employee\_name\_and\_birth(): bejárja az employee listát és vissza adja egy vezető employee nevét és születési dátumát.

Az employee\_manager forráskód tartalmaz egy EmployeeManager nevű osztályt és egy fő(main) függvényt. Az osztály megkapja az init függvényben, ami egy konstruktorként szerepel, paraméterként a RelationsManager osztályt. Ebben az osztályba további 2 függvény található:

* calculate\_salary(): kiszámolja egy adott alkalmayott fizetését, ha az alkalmazott egy vezető akkor a fizetéséhez hozzá lesz adva bónusz minden tag után aki a csapatában van. Miután kiszámolta a fizetést visszatéríti ennek az értékét.
* calculate\_salary\_and\_send\_email(): kiszámítja egy alkalmazott fizetését meghívva az előző függvényt és kiír a terminálra egy üzenetet.

A fő függvényben meg le van kérve az alkalmazottak listája és ki van iratva, meg van vizsgálva egy adott alkalmazott csapattagjait, majd ki van szmolva a fizetése úgy, hogy meg van hívva a calculate\_salary\_and\_send\_email() függvény.

Tehát a forrás kódok alkalmazottakról tárolnak el információkat és különböző műveleteket segítségével meg lehet vizsgálni ezeknek az információknak a helyességét.

# Metodológia

A projekt célja, hogy a 2 fejezetben tárgyalt adathalmazra sikeresek legyenek a tesztek. Elvégezzem a következő teszteket:

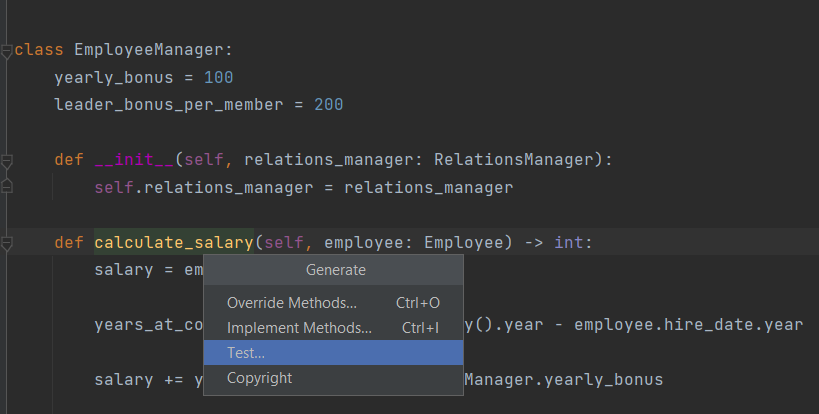
1. relations\_manager állományhoz tartozó tesztek:
2. Megnézni, hogy az adathalmazban van olyan csapatvezető, akit John Doe-nak hívnak és a születési dátuma 31.01.1970.
3. Megnézni, hogy ha John Doe csapatában szerepel Mzrta Torkelson és Jettie Lynch nevű alkalmazottak.
4. Megbizonyosodni, hogy Tomas Andre nincs John Doe csapatában.
5. Le ellenőrizni, hogy Gretchen Walford alap fizetése egyenlő 4000$.
6. Megbizonyosodni, hogy Tomas Andre nem csapatvezető.
7. Megbizonyosodni, hogy Jude Overcash nem szerepel az adatbázisban.
8. employee\_manager állományhoz tartozó tesztek:
   1. Le ellenőrizni egy alkalmazott fizetését aki nem csapatvezető és az alkalmazásának a dátuma egyenlő 10.10.1998 és az alapfizetése egyenlő 1000$. Megbizonyosodni, hogy a kapott eredmény 3000$.

# Eszközök

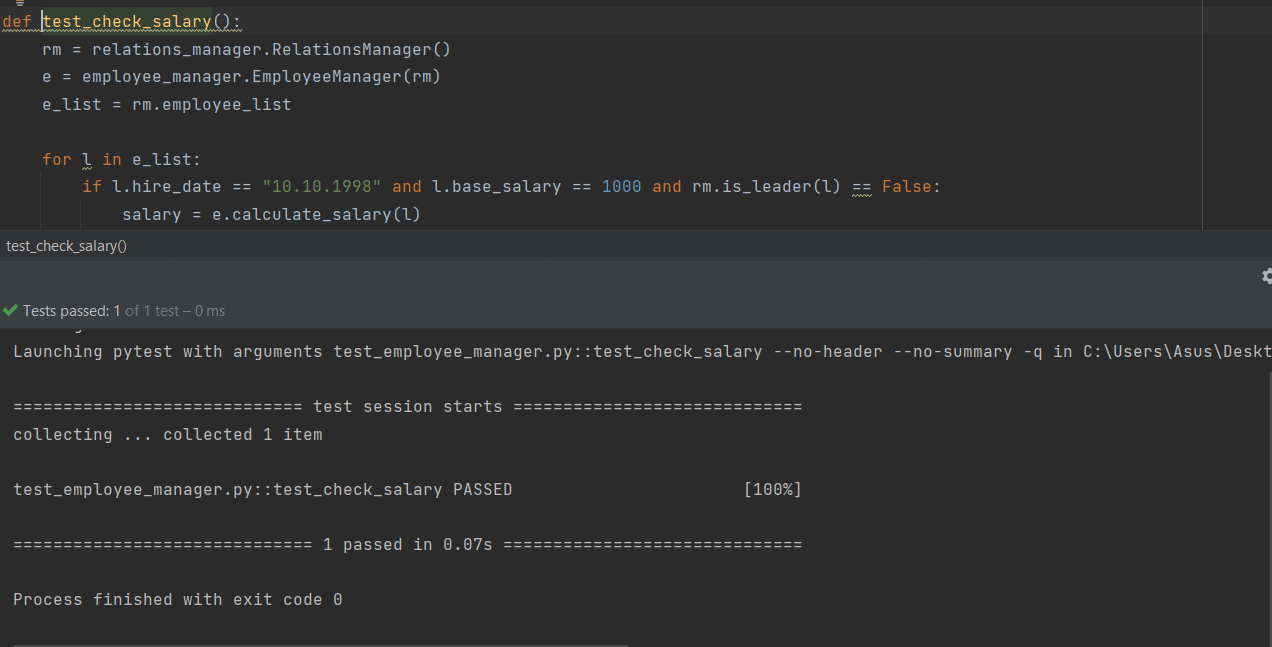
Az én választásom a python programozási nyelv alatti tesztelés ezért meg kellett vizsgálnom, hogy milyen lehetőségeim vannak. Pythonban lehetőség van unittest és pytest elvégzésére, ezek közül az utóbbinak számos előnye van.

* Egyik leg elterjedtebben használt nyílt forráskódú tesztelési keretrendszer Python 3.5 alatt.
* Támogatja a unit teszt lehetőségét, továbbá funkcionális és API tesztet is egyaránt.
* Engedélyezi az összetett és egyszerű teszteket.
* Könnyen terjeszthető különböző bővítmények segítségével.

A projektem során én is a pytest-et választottam az előnyeinek köszönhetően. Ennek a telepítése nem igényelt bonyodalmakat, mert Pycharm fejlesztői környezetet használtam és a környezet telepítése után könnyen lehet könyvtárakat telepíteni a környezetben, és így egy pytest nevű könyvtár telepítésével megoldódott a telepítés.

 Ahhoz, hogy tesztet írjunk, függvényeket kell létrehozzunk és a függvények nevét test szóval kell kezdeni. A függvények végén „*assert*” kulcsszóval tudjuk leellenőrizni, hogy a kívánt eredményt kaptuk. Úgy is tudunk tesztet írni, hogy állományban ahol a tesztelendő függvény található jobb kattintás segítségével megjelenő menüből kiválasztjuk a megfelelő opciót (*1. ábra*) és ezzel automatikusan létrejön egy teszt függvény.

1. ábra: Automatikus test létrehozás

Amiután lefuttatunk egy tesztet, a terminálra ki fogja írni, hogyha egy teszt sikeres lett vagy sem, amelyet a *2. ábra* szemlélteti.

2. ábra: Pytest példa

Hogyha összehasonlítjuk, hogy a pytest miben különbözik a unit test-től, akkor a *3. ábra* alapján egyértelmű, hogy első sorban a függvényeket egy osztályba kell megírni, ehhez szükséges egy TestCase amit a fájl elején kell importálni. Másod sorban a futtatás során a pytest-el ellentétben csak annyit ír ki, hogy hány tesztet futtatunk és sikeresek ezek vagy sem.

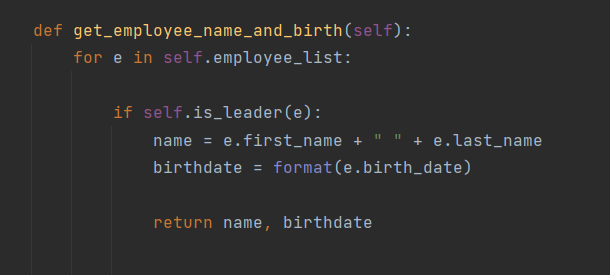
3. ábra: Unit test példa

# Tesztelés

---hogyan alkalmaztam a tesztelést, kódrészletek

A következőkben szemléltetni fogok néhány tesztet, amit én írtam meg a projektem során. Első sorban relations\_manager állományhoz tartozó teszteket végeztem el.

1. Alábbiakban látható az első teszt, amelyben szükség volt egy relations\_manager objektumre, majd le kellett kérjem ennek a nevét és születési dátumát. Ehhez írtam egy függvény, amely a *4. ábra* szemléltet. Amikor megkaptam a nevet és dátumot akkor assert –el megnézem, hogy megfelelnek a változók a követelményeknek.

def test\_get\_employee\_name\_and\_birth():  
 e = relations\_manager.RelationsManager()  
 name, birthdate = relations\_manager.RelationsManager.get\_employee\_name\_and\_birth(e)  
  
 assert name == "John Doe"  
 assert birthdate == '1970-01-31'

4. ábra: Név és születési dátum lekérése

1. A következő teszthez le kell kérnem azoknak az alkalmazottaknak az azonosítóját, amelyek John Doe csapatában vannak, ezután bejárom az alkalmazott listát és megkeresem a megfelelő alkalmazottat és kimentem az azonosítóját, majd ugyan csak assert –el le ellenőrzőm, hogy ez az azonosító nincs benne az előzőleg le kért azonosítók listájában.

def test\_if\_not\_team\_member():  
 e = relations\_manager.RelationsManager()  
 member\_ids = e.teams[e.employee\_list[0].id]  
 for l in e.employee\_list:  
 if l.first\_name == "Tomas" and l.last\_name == "Andre":  
 e\_id = l.id  
 assert e\_id not in member\_ids

Az employee\_manager állományhoz tartozó tesztek közül a *2. ábra* és *3. ábra* szemlélteti az egyik tesztet, amelyben meg kellett vizsgálni egy alkalmazott fizetését.

# Következtetések

---mit tudtam elérni, mit vonok le az egészből(mi jó, mi nem)