**Szoftver tesztelés projekt**

****

**Név:** Mikló József-Péter

**Szak:** Automatika és alkalmazott informatika

**Évfolyam:** IV. év

**Tantárgy:** Szoftver tesztelés

**Vezető tanár:** Dr. Szántó Zoltán

**Év:** 2024

Tartalomjegyzék

[Bevezető 2](#_Toc161930103)

[Kódbázis 2](#_Toc161930104)

[Tesztelés 4](#_Toc161930105)

# Bevezető

A szoftvertesztelés egyik elengedhetetneln gyakorlata a szoftverfejlesztésnek. A unit tesztelés egy szoftverfejlesztési módszer, aminek célja, hogy ellenőrizze az alkalmazás kisebb részeit, az egységeket. Ezek az egységek általában az alkalmazás legkisebb lehetséges funkcionális részei, például egy függvény vagy egy osztály metódusa. A unit tesztek automatizáltan futtathatóak, és segítenek megbizonyosodni róla, hogy az egyes egységek helyesen működnek.

A unit tesztek azért fontosak, mert ezek segítségével könnyen azonoíthatjuk és javíthatjuk a kódban felmerülő hibákat. A tesztek biztosítják, hogy a kód módosítása során ne kerülhessenek be hibák a kódba.

A unit tesztek felépítése három fő részből tevődik össze:

* előkészítés: inicializáljuk az egységet (osztályt) és a szükséges adatokat
* művelet: meghívjuk az egységünknek azt a funkcióját vagy metódusát, amelyet tesztelni szeretnénk
* ellenőrzés: ellenőrizzük, hogy az egységünk az elvárt eredményt adta-e a tesztelés során.

# Kódbázis

A projekt leírást *Dr. Szántó Zoltán* foglmazta meg egy online docs-ban[[1]](#footnote-1). A projektet egy “Software testing project” nevű github repository[[2]](#footnote-2) tartalmazza. A repository tartalmaz egy src mappát a forráskódokkal, egy test mappát a tesztkódokkal és egy docs mappát,ahol a dokumentáció található.

A kódbázis három osztályból, azokak adattagjaiból és metódusaiból, illetve két teszt file-ból tevődik össze. Az egész kódbázis Python programozási nyelvben lett megírva.

Az “employee.py” Python script tartalmaz egy Employee osztályt és azok adattagjait.

A “relations\_manager.py” script egy RelationManager osztályt tartalmaz, ahol az \_\_init\_\_ metódusban létrehozunk egy employee listát és megadjuk a csapatokat. Van egy függvényünk, ami egy employee-ról megállapítja, hogy leader vagy sem, egy következő függvény lekérdezi az összes employeet és egy másik függvény, ami visszatéríti egy leader-nek a csapattagjait.

Az “employee\_manager.py” file-ban egy EmployeeManager osztállyal találkozunk. Ez az osztály felelős az alkalmazottak fizetésének kiszámításáért és az e-mailben történő kiküldéséért. Az osztály \_\_init\_\_ metódusa egy RelationsManager típusú objektumot vár az inicializálás során. Ennek az osztálynak még két metódusa van: egyik ami vissatéríti egy alkalmazott (employee) fizetését és a másik metódus kiszámolja és kiírja és visszatéríti az alkalmazott nevével együtt a fizetését (egyfajta email-es visszajelzés az alkalmazott felé).

A projekt során ezen osztályok metódusait szeretnénk unit test-ekkel tesztelni, hogy megbizonyosodjunk a helyes működésükről. A “relations\_manager\_test.py” script-ben teszteljük a RelationsManager osztály metódusait. A kód elején importoljuk az src mappából szükséges modulokat, majd létrehozunk egy RelationsManager típusú objektumot, amire szükségünk lesz a tesztelés alatt.

Az első unit test-el azt szeretnénk ellenőrizni, hogy létezik-e John Doe nevű team leader, akinek a születési dátuma 31.01.1970. Hogy könnyen megértsük a unit tesztelés fázisait tekintsük a következő kódrészletet:

rel\_manager = relations\_manager.RelationsManager()

def test\_is\_leader():

    employee1 = Employee(id=1, first\_name="John", last\_name="Doe", base\_salary=3000, birth\_date=datetime.date(1970, 1, 31), hire\_date=datetime.date(1990, 10, 1))

    output = rel\_manager.is\_leader(employee1)

    assert output == True

A kódrészlet első sorában létrehozunk egy példányt a RelationsManager osztályból. Mivel erre minden unit test-nél szükség lesz, ezért globálisként hoztuk létre. A test\_is\_leader() függvény első soraiban létrehozunk egy John Doe employee-t, a megfelelő születési dátummal. A kódrészlet első sora és a test\_is\_leader() függvény első három sora a unit teszt előkészítési fázisa. A függvény negyedik sorában található a művelet fázis, ahol meghívjuk a RelationsManager osztály példánynak az is\_leader() metódusát a létrehozott employee-ra, majd ezt eltároljuk egy output változóban. A függvény utolsó sora pedig a unit tesztelés harmadik fázisa, tehát az ellenőrzés, ahol ellenőrízzük, hogy a művelet során az elvárt eredményt kaptuk vagy sem.

A második unit tesztel azt ellenőrízzük, hogy John Doe csapatában Myrta Torkelson és Jettie Lynch vannak. Ugyanúgy, mint előzőleg létrehozzuk John Doe employee-t, majd a RelationsManager osztály get\_team\_members() függvényével lekérjük John Doe csapatában szereplő embereket és kimentjük az output-ba. A kódolás során tudjuk, hogy Myrta Torkelson és Jettie Lynch 2 és 3-as id-val rendelkeznek, ezért az assertnél az output-ot ellenőrizzük, hogy a [2,3]-as id-t tartalmazza. Ugyanebben a függvényben tudjuk tesztelni azt is, hogy John Doe csapatában nem szerepel Tomas Andre, aki 5-ös id-val szerepel a kódbázisban. Tehát még egy assert-el megnézzük, hogy az output különbözik-e az 5-ös id-tól.

A következő tesztelő függvényben azt teszteljük, hogy Gretchen Watford alapfizetése 4000$. Ezt úgy tudjuk megállapítani, hogy lekérjük az alkalmazottak listáját, majd ezen egy for ciklussal végigmegyünk. A for ciklus elött az output változot False-ra állítjuk, majd a for cikluson belül egy ha igaz, hogy a listából egyik alkalmazott Gretchen Watford és a fizetése 4000$ akkor az output változót True-ra állítjuk. Majd a legvégén assert-el ellenőrízzük, hogy az output változó True lett vagy sem. Ha True lett akkor tudjuk, hogy Gretchen Watford alapfizetése 4000$, ellenkező esetben nem.

A következő függvényben teszteljük, hogy Tomas Andre nem leader. Ebben az esetben is létrehozunk egy Tomas Andre employee-t a megfelelő adatokkal, majd a RelationsManager is\_leader() metódusát meghívva a létrehozott employee-ra elmentjük az output-ba. Assert kulcsszóval ellenőrízzük, hogy False lett-e az output. Ha pedig le szeretnénk kérdezni a csapatában szereplő alkalmazottakat, akkor None értéket kapnánk vissza, amit utána ellenőrizünk ugyancsak egy assert-ben.

Az utolsó függvény pedig ebben a script-ben azt ellenőrzi, hogy az adatbázisban szerepel-e Jude Overcash nevezetű alkalmazott. Ez esetben is lekérve az alkamazottak listását egy output változót False-ra állítunk. Végig iterálva az alkamazottak listáján, ha találunk egy Jude Overcash nevű alkalmazottat akkor az output-ot True-ra állítjuk. Majd legvégén, mint az eddigi függvényekben is assert-el ellenőrizzük, hogy az output False maradt vagy nem.

Az “employee\_manager\_test.py” scriptben az EmployeeManager osztály metódusait fogjuk tesztelni. A script elején importoljuk a szükséges modulokat, majd példányosítjuk az EmployeeManager osztályt.

Első tesztelő függvényben arról kell megbizonyosodjunk, hogy egy employee, aki nem leader és a céghez való csatlakozási dátuma 10.10.1998 és a kezdőfizetése 1000$ akkor a calculate\_salary függvény által számított fizetés 3000$. Létrehozunk egy employee-t a megadott adatokkal, majd a példányosított EmployeeManagernek meghívjuk a calculate\_salary metódusát erre az employee-ra és kimentjük egy salary változóba. A függvény végén pedig assert-el ellenőrízzük ,hogy ez a salary egyenlő-e 3000$-al.

A második függvénnyel teszteljük, hogy egy employee, aki leader és 3 főből áll a csapata, csatlakozási dátuma 10.10.2008 és kezdő fizetése 2000$, akkor a mostani fizetése 3600$. Itt is hasonló módon járunk el, mint az előző tesztelésnél. Létrehozzuk az employee-t és a RelationsManager osztályhoz még hozzáadjuk ehhez employee-hoz tartozó csapatot. Ezek után a calculate\_salary függvényhívással lekérjük az employee fizetését. Végül pedig ezt hasonlítjuk össze a kívánt értékkel.

Az utolsó unit tesztel pedig azt szeretnénk tesztelni, hogy amikor számolunk egy fizetést és azt email-on el szertnénk küldeni a megfelelő alkalmazottnak, akkor az email küldő szolgálat a megfelelő adatokat használja(név, fizetés). Létrehozunk egy employee-t, aki nem leader, 05.10.2008 a csatlakozási dátuma és 1500$ a kezdő fizetése. Erre az employee-ra hívjuk meg az EmployeeManager calculate\_salary\_and\_send\_email() metódusát, majd a visszatérített szöveget eltároljuk egy message változoban. Ezt a változót feldaraboljuk a szóközök mentén és ellenőrízzük assert-el, hogy az elvárt adatokat kaptuk a függvénytől.

# Tesztelés

A tesztek futtatására Python nyelvben több módszer is létezik. A két leggyakoribb módszer unittest modul használata vagy a pytest keretrendszer használata.

A unittest modul a Python beépített modulja, amely lehetővé teszi a tesztek írását és futtatását. Ez a modul a TestCase osztályokat használja a tesztek definiálására, amelyek egységes és strukturált módon tesztelik az alkalmazásokat.

A pytest egy népszerű harmadik féltől származó tesztelési keretrendszer a Pythonhoz. A pytest egyszerűbb és rugalmasabb módszert kínál a tesztek írására és futtatására, mint az unittest modul. A pytest automatikusan észleli és futtatja a teszteket a kód alapján, és lehetővé teszi a könnyebb tesztek írását is.

Mi a tesztelés során a pytest-et használtuk, amit könnyen futtathatunk terminal-ban.

Először is elindítjuk a terminalt (command prompt) és a projekt mappába navigálunk. Mivel a pytest egy virtualenv-be dolgozik ezért létre kell hoznunk és aktiválnunk a következő parancsal:

$ python3 -m venv pytest-env

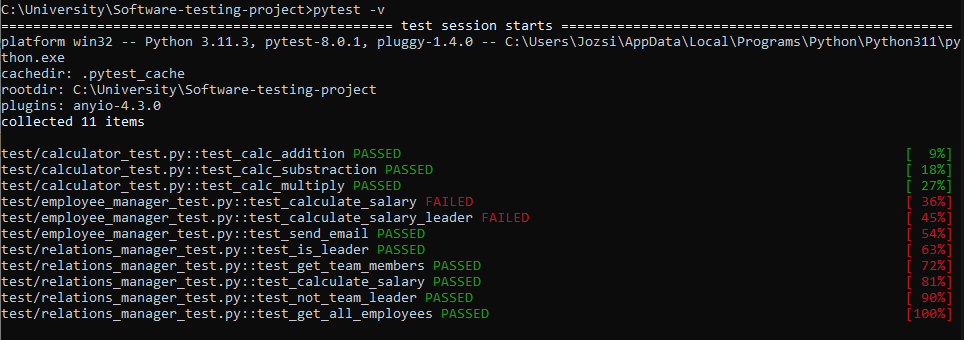
$ source pytest-env/bin/activate

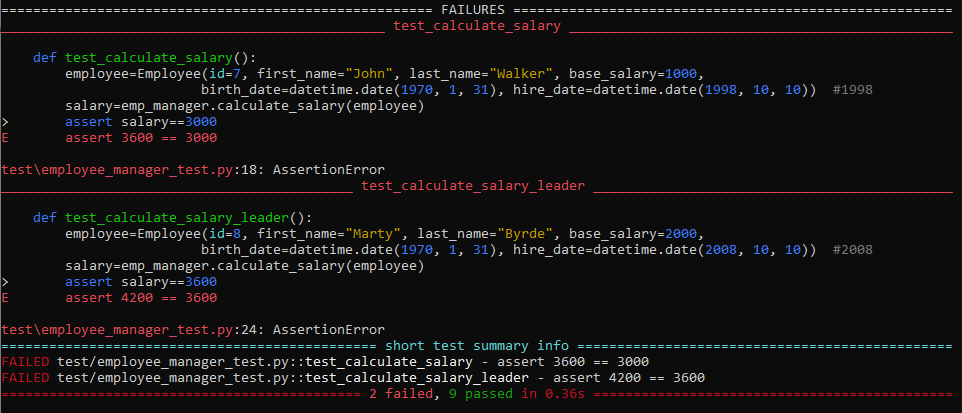
A pytest telepítését pedig a következő és a tesztelés inditását parancsokkal oldjuk meg:

$ pip3 install pytest

$ pytest –v

A –v opció a bőbeszédű kimenetet szolgálja. Ezeket a parancsokat futtatva következő kimenetet kapjuk:





Látható, hogy két unit teszt nem ment Failed lett a teszt során, a többi pedig Passed. A test\_calculate\_salary függvényben a salary változóban kapott érték 3600$, viszont ay elvárt érték 3600$. A fizetést a következőképpen számoljuk ki: 1000$ + 26 X 100$, ahol 1000$ a kezdőfizetés, 26 az évet jelenti, amióta az alkalmazott a cégnél dolgozik, 100$ pedig az éves bónusz. Ezt a matematikai kifejezést elvégezve megkapjuk, hogy a számolt fizetés valóban 3600$. Tehát valószínüleg vagy rosszul volt megadva az alkalmazott csatlakozási dátuma vagy a projektleírás régebb volt tervezve és más volt az elvárt kimenet, de azóta eltelt néhány év és változott az elvárt kimenet.

Hasonló a probléma a test\_calculate\_salary\_leader függvényben is, ahol a következőképpen számoltuk a fizetést az adott leader számára: 2000$ + 16 X 100$ + 3 X 200$, ahol 2000$ a kezdőfizetés, 16 a cégnél dolgozott idő, 100$ az éves bónusz, 3 a csapatában szereplő alkalmazottak száma és 200$ pedig a leader bónuszt jelöli. Ezeket a műveleteket elvégezve megkapjuk, hogy valóban a fizetés 4200$, amit helyesen kaptunk vissza a salary változoban. Tehát itt is az a probléma merül fel, mint az előző függvényben.

1. https://docs.google.com/document/d/1FUUzWrYopCJinb09hKW8EGxBSl7KeYKaA3f8zN8kQDY/edit [↑](#footnote-ref-1)
2. <https://github.com/Jozsyy/Software-testing-project> [↑](#footnote-ref-2)