

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

1.Soru: iyi bir Birim Testinin dört esası size göre nelerdir? Nedenlerini örnekler üzerinde de açıklayarak cevaplamanız beklenmektedir?

1-Regresyonlara (bugs) karşı koruma

regresyonlara karşı koruma: Temelde yazılımın veya kodun sahip olabileceği ve zamanla karşılaşılabileceği buglara karşı korumadır, ve iyi bir birim testinin en önemli esaslarından biridir.

Projelerimizi sürdürmek ve sistemimizde biriken hataların ortasında sıkışıp kalmamak için bu tür bir korumaya ihtiyacımız var. Ne kadar çok kod yazarsak ve ne kadar çok işlev eklersek, buglar o kadar karşımıza çıkabilir.

Bir test puanının ne kadar iyi olduğunu değerlendirmek için aşağıdakilere dikkate etmeliyiz:

- Test sırasında yazdığımız kod miktarı, daha fazla kodu test edersek, daha fazla bugs karşımıza çıkabilir.
- Kodun karmaşıklığı.
- Kodun etki alanı önemi.

Ayrıca, koda ihtiyaç duymayan küçük parçalar üzerinde Unıt testi yapmaktan kaçınmalıyız, çünkü bug olma olasılığı daha düşüktür. Kodumuzda kullandığımız libraries ve framework test etmeyi unutmamamız gerekiyor çünkü yazılımın ve kodumuzun çalışmasını etkiler.

2- Yeniden düzenlemeye karşı direnç

Bu, iyi bir birim testinin başka bir esasıdır, bir test biriminin, davranışını değiştirmeden, başarısız olmadan yeniden düzenleme ve ilk kodun modifikasyonlarını sürdürebilme işlemidir.

Örnek olarak, herhangi bir yerde sorunsuz bir şekilde çalışan bir kodumuz olduğunu düşünelim, daha iyi görünen bir koda sahip olmak için bazı değişiklikler ve yeniden düzenleme yapmaya karar verdik diyelim, ancak sonunda unit testinde bazı hatalar ortaya çıkarsa da, işlevler mükemmel çalışabilir. Bu soruna veya duruma "**Yanlış pozitif**" denir ve bu, işlevler ne kadar iyi çalışırsa çalışsın testin başarısız olduğunu gösterir, genellikle aynı davranışı korurken kodumuzu yeniden düzenlediğimizde gerçekleşir.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

3- Hızlı geri bildirim

İyi bir Unit testinin önemli bir esasıdır. Geri bildirim döngüsü, kodu kırıdığınız anda testler sizi buglar konusunda uarmaya başlayacak ve hata düzeltmelerinin maliyetini neredeyse ortadan kaldıracak şekilde önemli ölçüde kısaltılabilir. Bununla ilgili bir örnek verecek olursam: çok satırlı bir koda sahip olduğumuzu düşünelim ve geri bildirimin yavaş olduğunu farz edelim, Bu yavaşlığı zaman kaybına ve yanlış yönde ilerlemeye neden olabilir o yüzden hızlı geri bildirim Unit testlerde çok önemli olduğunu diyebiliriz.

4-Sürdürülebilirlik

Sürdürülebilirlikten bahsederken, kodun anlaşılabilirliğini göz önünde bulundurmalıyız, kesinlikle basit ve okunabilir bir test kodu yazmalıyız, böylece gerektiğinde değişiklik yapmak daha kolay olacaktır. Ayrıca test çalıştırmanın zorluğunu da göz önünde bulundurmalıyız, eğer test süreç dışı bağımlılıklarla geçerse, bu bağımlılıkları sürdürmek için zaman harcamanız gerekecektir. Örneğin, veri tabanı sunucusunu yeniden başlatmanız veya bir ağ bağlantısı sorununu düzeltmeniz gerekebilir.

2.Soru: Martin Fowler ismi yazılım testi dünyasında öncü isimlerdendir. «Martin Fowler, Birim Testi (Unit Testing) ve Temiz Kod (Clean Code) » üçlemesini nasıl değerlendirirsiniz?

öncelikle şunu söylemeliyim, clean code ile unit test tamamen birbirine bağlı ve eğer iyi bir unit test yazmak istiyorsak, anlayabileceğimiz ve okunması kolay temiz bir kod yazmalıyız. **Test yazmanın bir sırrı yok...**

Am **test edilebilir bir kod yazmanın sırrı vardır!** Dolayısıyla, tüm bunlara dayanarak, daha derin bir şekilde bakarsak ve odaklanırsak Test Edilebilirliğin ne kadar önemli olduğunu anlamış oluyoruz.

İkinci olarak Eğer temiz kod ve Unit testi hakkında konuşacaksam mutlaka **Test Driven-Development (TDD)** hakkında konuşmam gerekecek.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

TDD'nin üç kuralı vardır:

1. **Başarısız bir test yazmak**: bir kodu yazmadan önce başarısız bir unit testi yazmalıyız.

2. **Make the test pass (Testi geçin)**: Daha fazla başarısız bir test yazmamalıyız, temiz bir test yazmayı denemeliyiz.

3. **Gerektiği gibi yeniden düzenleme**: başarısız olan testi geçmek için çok fazla kod yazmamamız gerekiyor.

Tests Enable the – ilities

Unit testleri, kodumuzu esnek, sürdürülebilir ve yeniden kullanılabilir kılan şeydir, yazdığın kod için bir unit testin varsa ne kadar değişiklik yapmaktan korkmazsın ama aynı zamanda yazdığın kodun unit testi yoksa yaptığın herhangi bir değişiklik bir bug'a neden olabilir dolayısıyla değişiklik yapmaktan korkacaksın, yani kısacası ne kadar iyi bir testin varsa , korkun o kadar az olur.

Clean Test (Temiz test)

Temiz kod gerçekten çok önemli olduğunu söyleyebilirim hatta benim açımdan unit testten daha önemli olduğunu diyebilirim çünkü temiz bir kod olmadan iyi bir unit testi yazamayız. Ve temiz bir kod olmadan, kodun okunması ve anlaşılması daha da zor olacak o yüzden okunabilirlik ve anlaşılabilirlik bir kodun bulunması unit testten daha önemlidir.

Peki okunabilirlik bir kod nasıl yazabiliriz?

Readable code (Okunabilirlik kod): netlik, basitlik ve iyi ifade etmek. Kodumuzu elimizden geldiğince birkaç ifadeyle yazmamız gerekiyor yani çok fazla sayıda kod yazmaktan kaçınmalıyız.

Unit test hakkında ilk başta aslında bahsetmiştim ama özetleyecek olursam şöyle bir şey diyebilirim unit test hakkında:

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

- . Ayrı ayrı sınıfları ayrı ayrı test etmek
- . Tüm hata koşullarını simüle edebilir
- . Geliştiriciler bunları her dosya değişikliğinden sonra çalıştırabilir.
- . Çok hızlı

. Green:

- . Yüksek güven sınıfı OK
- . En düşük uygulama kapsamı, bunlardan birçoğuna ihtiyaç var
- . Sure class OK, Sure olmayan etkileşim class OK

. Red:

- . Failures'i çoğaltmak kolaydır.
- . Failure'i neden olduğunu anlamak için debugger'e ihtiyaç yoktur

Unit Tests En küçük parçaları test ederek uygulama mantığına odaklanıyor, Çok hızlı, I/O yok, Debugger yok.

Şimdi Eğer **Martin fowler** hakkında konuşacaksam ve Unit test ile clean code hakkında nasıl bir açıklama yaptığını konuşacaksam, benim yukarıda unit test ile clean code hakkında söylediklerimi hemen hemen bire bire benzediğini görmüş oluyoruz daha bir şekilde açıklayacak olursam şöyle örnek vererek açıklıyorum:

Martin fowler, clean code (Temiz kod) hakkında şöyle bir ifade söyledi, Yazılımın, özelliklerdeki beklenmedik değişikliklere uyum sağlayabilecek şekilde oluşturulması gerekir. Bunu yapmanın en önemli yollarından biri, programın ne yapması gerektiğini anlamayı kolaylaştıran temiz kod yazmaktır. Bu kod, geliştiricilerin yalnızca değişiklik yapmak için ihtiyaç duydukları sistem parçalarını anlamalarına izin veren modüllere bölünmelidir.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

Martin fowler, Yeniden Düzenleme hakkında şöyle bir ifade söyledi:

Yeniden düzenleme, mevcut bir **kod gövdesini** yeniden yapılandırmak, **dış davranışını** değiştirmeden **iç yapısını** değiştirmek için disiplinli bir tekniktir. Ve ne kadar yeniden düzenleme küçükse, yanlış gitme olasılığının daha düşük olduğunu söyledi.

Ve Unit testin ne kadar önemli olduğundan da bahsetmişti ve her kodumuz için bir unit test yazmamızın gerektiğini söyledi.

Martin fowler, Unit test hakkında şöyle bir ifadede bulundu:

Bir dizi otomatik testler ile birlikte kendi kodumuzda ne kadar değişiklik yaparsak yapalım hiç korkmamıza gerek olmadığını söyledi, sırf bir testimiz var diye, dolayısıyla her koda bir unit test yazmamız bizim için daha avantajlı ve daha güvenli bir şekilde davranabileceğimizi söylüyor, Testimiz varken kodda herhangi bir değişiklik yapmaktan korkmamıza gerek kalmadığını söyledi.

Özet olarak, **Martin fowler** Unit test, Clean code ve yeniden düzenlemeyi kodumuzun daha okunabilirlik ve anlaşılabilirlik bir hale gelmesi için ne kadar önemli olduklarını ifade ettiğini görmüş olduk.

3.Soru: Unit Test Frameworks ve çalıştıkları ortamları araştırdıktan bunlarla ilgili olarak size göre nasıl bir sınıflandırma yaparsınız? Açıklayarak cevaplayın.

Unit testinin bir sürü framewokü bulunuyor ve açıkçası frameworkleri sınıflandırmak o kadar kolay değil çünkü her framework farklı yeteneklere sahip ve farklı bir kodlama tipi de vardır, yani birinin diğerlerinden üstün olduğunu diyemeyiz çünkü **projenin gereksinimlerine** göre bir frameworkü seçmeliyiz.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

O zaman var olan birkaç tane frameworktan bahsedeceğim:

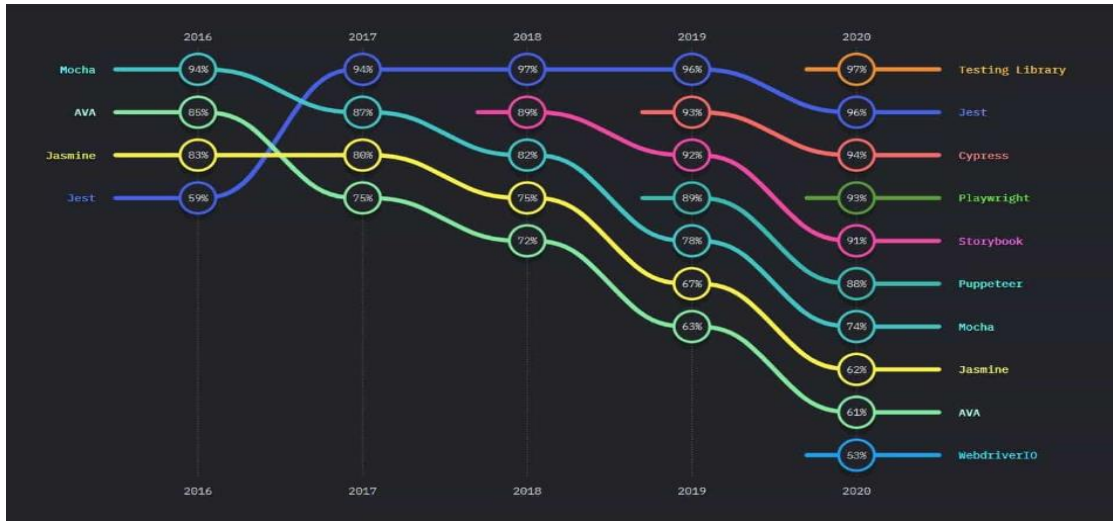
- 1.Jest (Category: Unit Testing)
- 2.NUnit (Category: Unit Testing)
- 3.XUnit (Category: Unit Testing)
- 4.MStest (Category: Unit Testing)
- 5.Jasmine (Category: Unit Testing, End-to-End Testing)
- 6.JUnit (Category: Unit Testing, Regression Testing)
- 7.Cypress (Category: Unit Testing, integration Testing, End-to-End Testing)
- 8.Mocha (Category: Unit Testing, integration Testing, End-to-End Testing)

Şimdi jest, Mocha ve Jasmine arasında bir karşılaştırma yapmak istiyorum :

. React veya Next.JS kullanılarak oluşturulan uygulamalar için ve tek bir entegre sağladığı için daha küçük projeler için **Jest** kullanılması önerilir.

. proje bug ayıklama gerektiriyorsa ve proje oldukça büyükse ve çeşitli harici libraries(kitaplıkların) entegrasyonunu gerektiriyorsa en iyi seçenek **Jasmine** olacaktır.

. Büyük Node.js projeleri için **Mocha** en iyi seçenektir.



Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

Şimdi NUnit, XUnit ve MSTest framework arasında bir karşılaştırma yapmak istiyorsam aşağıdaki şeylere göre karşılaştırma yapabiliriz:

1.Testlerin İzolasyonu:

XUnit framework, NUnit ve MSTest framework'e göre testlerin daha çok iyi izolasyonunu sağlar.

2.Genişletilebilirlik:

Hangi framework diğerinden daha fazla genişletilebilirliğini seçmek istediğimizde, bu seçim projenin ihtiyaçlarına bağlı olabilir. Ancak [Fact] ve [Theory] öznitelikleri XUnit framework tarafından kullanıldığından, bu onu MSTest ve NUnit testinden daha genişletilebilir kılar.

3.Assertion mechanism (İddia mekanizması):

XUnit testi `Assert` kullanıyor, NUnit ve MSTest ise `ExpectedException` kullanıyor, `ExpectedException` sorunu şöyle, Eğer buglerin kodun yanlış bölümünde meydana gelirse `ExpectedException` bu hatayı raporlamayabilir. O yüzden Assert kullanmak daha avantajlı oluyor.

4.Soru: Herhangi bir test framework indirerek işlevinin basit olmadığı bir kod parçasının birim testini 3A yaklaşımı ile yazın, gerekli açıklamaları yaparak ekran çıktılarını paylaşın. Ayrıca problem ifadesinin açık olarak verilmesi gerekmektedir.

İlk olarak yazacağım kodun ne işe yaradığını ne yaptığını kısaca anlatmak istiyorum.

Bu kodun yaptığı şey, çalışanların adını ve soyadını ve alacağı maaş belirleyip çalışan örnekleri oluşturmamıza izin veriyor ve ardından e-postada adı ve soyadı olan çalışanların e-posta.com adresini döndüren bazı yöntemlerimiz var, Ondan sonra sadece adları ve soyadlarının birleşiminden oluşan çalışanların tam adını döndüren bir yöntemimiz var ve ayrıca dizileri uygulayabileceğimiz regular metodu kullanarak bir yöntemimiz var ve maaşlarını mevcut ödeme sürelerine, ışınlarına ayarlayacaktır. Varsayılan olarak bize burada 5% olan miktar. şimdi bu kodun ayarlanma şekli söyle, bir çalışanın adı veya soyadı değişirse, bu otomatik olarak e-postaya ve adına yansıtılacaktır.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

Şimdi kodları göstereceğim ilk dosyam employee.py

employee.py dosyamdaki kodlar:

```
import requests #Mock islemi icin imoprt etmemiz gereken

class Employee:
    raise_amt = 1.05 #sabit degerimiz

    def __init__(self, first, last, pay): #burada isim soyisim maasi
tanimladik
        self.first=first
        self.last=last
        self.pay=pay

    @property
    def email(self): #return mail adres (mail adresimizi dondurecek)
        return '{}.{}@email.com'.format(self.first, self.last)
    @property
    def fullname(self): #return full name (full isim dondurecek)
        return '{} {}'.format(self.first, self.last)

    def apply_raise(self): #regular method where we can apply raise
        self.pay= int(self.pay * self.raise_amt)

    def monthly_schedule(self, month): #Mock islemimiz
        response =
requests.get(f'http://company.com/{self.last}/{month}') #burada bir
tane URL tanimladim formati de verdim
        if response.ok: #Eger dogru sonuc dondururse texti verecek
            return response.text
        else: #degilse bad response degerini dondurecek
            return 'Bad response!'
```

Şimdi diğer test dosyamın kodlarını göstereceğim

test.employee.py dosyamdaki kodlar:

```
import unittest
from unittest.mock import patch
from employee import Employee #importing our employee class from the
employee module

class TestEmployee(unittest.TestCase): #here we are creating our test
case that inherits from unit test test case

    @classmethod
    def setUpClass(cls): #this will run before any test just a once
        print('setUpClass')

    @classmethod
    def tearDownClass(cls):#this will run after all of tests just a
once
        print('tearDownClass')
```


Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

```
def setUp(self): #Here by using setUp we are creating 2 employees
    print('SetUp')
    self.emp_1 = Employee('Kerim', 'Tawil', 40000)
#self.emp_1: thats mean i will set them as instance attributes
    self.emp_2 = Employee('Abdalkarim', 'Altawil', 30000)

def tearDown(self):
    print('tearDown\n')
#here we have three different tests
    def test_email(self): #Not:our test have to start with a word
test,Otherwise compiler will run 0 test.
        print('test_email')
        self.assertEqual(self.emp_1.email, 'Kerim.Tawil@email.com')
#the second argument its what we expected
        self.assertEqual(self.emp_2.email,
'Abdalkarim.Altawil@email.com')

        self.emp_1.first='Amro' #here Im changing our employees names
        self.emp_2.first='Ali'

        self.assertEqual(self.emp_1.email, 'Amro.Tawil@email.com')
#here Im checking the emails againg because its should change
        self.assertEqual(self.emp_2.email, 'Ali.Altawil@email.com')

    def test_fullname(self):
        print('test fullname')
        self.assertEqual(self.emp_1.fullname, 'Kerim Tawil')
        self.assertEqual(self.emp_2.fullname, 'Abdalkarim Altawil')

        self.emp_1.first = 'Amro'
        self.emp_2.first = 'Ali'

        self.assertEqual(self.emp_1.fullname, 'Amro Tawil')
        self.assertEqual(self.emp_2.fullname, 'Ali Altawil')

    def test_apply_raise(self):
        print('test_apply_raise')
        self.emp_1.apply_raise() #here im applying raise and by
default that's 5%
        self.emp_2.apply_raise()

        self.assertEqual(self.emp_1.pay, 42000)
        self.assertEqual(self.emp_2.pay, 31500)

    def test_monthly_schedule(self):
        with patch('employee.requests.get') as mocked_get: #within
patch we pass what we want to mock
            mocked_get.return_value.ok= True # burada dogru bir deger
dondururse
            mocked_get.return_value.text = 'Success' #Success textini
verecek

            schedule = self.emp_1.monthly_schedule('May') #buraya May
ayimizi verdik may disinda bir deger girilirse bug olur

mocked_get.assert_called_with('http://company.com/Tawil/May') #Bizim
```

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

```
URL formatini bu sekile tanımladım
        self.assertEqual(schedule, 'Success')

        mocked_get.return_value.ok = False #burayı da yanlış bir
sey girilirse

        schedule = self.emp_2.monthly_schedule('June')

mocked_get.assert_called_with('http://company.com/Altawil/June')
        self.assertEqual(schedule, 'Bad response!')

if __name__ == '__main__': #allow us to run our test from directly
within our editor
    unittest.main()
```

2.dosyam ile ilgili genel açıklamalar:

Tüm testlerimde tekrar tekrar 2 çalışan oluşturuyorum, bunun yerine tüm test fonksiyonlarımda çalışanları oluşturan setUp yöntemlerini kullandım.

setUp yöntemi, kodunu her testten önce çalıştırır

tearDown yöntemi, her bir testten sonra kodunu çalıştırır

Herhangi bir testten önce bir şey çalıştırmak istiyorsak

@classmethod kullanıyoruz, ve bu kodu herhangi bir şeyden önce bitirmek istiyorsak **TearDown** kullanıyoruz.

Mocking, testimizde gerçekten yardımcı olan bir şeydir, örneğin, bir web sitesine giden ve bazı bilgileri çeken bir işlevimiz varsa, şimdi bu web sitesi kapalıysa, işlevlerimiz başarısız olur ve bu da testimizin başarısız olmasına neden olur ve biz bunu istemiyoruz.istediğimiz şey testimiz sadece Eğer kodda bir hata bug varsa başarısız olsun . bunun için Mocking kullanmamız çok uygun olacaktır.

Şimdi burada kodun çıktısını göstereceğim hem bug olduğu zamanda nasıl bir çıktı vereceğini göstereceğim hemde hatasız (bug olmadan) bir şekilde nasıl bir çıktı vereceğini göstereceğim.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

İlk olarak faile verdiği zamanda:

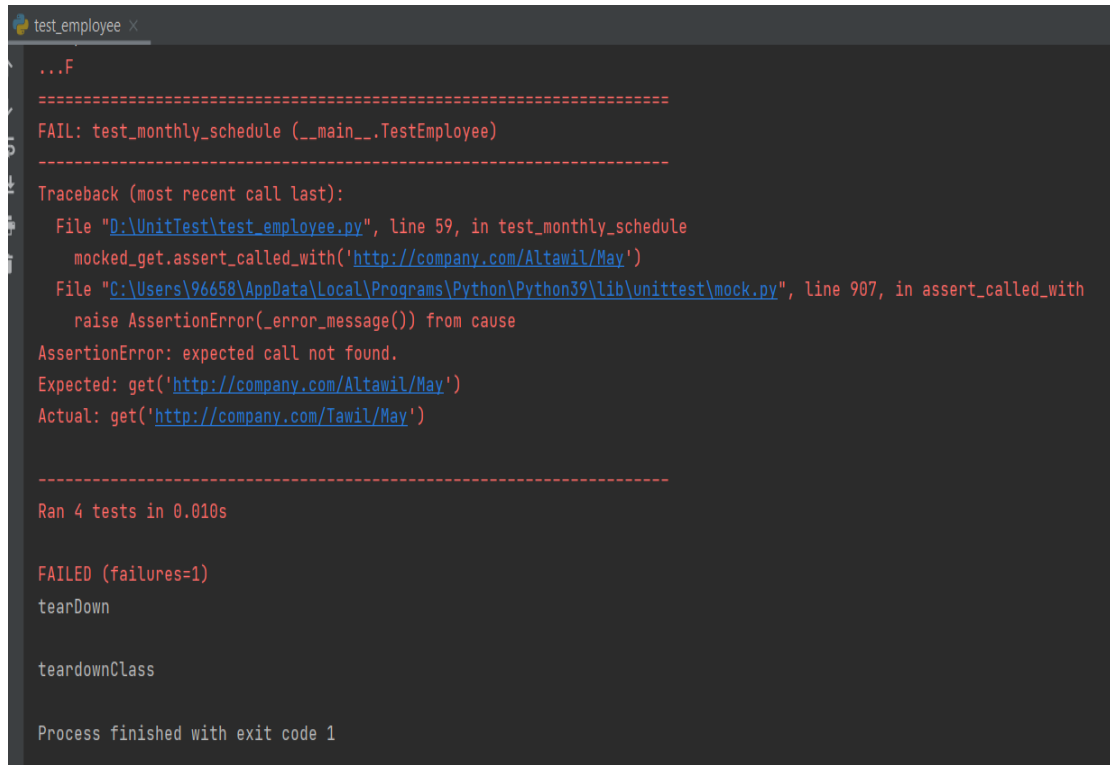
Mock kısmındaki URLdeki last name'i yanlış yazacağım bakalım nasıl bir çıktı verecek bize .

Normalde doğru kodumuz bu şekilde olmalı:

```
mocked_get.assert_called_with('http://company.com/Tawil/May')
```

Ben yukarıdaki Tawil, Altawil olarak yazacağım yani şöyle:

```
mocked_get.assert_called_with('http://company.com/Altawil/May')
```



```
test_employee x
...F
=====
FAIL: test_monthly_schedule (__main__.TestEmployee)
-----
Traceback (most recent call last):
  File "D:\UnitTest\test_employee.py", line 59, in test_monthly_schedule
    mocked_get.assert_called_with('http://company.com/Altawil/May')
  File "C:\Users\96658\AppData\Local\Programs\Python\Python39\lib\unittest\mock.py", line 907, in assert_called_with
    raise AssertionError(_error_message()) from cause
AssertionError: expected call not found.
Expected: get('http://company.com/Altawil/May')
Actual: get('http://company.com/Tawil/May')
-----
Ran 4 tests in 0.010s

FAILED (failures=1)
tearDown
tearDownClass

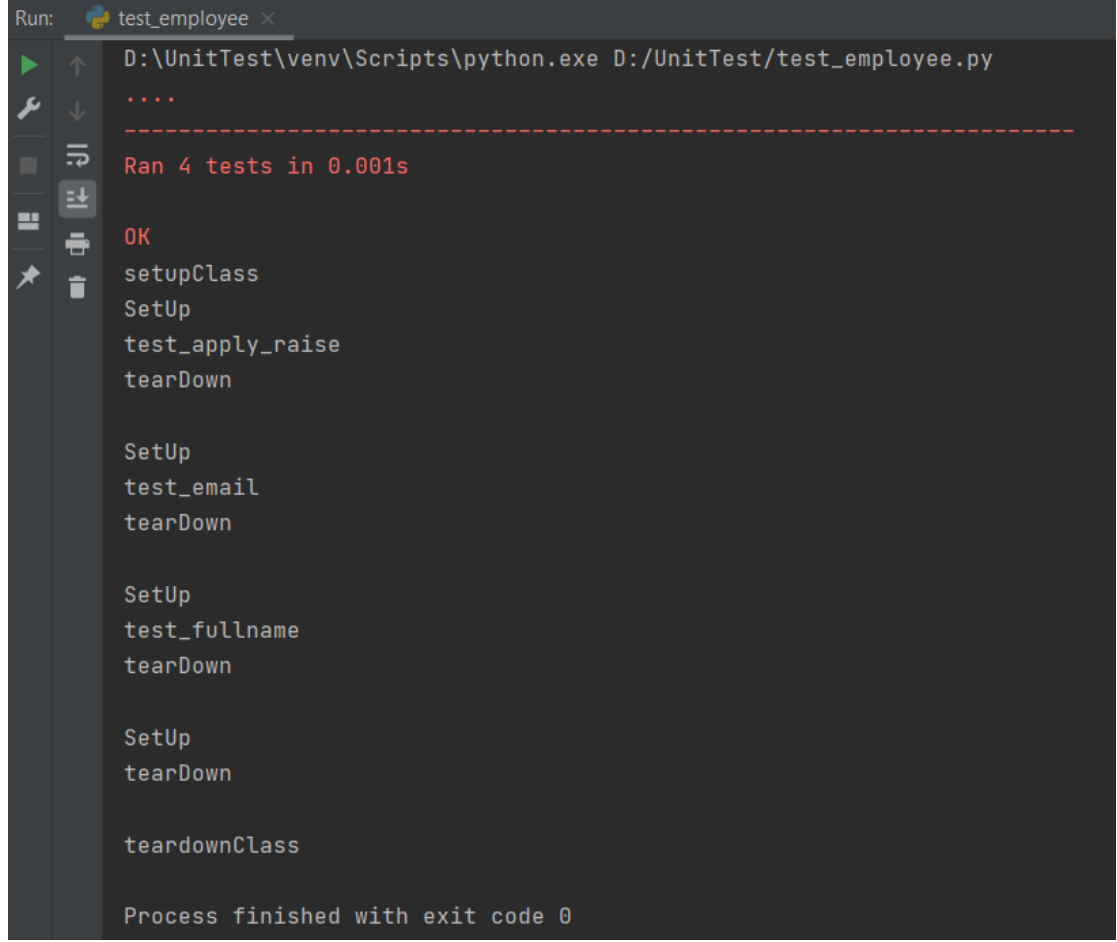
Process finished with exit code 1
```

Burada gördüğümüz gibi Expected ve Actual ne olduğunu görebiliyoruz. Şimdi Bu Failed'i düzelterek Nasıl bir çıktı vereceğine bakalım.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084



```
Run: test_employee x
D:\UnitTest\venv\Scripts\python.exe D:/UnitTest/test_employee.py
....
-----
Ran 4 tests in 0.001s
OK
setUpClass
setUp
test_apply_raise
tearDown

setUp
test_email
tearDown

setUp
test_fullname
tearDown

setUp
tearDown

tearDownClass

Process finished with exit code 0
```

Gördüğümüz gibi testimiz başarılı bir şekilde çalıştı.

Kaynakça bir sonraki sayfada paylaşacağım

Teşekkürler Sayın Dr. Öğr. Üyesi: Zeynep ALTAN

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

Kaynakça:

<https://notesbylex.com/4-pillars-of-good-unit-tests.html>

[https://sd.blackball.lv/library/unit_testing_\(2020\).pdf](https://sd.blackball.lv/library/unit_testing_(2020).pdf)

<https://brightsec.com/blog/unit-testing-frameworks/>

<https://brightsec.com/blog/unit-testing-best-practices/#deterministic-tests>

<https://medium.com/ltunes/erken-uyar%C4%B1-sistemi-unit-test-9e2ab05cc760>

<https://www.lambdatest.com/blog/jest-vs-mocha-vs-jasmine/>

<https://insights.stackoverflow.com/survey/2021#overview>

https://knapsackpro.com/testing_frameworks/difference_between/nunit/vs/cypress-io

https://knapsackpro.com/testing_frameworks/difference_between/nunit/vs/junit

<https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>

<https://www.lambdatest.com/blog/cypress-vs-selenium-comparison/>

<https://martinfowler.com/>

<https://martinfowler.com/aboutMe.html>

<https://refactoring.com/>

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084