Uygulamalı Laboratuvar: Test Doğrulamalarını Yazma

Gerekli tahmini süre: 30 dakika

Test Doğrulamalarını Yazma konulu uygulamalı laboratuvara hoş geldiniz. Doğrulamalar, test sonuçlarımızın geçti mi yoksa başarısız mı olduğunu belirlemek için kontrol olarak kullanılır: Doğru (True) geçildiği, Yanlış (False) ise başarısız olduğu anlamına gelir. Doğrulamaların ilginç yanı, eğer Yanlış olarak değerlendirilirse bir istisna (exception) oluşturmasıdır, bu da testi başarısız olarak işaretler.

Bu laboratuvarın sonunda, test durumları için nasıl test doğrulamaları yazacağınızı öğreneceksiniz.

Öğrenme Hedefleri

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- Nose ile test durumlarını çalıştırmak
- Başarısız olan test durumlarını tanımlamak
- Doğrulamalar kullanarak birim testleri yazmak

Laboratuvar Ortamını Kurma

Laboratuvara başlamadan önce biraz hazırlık yapmanız gerekiyor.

Terminali Açın

Editördeki menüyü kullanarak bir terminal penceresi açın: Terminal > Yeni Terminal.

Terminalde, eğer zaten `/home/projects` klasöründe değilseniz, şimdi proje klasörünüze geçin.

```
cd /home/project
```

Kodu Repo'sunu Klonlayın

Şimdi test etmeniz gereken kodu alın. Bunu yapmak için, git deposunu klonlamak için `git clone` komutunu kullanın:

```
git clone https://github.com/ibm-developer-skills-network/duwjx-tdd_bdd_PracticeCode.git
```

Your output should look similar to the image below:

Laboratuvar Klasörüne Geçin

Depoyu klonladıktan sonra, laboratuvar dizinine geçin.

```
cd duwjx-tdd_bdd_PracticeCode/labs/02_writing_test_assertions/
```

Bu dizinin içeriğini listeleyerek bu laboratuvar için belgeleri görebilirsiniz.

```
ls -l
```

Dizin aşağıdaki listeye benzer görünmelidir:

İzleyeceğiniz adımlarda çalıştıracağınız birkaç alıştırma dosyanız olmalıdır.

Son hazırlık adımı, laboratuvar için gerekli Python paketlerini yüklemek için `pip` kullanmaktır:

```
python3.8 -m pip install -r requirements.txt
```

Laboratuvara başlamak için hazırsınız.

İsteğe Bağlı

Terminalde çalışmak zorlaşırsa çünkü komut istemi çok uzunsa, istemi kısaltmak için aşağıdaki komutu kullanabilirsiniz:

```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Test Dosyaları ile Çalışmak

Laboratuvar klasöründen `stack.py` ve `test_stack.py` adlı iki dosyayı kullanacaksınız.

- `stack.py`, test etmek istediğiniz kodu içeriyor.
- `test_stack.py`, `push()`, `pop()`, `peek()` ve `is empty()` yöntemleri için test durumlarıyla birlikte test iskeletini içeriyor. Şu anda her test bir istisna oluşturuyor.

Bu laboratuvarı tamamlamak için, `test_stack.py` dosyasındaki test durumlarını düzenleyecek ve uygun uygulamayı sağlayacaksınız.

Düzenleyicide `test_stack.py` dosyasını açın. Düzenleyiciyi açmak için aşağıdaki düğmeye tıklayın.

Open **test_stack.py** in IDE

Gözden Geçirme: Yığın

Test vakalarını yazmadan önce, `Stack` sınıfının uyguladığı yöntemleri gözden geçirin:

```
"""Implements a Stack data structure"""
class Stack:
    def push(self, data: Any) -> None:
    def pop(self) -> Any:
    def peek(self) -> Any:
    def is_empty(self) -> bool:
```

Fonksiyon	Açıklama
`push()`	Bir öğeyi yığının **üstüne** ekler
`pop()`	Yığının **üstündeki** öğeyi kaldırır ve o öğeyi döndürür
`peek()`	Yığının **üstündeki** öğenin değerini döndürür, ancak öğeyi yığında bırakır
`is_empty()`	Yığın boşsa `True` döndürür, aksi takdirde `False` döndürür

> `is_empty()` haricindeki tüm yöntemlerin yığının **üstündeki** öğe üzerinde çalıştığını unutmayın. Bu nedenle, test senaryolarınızın kapsamlı olması için yığında iki veya daha fazla öğe bulunmalıdır, böylece **üstteki** öğeyi manipüle ettiğinizden emin olursunuz, **alttaki** öğeyi değil.

Nosetests Çalıştırma

Herhangi bir kod yazmadan önce, test vakalarının geçip geçmediğini kontrol etmelisiniz. Aksi takdirde, eğer başarısız olurlarsa, kodu *siz* kırdınız mı yoksa başkası ya da başka bir şey mi kırdı, bilemezsiniz.

Bu test vakalarının ne döndüğünü görmek için `nosetests` komutunu çalıştırın:

```
nosetests
```

Terminalde yukarı kaydırarak tüm test vakalarının kırmızı olarak listelendiğini görebilirsiniz, bu da onların başarısız olduğu anlamına geliyor. Başarısız oldular çünkü hepsi basitçe “uygulanmadı” istisnası fırlatıyor. Bu, yeni test vakalarını eklemeniz gereken yeri hatırlatmak içindir.

Çıktı, her test vakasının beklendiği gibi bir istisna fırlattığını gösteriyor. Şimdi bu test vakalarının geçmesi için bazı kodlar yazmanız gerekiyor.

İlk hatayı bulma

Nosetests'in `--stop` seçeneği, ilk başarısız test durumunda testleri durdurur. Bu laboratuvar boyunca, bir sonraki başarısız test durumunu bulmak için bu seçeneği kullanacaksınız.

Hangi test üzerinde çalışmanız gerektiğini öğrenmek için `nosetests --stop` komutunu çalıştırın:

```
nosetests --stop
```

Nosetests, ilk başarısız test durumunda durur, bu da **Yığın boş mu kontrol et**.

> **"Yığın boş mu kontrol et"** test durumu `test_stack.py` içindeki ilk test durumu değildir. Nose, testleri sahte rastgele bir sırada çalıştırır. Bu, test durumlarının yürütme sırasını etkilememesini veya buna bağlı olmamasını sağlamak içindir.

Bu test durumunu yazın.

Adım 1: is_empty() yöntemini test etme

İlk başarısız test durumunuz **Yığın boş mu?** olup, bu `test_is_empty()` yönteminde uygulanmıştır. Bu yöntem, yığın boşsa `True`, boş değilse `False` döndürür. Her iki sonucu test etmek için her iki senaryonun da test edilmesi gerekmektedir.

Göreviniz

`is_empty()` yönteminin her iki olası sonucunu test eden bir test durumu yazın.

`setUp()` yöntemini görüntülediğinizde, yeni bir `Stack` oluşturduğunu göreceksiniz. Başlangıçtaki yığın boş olmalıdır, bu nedenle ilk doğrulamanız bunu kontrol edebilir. Ancak yığın boşken `is_empty()`'nin `True` döndürdüğünü doğrulamak yeterli değildir; ayrıca yığın *boş değilken* yöntemin `False` döndürdüğünü de doğrulamanız gerekir. Tüm olası sonuçları test etmek için her iki doğrulamaya da ihtiyacınız var.

▼ İpucu için buraya tıklayın.
İşte kodunuzun bir taslağı: 1. Yığının başlangıçta boş olduğunu doğrulayın. 1. Yığına bir değer ekleyin. 1. Yığının artık boş olmadığını doğrulayın.

Çözüm

▼ Cevap için buraya tıklayın.
Aşağıdaki kodu kopyalayın ve `test_is_empty()` fonksiyonunu değiştirmek için kullanın. Doğru girintilemeyi yaptığınızdan emin olun:

```
    def test_is_empty(self):
        """Test if the stack is empty"""
        self.assertTrue(self.stack.is_empty())
        self.stack.push(5)
        self.assertFalse(self.stack.is_empty())
```

Çözümünüzü Test Edin

Tekrar `nosetests --stop` komutunu çalıştırın ve `test_is_empty()`'in geçip geçmediğini not edin:

```
nosetests --stop
```

Şunu bulmalısınız:

- **Yığın boş mu test et** geçti ve
- **Yığının üstüne bakma test** bir sonraki başarısız test durumunuz.

Yığının üstüne bakma testine geçiyorsunuz.

Adım 2: peek() metodunu test etme

Uygulayacağınız bir sonraki test durumu `test_peek()` metodudur.

`peek()` metodu, yığının tepe noktasındaki değeri döndürür. Yıkıcı değildir ve `pop()` gibi değeri kaldırmaz. Eğer yığının durumunu değiştirmeden `peek()` metodunu ardışık olarak birkaç kez çağırırsanız, her seferinde aynı değeri döndürmesi gerekir.

Göreviniz

`peek()` için bir test durumu yazın.

`peek()` metodunu test etmek için yığında bir şey olmalıdır. Öncelikle yığına bir değer `push()` etmeniz gerekir. Ancak bu, yararlı bir test durumu için yeterli mi? `peek()` yığının tepe noktasındaki değeri döndürmelidir. Ama eğer yığında yalnızca bir değer varsa, tepe ve taban aynı değerdir. `peek()`'in yığının tabanındaki değeri döndürmediğinden nasıl emin olabilirsiniz? Taban değeri ile tepe değerini farklı kılmak için ikinci bir değer `push()` etmeniz gerekir. Ardından `peek()`'in yığına en son eklediğiniz değeri döndürdüğünü doğrulamanız gerekir.

▼ İpucu için buraya tıklayın.
İşte kodun bir taslağı: 1. Yığına iki değer ekleyin. 1. `peek()`'in en son eklenen değeri döndürdüğünü doğrulayın.

Çözüm

▼ Cevap için buraya tıklayın.
Aşağıdaki kodu kopyalayın ve `test_peek()` fonksiyonunu değiştirmek için kullanın. Doğru şekilde girintilediğinizden emin olun:

```
        def test_peek(self):
            """Test peeking at the top the stack"""
            self.stack.push(3)
            self.stack.push(5)
            self.assertEqual(self.stack.peek(), 5)
```

Çözümünüzü Test Edin

`nosetests --stop` komutunu tekrar çalıştıralım ve `test_peek()`'in geçip geçmediğine bakalım:

```
    nosetests --stop
```

Şunu bulmalısınız:

- **Yığın üzerindeki en üstteki öğeyi kontrol etme testi** geçti ve
- **Yığından bir öğe çıkarma testi** bir sonraki başarısız test durumunuzdur.

Sıradaki test **Yığından bir öğe çıkarma testi**.

Adım 3: pop() metodunu test etme

Uygulayacağınız bir sonraki test durumu `test_pop()` metodudur.

`pop()` metodu, yığının üstündeki değeri "çıkarır" veya "kaldırır" ve ardından bu değeri döndürür. Bu metodu test etmek için yığına sadece bir değer eklemek yeterli değildir; üst ve alt aynı olacaktır. Bu nedenle, `peek()` metodunu test ettiğinizde olduğu gibi, yığına iki değer eklemeniz gerekir. Bu şekilde, yığında iki değer olur ve yığının üstü ile altı aynı değere işaret etmez.

Göreviniz

`pop()` için bir test durumu yazın.

`pop()`'u test etmek için, kaldırılacak bir şeye ihtiyacınız var. Öncelikle yığına bir değer `push()` etmeniz gerekiyor. Ama bu, yararlı bir test durumu için yeterli mi? `pop()` yığının üstündeki değeri döndürmeli. Ancak, yığında sadece bir değer varsa, üst ve alt aynı değerdir. `pop()`'un yığının altındaki değeri döndürmediğinden nasıl emin olabilirsiniz?

Bu, `peek()`'ü test ederken karşılaştığınız aynı sorundur ve aynı çözümü vardır. Alt değeri ve üst değerini farklı hale getirmek için ikinci bir değer eklemeniz gerekiyor. Ardından, `pop()`'un yığına eklediğiniz son değeri döndürdüğünü doğrulamalısınız.

Ayrıca, öğenin kaldırıldığını test etmeniz gerekir. Unutmayın ki `peek()` yığının üstündeki değeri döndürecektir. Bu nedenle, `pop()`'tan sonra `peek()`'in çağrılmasının yığındaki önceki değeri döndüreceğini doğrulayabilirsiniz. Bu değer artık üstte olmalıdır.

▼ İpucu için buraya tıklayın.
İşte kodun bir taslağı: 1. Yığına iki değer ekleyin. 1. Bir değeri çıkarın ve bunun son eklediğiniz değer olduğunu doğrulayın. 1. `peek()`'in artık eklediğiniz önceki değeri döndürdüğünü doğrulayın. 1. (İsteğe bağlı) `pop()`'un yığındaki son değeri kaldırıp kaldırmadığını daha fazla test etmek için, `pop()`'u tekrar çağırabilir ve ardından yığının artık boş olduğunu doğrulayabilirsiniz.

Çözüm

▼ Cevap için buraya tıklayın.
Aşağıdaki kodu kopyalayın ve `test_pop()` fonksiyonunu değiştirmek için kullanın. Doğru şekilde girintilediğinizden emin olun:

```
        def test_pop(self):
            """Test popping an item of off the stack"""
            self.stack.push(3)
            self.stack.push(5)
            self.assertEqual(self.stack.pop(), 5)
            self.assertEqual(self.stack.peek(), 3)
            self.stack.pop()
            self.assertTrue(self.stack.is_empty())
```

Çözümünüzü Test Edin

`nosetests --stop` komutunu tekrar çalıştıralım ve `test_peek()`'in geçip geçmediğine bakalım:

```
    nosetests --stop
```

Şunu bulmalısınız

- **Yığın üzerinden bir öğeyi çıkarma testinin** başarılı olduğunu ve
- **Yığına bir öğe ekleme testinin** bir sonraki başarısız test durumu olduğunu.

Yığına bir öğe ekleme testine geçiyorsunuz.

Adım 4: push() yöntemini test etme

Uygulayacağınız son test durumu `test_push()` yöntemidir.

`push()` yöntemi, bir değeri yığının en üstüne ekler ve yığın, üzerine eklenen her yeni değerle büyür. Bu yöntemi test ettiğinizde, değerin yığının en üstüne eklendiğinden, en altına değil, emin olmalısınız.

Göreviniz

`push()` için bir test durumu yazın.

`push()` fonksiyonunu çağırmanız ve ardından doğru davrandığını doğrulamanız gerekecek. Sonra, yığına `peek()` yaptığınızda, `peek()`'in yeni eklediğiniz değeri döndürdüğünü doğrulayabilirsiniz. Alternatif olarak, yığından `pop()` yaptığınızda, `pop()`'un yeni eklediğiniz değeri döndürdüğünü doğrulayabilirsiniz. Bu doğrulamalardan biri yeterlidir.

▼ İpucu için buraya tıklayın.
İşte kodun bir taslağı: 1. Yığına bir değer ekleyin. 1. Yığına bakmanın o değeri döndürdüğünü doğrulayın.

Çözüm

▶ Cevap için buraya tıklayın.

Çözümünüzü Test Edin

Son bir kez `nosetests --stop` komutunu çalıştırın ve `test_push()` testinin geçip geçmediğine bakın:

```
nosetests --stop
```

Tüm testleriniz geçiyor. Harika iş!

Sonuç

Bu İddia Laboratuvarını Tamladığınız İçin Tebrikler

Umarım artık test iddiaları yazmayı biliyorsunuzdur. İddiaların, testlerin geçip geçmediğini belirlemek için yapılan kontroller olduğunu biliyorsunuz. Testlerinizi daha okunabilir hale getirmek için `PyUnit涼ase` sınıfının sağladığı iddiaları kullandınız.

Artık kendi projeleriniz için iddialarla test senaryoları yazmalısınız, böylece kodunuzun beklediğiniz gibi çalıştığından emin olabilirsiniz.

Author(s)

John J. Rofrano

Contributor(s)

Srishti Srivastava