Unit Tests

Hozirgacha siz `print` yordamida o'z kodingizni tekshirib ko'rgansiz. Dasturlash sohasida ozingiz yozgan code ni tekshirish uchun code yozish keng tarqalgan amaliyot dur.

Konsol oynasida `code calculator.py` kodini kiriting. E'tibor bering, siz ushbu faylni oldingi ma'ruzada oldindan kodlagan bo'lishingiz mumkin, kodingiz quyidagicha ko'rinishiga ishonch hosil qiling:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

E'tibor bering, siz yuqoridagi kodni o'zingiz aniq bir raqamlar yordamida sinab ko'rishingiz mumkin, masalan, 2. Biroq, nima uchun yuqoridagi kod to'g'ri ishlashini ta'minlaydigan test yaratishni xohlashingiz mumkinligini ko'rib chiqing?.

Barcha dasturchilar nomlagandaqa test_calculator.py ni yarating, yangi test dasturini yaratamiz va kodingizni quyidagicha o'zgartiramiz:

```
from calculator import square


def main():
    test_square()


def test_square():
    if square(2) != 4:
        print("2 squared was not 4")
    if square(3) != 9:
        print("3 squared was not 9")


if __name__ == "__main__":
    main()
```

E'tibor bering, biz square funktsiyasini square.py dan kodning birinchi qatoriga import qilmoqdamiz. Shartli ravishda, test_square deb nomlangan funktsiyani yaratmoqdamiz. Ushbu funktsiya ichida biz sinov uchun ba'zi shartlarni belgilaymiz.

Konsol da `python test_calculator.py` yozing. Hech narsa chiqmayotganini sezasiz. Hamma narsa yaxshi ketayotgan bo'lishi mumkin! Shu bilan bir qatorda, bizning test funksiyamiz xatoga olib kelishi mumkin bo'lgan muammo larni aniqlamagan bo'lishi mumkin.

Hozir bizning kodimiz ikkita shartni sinovdan o'tkazmoqda. Agar biz yana ko'p shartlarni sinab ko'rmoqchi bo'lsak, test kodimiz osongina kattalashi ketishi mumkin. Sinov kodimizni kengaytirmasdan test imkoniyatlarimizni qanday kengaytirishimiz mumkin?

Assert

Python-ning **assert** buyrug'i bizga kompilyatorga biror narsa, ba'zi bir tasdiqlar haqiqat ekanligini aytishga imkon beradi. Buni sinov kodimizga quyidagicha qo'llashimiz mumkin:

```
from calculator import square


def main():
    test_square()


def test_square():
    assert square(2) == 4
    assert square(3) == 9


if __name__ == "__main__":
    main()
```

E'tibor bering, biz square(2) va square(3) qanday teng bo'lishi kerakligini aniq tasdiqlaymiz. Bizning kodimiz to'rtta test qatoridan ikkitaga qisqartirildi.

Biz kalkulyator kodini quyidagi tarzda o'zgartirish orqali ataylab buzishimiz mumkin:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
    main()
```

E'tibor bering, biz kvadrat funksiyada * operatorini + ga o'zgartirdik.

Endi konsol da python test_square.py ishga tushirilganda, kompilyator tomonidan AssertionError ko'tarilganini sezasiz. Aslida, bu bizning shartlarimizdan biri bajarilmaganligini aytadigan kompilyator.

Biz hozir duch kelayotgan muammolardan biri shundaki, agar biz foydalanuvchilarga ko'proq tavsiflovchi xato chiqishini taqdim qilmoqchi bo'lsak,

bizning kodimiz yanada og'irlashishi mumkin. To'g'ri, biz quyidagi tarzda kodlashimiz mumkin:

```
from calculator import square


def main():
    test_square()


def test_square():
    try:
        assert square(2) == 4
    except AssertionError:
        print("2 squared is not 4")
    try:
        assert square(3) == 9
    except AssertionError:
        print("3 squared is not 9")
    try:
        assert square(-2) == 4
    except AssertionError:
        print("-2 squared is not 4")
    try:
        assert square(-3) == 9
    except AssertionError:
        print("-3 squared is not 9")
    try:
        assert square(0) == 0
    except AssertionError:
        print("0 squared is not 0")


if __name__ == "__main__":
    main()
```

E'tibor bering, ushbu kodni ishga tushirish bir nechta xatolarga olib keladi. Biroq, u yuqoridagi barcha xatolarni keltirib chiqarmaydi. Bu kodlashda xatolar mavjud bo'lgan vaziyatlarni qo'lga kiritish uchun bir nechta holatlarni sinab ko'rishga arziydigan yaxshi misoldir.

Yuqoridagi kod katta muammoni ko'rsatadi: yuqoridagi kabi o'nlab kod satrlarisiz kodingizni sinab ko'rishni qanday osonlashtiramiz?

Pytest

pytest uchinchi tomon kutubxonasi bo'lib, u sizning dasturingizni birlikda sinab ko'rish imkonini beradi. Ya'ni, o'z funksiyalaringizni dasturingiz doirasida sinab ko'rishingiz mumkin.

Pytest-dan foydalanish uchun konsol ga `pip install pytest` yozing. Pytestni o'z dasturimizga qo'llashdan oldin test_calculator funksiyangizni quyidagicha o'zgartiring:

```
from calculator import square


def test_assert():
    assert square(2) == 4
    assert square(3) == 9
    assert square(-2) == 4
    assert square(-3) == 9
    assert square(0) == 0
```

Yuqoridagi kod biz sinab ko'rmoqchi bo'lgan barcha shartlarni qanday tasdiqlaganiga e'tibor bering. pytest bizga dasturimizni to'g'ridan-to'g'ri u orqali ishga tushirishga imkon beradi, shuning uchun test shartlarimiz natijalarini osonroq ko'rishimiz mumkin.

Terminal da pytest test_calculator.py yozing. Natija taqdim etilishini darhol sezasiz. Natijaning yuqori qismidagi qizil F ga e'tibor bering, bu sizning kodingizdagi biror narsa muvaffaqiyatsiz ekanligini ko'rsatadi. Bundan tashqari, qizil E calculator.py dasturingizdagi xatolar haqida ba'zi maslahatlar berishiga e'tibor bering. Natija asosida siz 3 * 3 9 o'rniga 6 ni chiqargan stsenariyni tasavvur qilishingiz mumkin. Ushbu test natijalariga ko'ra biz calculator.py kodimizni quyidagicha tuzatishimiz mumkin:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

E'tibor bering, biz kvadrat funksiyadagi + operatorini * ga o'zgartirdik.

Pytest test_calculator.py qayta ishga tushirilsa, qanday qilib xatolik yuzaga kelmasligiga e'tibor bering. Tabriklaymiz!

Ayni paytda, pytest birinchi muvaffaqiyatsiz sinovdan keyin ishlashni to'xtatishi ideal emas. Yana, keling, calculator.py kodimizni buzilgan holatiga qaytaramiz:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
    main()
```

E'tibor bering, biz * operatorini kvadrat funksiyadagi + ga o'zgartirdik va uni buzilgan holatga qaytardik.

Sinov kodimizni yaxshilash uchun test_calculator.py kodini turli test guruhlariga bo'lish uchun o'zgartiramiz:

```
from calculator import square
```

```
def test_positive():
    assert square(2) == 4
    assert square(3) == 9


def test_negative():
    assert square(-2) == 4
    assert square(-3) == 9


def test_zero():
    assert square(0) == 0
```

E'tibor bering, biz bir xil beshta testni uch xil funktsiyaga ajratdik. Pytest kabi test tizimlari har bir funktsiyani bajaradi, hatto ulardan birida nosozlik bo'lsa ham. Pytest test_calculator.py qayta ishga tushirilsa, yana ko'plab xatolar ko'rsatilayotganini sezasiz. Ko'proq xato chiqishi sizning kodingizdagi muammolarni keltirib chiqarishi mumkin bo'lgan narsalarni ko'proq o'rganishga imkon beradi.

Sinov kodimizni yaxshilagandan so'ng, calculator.py kodingizni to'liq ish tartibiga qaytaring:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

Pytest test_calculator.py qayta ishga tushirilsa, hech qanday xato topilmaganini sezasiz. Xulosa qilib aytadigan bo'lsak, koder sifatida o'zingizga mos keladigan test shartlarini belgilash sizga bog'liq!

Pytestning pytest qollanmasida ko'proq ma'lumot olishingiz mumkin.

Testing Strings

Quyidagi hello.py kodini ko'rib chiqing:

```
def main():
    name = input("What's your name? ")
    hello(name)


def hello(to="world"):
    print("hello,", to)


if __name__ == "__main__":
    main()
```

E'tibor bering, biz hello funktsiyasining natijasini sinab ko'rishimiz mumkin. test_hello.py uchun quyidagi kodni ko'rib chiqing:

```
from hello import hello


def test_hello():
    assert hello("David") == "hello, David"
    assert hello() == "hello, world"
```

Ushbu kodni ko'rib chiqsangiz, testga bunday yondashuv yaxshi ishlaydi deb o'ylaysizmi? Nega bu test yaxshi ishlamasligi mumkin? E'tibor bering, hello.py-dagi hello funksiyasi biror narsani print etadi: Ya'ni, u qiymat qaytarmaydi!

Biz hello.py ichida hello funksiyamizni quyidagicha o'zgartirishimiz mumkin:

```
def main():
    name = input("What's your name? ")
    print(hello(name))


def hello(to="world"):
    return f"hello, {to}"
```

```
if __name__ == "__main__":
    main()
```

E'tibor bering, biz satrni qaytarish uchun hello funksiyamizni o'zgartirdik. Bu shuni anglatadiki, biz endi salom funktsiyasini sinab ko'rish uchun pytest-dan foydalanishimiz mumkin.

Ushbu darsdagi oldingi test ishimizda bo'lgani kabi, biz ham testlarimizni alohida ajratishimiz mumkin:

```
from hello import hello


def test_default():
    assert hello() == "hello, world"


def test_argument():
    assert hello("David") == "hello, David"
```

E'tibor bering, yuqoridagi kod bizning testimizni bir nechta funktsiyalarga ajratadi, shunda ular xatolik yuzaga kelgan taqdirda ham ishlaydi.

Testlarni papkalarga ajratish

Bir nechta testlardan foydalangan holda unit test ni kodi shunchalik keng tarqalganki, siz bitta buyruq bilan butun testlar papkasini ishga tushirishingiz mumkin.

Birinchi test deb nomlangan papkasini yarating. Keyin shu test papkasi ichida hello.py faylini yarating va shu fayl ichida quyidagi kodni kiriting:

```
from hello import hello


def test_default():
    assert hello() == "hello, world"


def test_argument():
```

```
    assert hello("David") == "hello, David"
```

E'tibor bering, biz avvalgidek test yaratmoqdamiz.

pytest maxsus __init__ faylisiz faqat ushbu fayl (yoki butun fayllar to'plami) bilan papka sifatida testlarni bajarishimizga ruxsat bermaydi. Test papkani ichida ushbu faylni yarating yani __init__.py . Avvalgidek testga e'tibor bering, shuningdek initning har ikki tomonidagi ikkita pastki chiziqqa e'tibor bering. Ushbu __init__.py faylini bo'sh qoldirgan taqdirda ham, pytest __init__.py o'z ichiga olgan butun papkada ishga tushirilishi mumkin bo'lgan testlar mavjudligi haqida xabar beriladi.

Endi terminalda `pytest test` ini yozib, kodning butun test papkasini ishga tushirishingiz mumkin.

Kodni sinab ko'rish dasturlash jarayonining tabiiy qismidir. Unit test testlari sizning kodingizning o'ziga xos jihatlarini sinab ko'rish imkonini beradi. Kodingizni sinab ko'radigan o'z dasturlaringizni yaratishingiz mumkin. Shu bilan bir qatorda, siz uchun unit testlarini bajarish uchun pytest kabi framework ladan foydalanishingiz mumkin. Ushbu ma'ruzada siz ...

- Unit tests
- assert
- pytest