بنامخدا

نام: حسین احمدپور-سید رضا جاویده

موضوع: unit test python

مدرس: استاد یدالهی

- یونیت تست (unit test)

- به فرایند نوشتن و اجرای خودکار تست‌ها جهت اطمینان یافتن از اجرای مطابق انتظار
کدنویسی تابع‌ها گفته می‌شود.

- با این که شاید فکر کنید تست یونیت دوباره‌کاری محسوب می‌شود، اما در واقع این یک اقدام پیشگیرانه برای جلوگیری از بروز باگ پیش از وقوع آن است.

Unitچیست ؟

به کوچک‌ترین مؤلفه نرم‌افزاری ممکن در اپلیکیشن (مانند تابع، کلاس یا کامپوننت) گفته می‌شود.

تست‌های یونیت منفرد ما را مطمئن می‌سازند که کامپوننت مرکزی اپلیکیشن، مطابق انتظار عمل می‌کند و این که یک کامیت فیچر به یک بخش از اپلیکیشن موجب بروز مشکلی در بخش دیگر نمی‌شود. اگر چنین مسئله رخ دهد احتمالاً در یکی از کدهای قدیمی یا جدید باگ دارید یا یک تست ضعیف یا قدیمی نوشته‌اید.

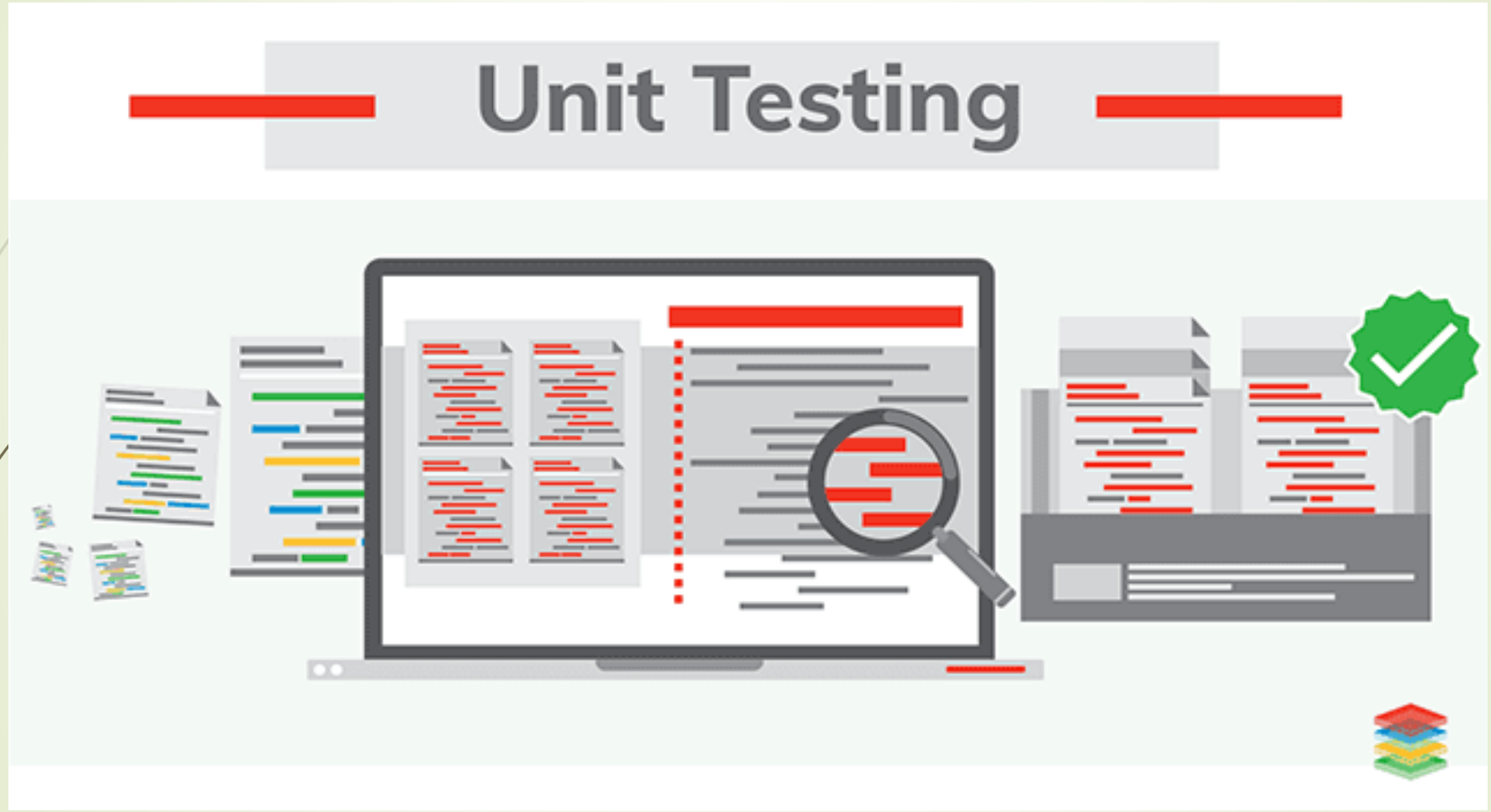
- هدف test unit

- هدف تست یونیت کاملاً روشن است و چیزی جز کاهش باگ نیست. تست یونیت به طور خاص باگ‌ها را هدف می‌گیرد که از یکپارچه‌سازیintegration نشات می‌گیرند.

- توسعه‌دهنده ممکن است فکر کند که همه چیز به صورت لوکال درست است و کدش را کامیت کند تا بفهمد آیا این کامیت موجب از کار افتادن اپلیکیشن می‌شود یا نه.

- تست یونیت موجب می‌شود بتوانیم بسیاری از این نقص‌ها را پیش از آن که واقع شوند، به دام بیندازیم و زمانی که آن را با روش‌های یکپارچه‌سازی پیوسته خودکارشده ادغام کنیم، می‌توانیم مطمئن باشیم که همه چیز به درستی کار خواهد کرد.

- اجرای تست یونیت منجر به کدبیس تمیزتر می‌شود



Unit test python framework

- 1-pyunit
- 2-pytest
- 3-Nose
- 4-Robot framework

- **شیوه اجرای تست یونیت**

- هر تست یونیت به طور معمول شامل سه مرحله است:

- **چیدمان Arrange** در این مرحله داده‌ها برای تست یونیت آماده می‌شوند. اگر لازم است که داده‌ها واکشی شوند، باید یک شیء پیچیده بسازید یا صرفاً مواردی که برای این مرحله لازم هستند آماده کنید.

- **عمل Act**در این مرحله یونیت فراخوانی می‌شود و پاسخ نیز لاگ خواهد شد.

- **تأکید Assert**در این مرحله عمده اتفاقات تست رخ می‌دهد. این همان جایی است که عملگرهای بولی بر مبنای آن نوشته می‌شوند.

- **ویژگی های pytest:**

- راحتی در assertion

- این مورد ویژگی خاصی نیست! ولی خب باعث راحتی کار میشه. دیگه موقع تست‌نویسی نیازی نیست که نوع برابری رو هم مشخص کنیم.
یعنی assertDict یا assertTrueو ... را نمی‌خواد و فقط با نوشتن assert می‌تونیم مقدار مورد انتظارمون رو با مقدار تست شده بررسی کنیم.

- به عنوان مثال تو unit test داشتیم:

```
from unittest import TestCase


class TestUserInput(TestCase):
      def test_input_values_validation_true(self):
            # stuff related to create input_values
           self.assertTrue(validate_values(input_values))
```

حالا اگر بخوایم این مورد را با pytest بنویسیم خواهیم داشت:

```
def test_input_values_validation_True():
      # stuff related to create input_values
      assert validate_values(input_values) == True
```

- اگر به این دو تیکه کد دقیق‌تر نگاه کنیم به یکی دیگه از ویژگی‌های pytestمی‌رسیم که اونم میشه:

- کدهای **Boilerplate**کمتر:

- اگر توجه کرده باشین تو تکه کد اول ما باید اول import unittestکنیم و بعد کلاس رو بنویسم و بعدش تازه می‌تونیم تست خودمون رو بنویسم ولی توی pytestحتی نیاز به importکردن هم نیست!! این مورد بعدا تو parameterizeهم خودش رو نشون میده.

- امکان **اجرای** تست های نوشته شده با **unittest** و **nose:**

- یکی از دلایلی که مهاجرت به pytestرو راحت و کم هزینه‌تر میکنه اینکه اگر قبلا تست‌هایی رو با unitestو یا noseنوشته باشیم، بدون هیچ نگرانی بابت اون‌ها شروع به نوشتن تست‌های جدید با pytestمی‌کنیم(واقعا امتیاز مهمیه).