Тестування

Що таке тести?

- Тест - це певний набір інструкцій виконання яких дає змогу перевірити правильну працездатність продукту.

Навіщо потрібні тести

- Протестований продукт зменшує ймовірність виникнення банальних (і не тільки) помилок, не правильно працюючий функціонал може принести збитки на великі суми, або навіть призвести до жертв.

THERAC 25


PATIENT NAME: John
TREATMENT MODE: FIX
BEAM TYPE: E
ENERGY (KeV): 10
ACTUAL
PRESCRIBED
UNIT RATE/MINUTE 0.000000 0.000000
MONITOR UNITS 200.000000 200.000000
TIME(MIN) 0.270000 0.270000
GANTRY ROTATION (DEG) 0.000000 0.000000 VERIFIED
COLLIMATOR ROTATION (DEG) 359.200000 359.200000 VERIFIED
COLLIMATOR X (CM) 14.200000 14.200000 VERIFIED
COLLIMATOR Y (CM) 27.200000 27.200000 VERIFIED
WEDGE NUMBER 1.000000 1.000000 VERIFIED
ACCESSORY NUMBER 0.000000 0.000000 VERIFIED
DATE: 2012-04-16
SYSTEM: BEAM READY
OP.MODE: TREAT
AUTO
TIME: 11:48:58
TREAT: TREAT PAUSE
X-RAY
173777
OPR ID: 033-tfs3p
REASON: OPERATOR
COMMAND:

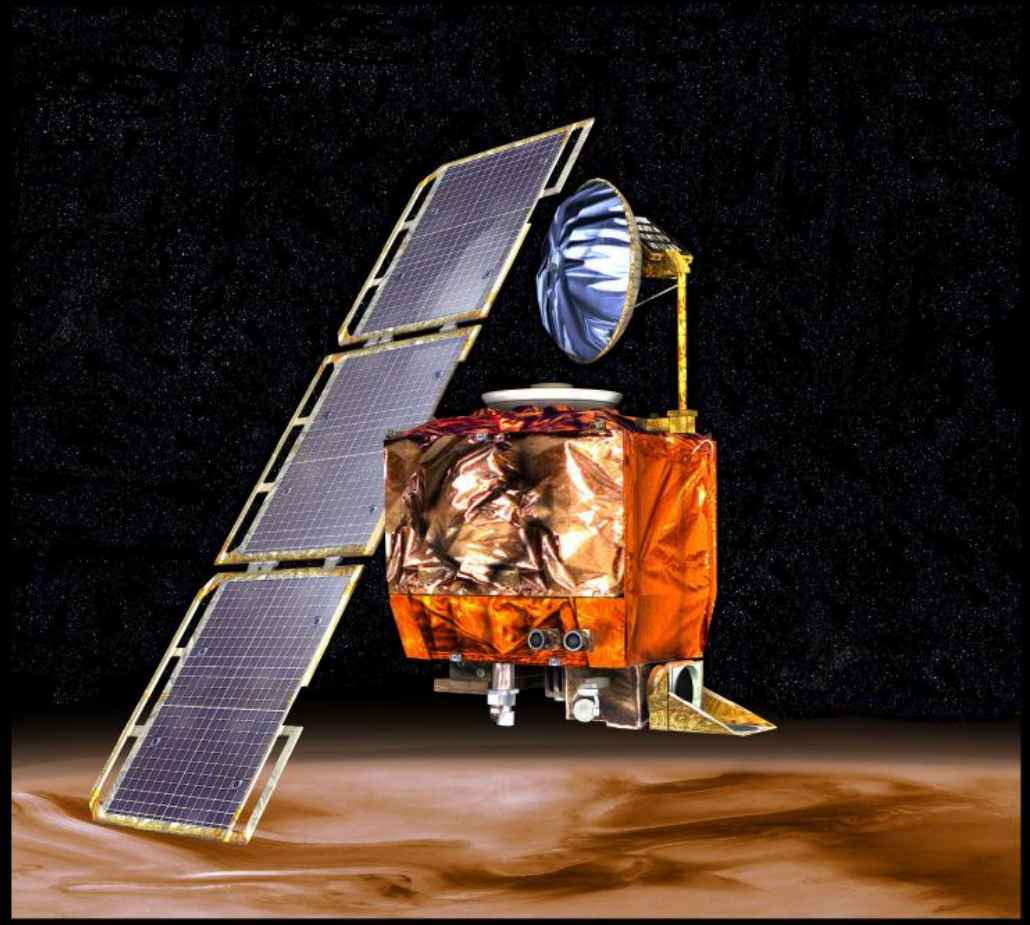
THERAC 25

- Перед вами інтерфейс програми THERAC 25. Так називався апарат для променевої терапії онкохворих і з ним все пішло вкрай невдало. В першу чергу у нього був невдалий інтерфейс. Дивлячись на нього, вже можна зрозуміти, що він не дуже хороший: лікарям було незручно забивати всі ці циферки. В результаті вони копіювали дані з карти попереднього пацієнта і намагалися правити тільки те, що потрібно було правити.

Проблеми в бекенді

- Ділення на нуль. Існував стан змінної, яку можна було задати дуже маленької величини. Відбувалося поділ на нуль, і після цього величина опромінення встановлювалася максимальною. Зрозуміло, що для людини це нічим добрим не закінчувалося.
- Стан гонки. У THERAC була змінна, яка відповідала за певну фізичну величину - поворот головки пристрою. Ця ж змінна використовувалася для побудови аналітичних даних. Припустимо, відбувається поворот, до кінця ще нічого не повернулось, але з змінної взялися якісь неправильні дані, щось на цих даних порахували - і пацієнт отримав неправильне лікування.

Mars Climate Orbiter



Mars Climate Orbiter

- Mars Climate Orbiter - апарат, який повинен був в атмосфері Марса зробити виміри атмосфери, подивитися, що там з кліматом.
- Але модуль, який був на землі, віддавав команди в системі СІ, в метричній системі. А модуль на орбіті Марса думав, що це британська система і неправильно це інтерпретував.
- В результаті модуль увійшов в атмосферу під неправильним кутом і зруйнувався.

Висновки з прикладів

- В результаті некоректної реалізації THERAC 25 офіційно загинуло 5 людей, не відомо скільки отримали шкоду для здоров'я
- В результаті прорахунків в Mars Climate Orbiter був зруйнований апарат. Збитків було на суму 125 млн дол


Можно избежать релиза
если ничего не передавать
в тестирование
memesmix.net

Хто займається тестуванням

- Тестувальники
- Програмісти

assert

- Інструкція assert призначена для порівняння значень. Працює вона по принципу if, але якщо умова не виконується то кидається ексепшин
- Приклад:
  assert sum(a, b) == c, "Error a + b != c"

- Дану інструкцію зачастку використовують як допоміжну, особливо на етапах прототипування продукту.

Як почати тестувати?

- Насправді, якщо ви думаєте що не займались тестуванням то ви помиляєтесь. Кожен запуск вашої програми це свого роду тест, запустилась значить - перший тест пройдений!
- Згадайте що робили ви з самого початку коли встановили Python? Ввели hello world. Це і був ваш перший тест.

doctest

- Це бібліотека Python, призначена для тестування документації. Чому це добре? Документація, яка написана в коді, має властивість дуже часто ламатися. Але коли у вас великий код, багато параметрів і ви в кінці щось дописали, то з дуже великою ймовірністю ви забудете поправити docstrings. Doctest дозволяє таких речей уникнути.

Приклад doctest

- Розглянемо такий код, в даному коді в докстрінгу помилковий тест
- ```
  def sum(a, b):
      """
      >>> Calc.sum(1, 1)
      3

      :param a:
      :param b:
      :return:
      """
      return a + b
  ```
  ```

- Якщо ми запустимо його з доктестом то отримаємо такий вивід

```
(venv) D:\Cursor>py -m doctest calc.py
27
**********************************************************************
File "D:\Cursor\calc.py", line 5, in calc.Calc.sum
Failed example:
    Calc.sum(1, 1)
Expected:
    3
Got:
    2
**********************************************************************
1 items had failures:
   1 of   1 in calc.Calc.sum
***Test Failed*** 1 failures.
```

- Написання доктестів це завдання програміста! Вони пишуться для самотестування написаної функції.
- В доктест є корисні директиви, наприклад *SKIP,* яка не запускає певний тест
- *Приклад:*
  ```
  >>> Calc.sum(1, 1) # doctest: +SKIP
  ```
- Деталі з іншими директивами: https://docs.python.org/3/library/doctest.html#doctest-directives

- Загалом доктест є інтегрованим тестом, тобто вони спрацьовують під час роботи скрипта і якщо вони не падають, то ви про їх роботу навіть не знатимете.

unittest

- Бібліотека Python призначена дял написання юніт тестів.
- юніт-тестування - процес в програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.
- Ідея полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвела чергова зміну коду до регресії, тобто до появи помилок в уже відтестованих місцях програми, а також полегшує виявлення і усунення таких помилок.

- Бібліотеку unittest не потрібно додатково скачувати.
- Написання тесту досить просте, створюємо клас який наслідується від `unittest.TestCase`
- Далі можемо використовувати методи для тестування, наприклад `self.assertEqual()`
- Загалом найпростіший клас виглядатиме ось так
- ```
  class TestA(unittest.TestCase):
      def test_sum(self):
          self.assertEqual(sum(1, 1), 2)
  ```
  ```

unittest.assertXXX

a == b	assertEqual(a, b)
a != b	assertNotEqual(a, b)
bool(x) is True	assertTrue(x)
bool(x) is False	assertFalse(x)
a is b	assertIs(a, b)
a is not b	assertIsNot(a, b)
x is None	assertIsNone(x)
x is not None	assertIsNotNone(x)
a in b	assertIn(a, b)
a not in b	assertNotIn(a, b)
isinstance(a, b)	assertIsInstance(a, b)
not isinstance(a, b)	assertNotIsInstance(a, b)

setUp, tearDown

- Фікстури це функції, які викликаються до або після виконання тесту. Вони потрібні, якщо тесту потрібно виконати спеціальне налаштування - створити тимчасовий файл після тесту, видалити тимчасовий файл; створити базу даних, видалити базу даних; створити базу даних, написати в неї щось
- В unittest фікстури представлені методами setUp, tearDown які виконуються до початку і після тесту відповідно

pytest

- Аналог unittest, потрібно додатково скачувати.
- Основна перевага - не завантажений інтерфейс ваших тестів.
- Для написання тестів потрібно зробити тест функцію і запустити файл командою:
- pytest test_pytest.py

Результат успіншого тесту

- ```
  def test_gg():
      assert 1 + 1 == 2
  ```
- Результат тестування

========== test session starts ==========

platform win32 -- Python 3.8.2, pytest-6.2.2, py-1.10.0, pluggy-0.13.1

rootdir: D:\Cursor\exception_logging

collected 1 item

calc_pytest.py

. [100%]

========== 1 passed in 0.01s ==========
  ```

Результат провaленого тесту

- ```
  def test_gg():
      assert 1 + 1 == 1
  ```
  ```

- ============================= test session starts ==============================
- platform win32 -- Python 3.8.2, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
- rootdir: D:\Cursor\exception_logging
- collected 1 item

- calc_pytest.py F [100%]

- ================================== FAILURES ===================================
- _____ test_gg _____

- def test_gg():
- > assert 1 + 1 == 1
- E assert (1 + 1) == 1

- calc_pytest.py:3: AssertionError
- =========================== short test summary info ============================
- FAILED calc_pytest.py::test_gg - assert (1 + 1) == 1
- ============================== 1 failed in 0.06s ===============================

Параметризація

- Можемо параметризувати тести, добавивши різні тестові випадки.
- Приклад
- ```
  @pytest.mark.parametrize(
      ("a", "b", "c"), [
          (1, 1, 2),
          (2, 1, 2),
          (2, 2, 4)
      ]
  )
  def test_gg(a, b, c):
      assert a + b == c
  ```
  ```

- Parametrize - хороша штука. І коли ви написали тест один раз, а далі робите багато-багато наборів параметрів - це хороша практика. Краще не робити багато варіантів коду на схожі тести, а один раз написати тест, далі зробити великий набір параметрів, і воно буде працювати

Selenium як інструмент тестування веб застосунків

- Selenium - веб драйвер для роботи з браузерами, дозволяє симулювати поведінку користувача, зробити парсинг сторінки, робити певні запити.
- Для python написана бібліотека яка дозволяє писати скрипти використовуючи селеніум

- `pip install selenium`
- Також для роботи знадобиться драйвер для вашого браузера
- На прикладі GoogleChrome
- Спочатку ідемо сюди chrome://settings/help і дивимось версію браузера
- Тут скачуємо відповідний драйвер
- https://chromedriver.chromium.org/downloads

Початок роботи

- Для початку потрібно отримати доступ до вебдрайвера. Кладемо скачаний вебдрайвер в папку з скриптом, або передаємо шлях `webdriver.Chrome("test\test\chromedriver")`

```
from selenium import webdriver

driver = webdriver.Chrome()
```

- `driver.get(url) -` даний метод переходить по вказаному `url`
- Далі вся робота заключається в парсингу сторінки, пошуку певних елементів, їх отримання і взаємодія з ними
- В прикладі нижче був знайдений певний елемент по css селектору, і потім здійснений клік по ньому
- ```
  elem = driver.find_element_by_css_selector("show-contacts")
  elem.click()
  ```
- `А в цьому прикладі була знайдена строка для введення в яку ввели свій текст`
- ```
  elem = driver.find_element_by_css_selector("input#search")
  elem.click()
  elem.send_keys("Hello")
  ```