

# 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  
TIME: 11:48:58  
OPR ID: 033-tfs3p

SYSTEM: BEAM READY  
TREAT: TREAT PAUSE  
REASON: OPERATOR

OP.MODE: TREAT AUTO  
X-RAY 173777  
COMMAND: █

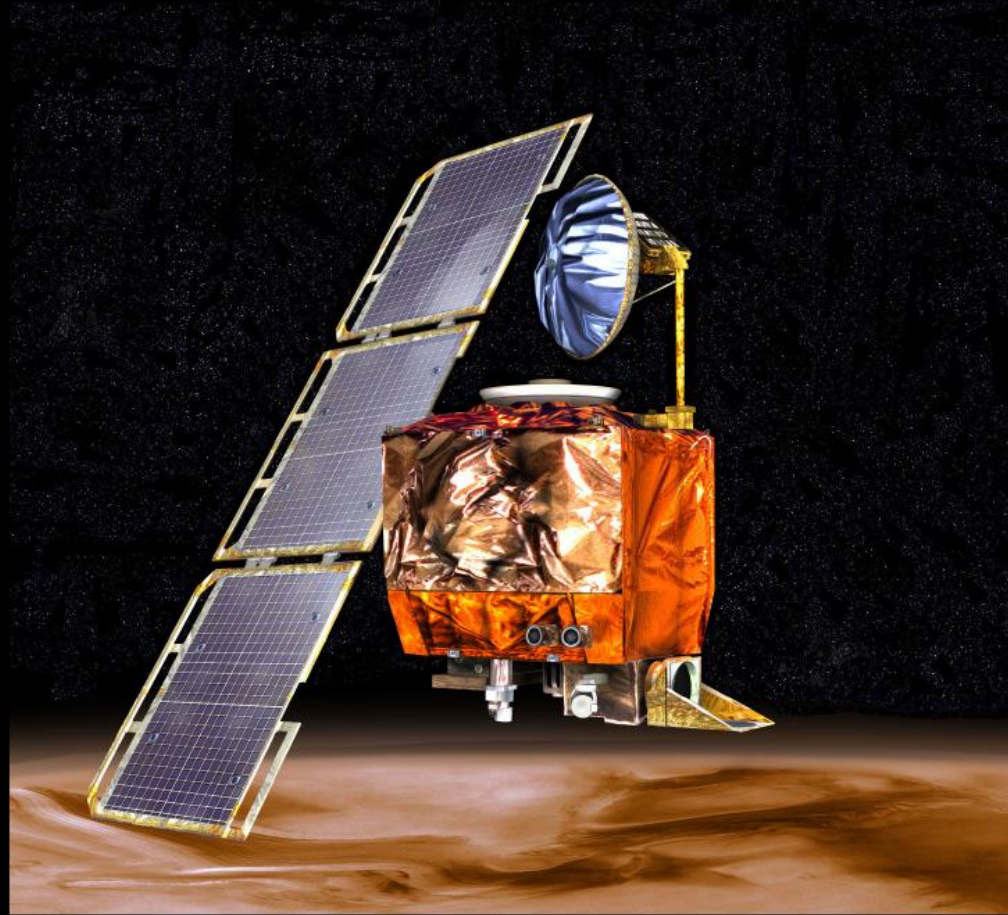
# THERAC 25

- 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 взяли 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 SI, 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

# Ще приклади відмоих багів

- На самому початку на Amazon можна було замовити негативне кількість товару. У такому випадку при покупці гроші не списувалися, а зараховувалися на карту клієнта. Така дурна помилка трапилася через те, що розробник магазину Джефф Безос намагався якомога частіше випускати оновлення для свого сервісу і у нього не вистачало часу на його тестування.

- Проблема Y2K ще відома як «проблема 2000 року». Розробники ПЗ в XX столітті часто використовували для позначення року в датах дві останні цифри замість звичних чотирьох. Наприклад, 7 квітня 1994 року уявлялося як «07.04.94».

# Перший в історії баг репорт

92

9/9

0800 Antan started

1000 " stopped - antan ✓

1300 (032) MP-MC { 1.2700 9.037 847 025  
2.130476415 } 9.037 846 995 correct

(033) PRO 2 2.130476415


correct 2.130676415

Relays 6-2 in 033 failed special speed test  
in relay " 10.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multi Adder Test.

1545  Relay #70 Panel F  
(moth) in relay.

First actual case of bug being found.

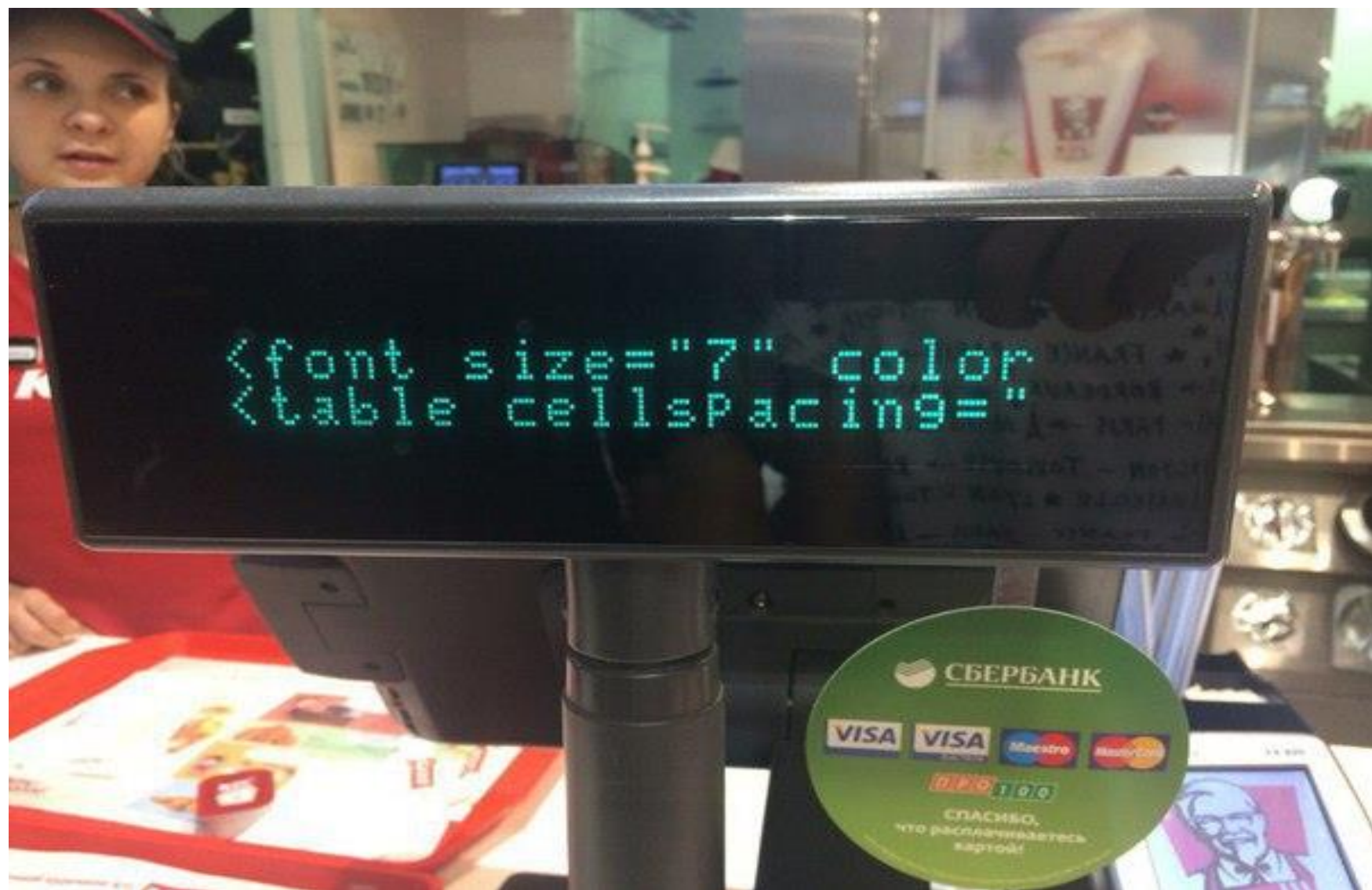
1630 Antan started.

1700 closed down.

- Один з 114 комутаторів АТ & Т був механічно пошкоджений. Він відправив повідомлення про поломку іншим АТС, а ті, в свою чергу, сусіднім. Запустилася ланцюгова реакція, яка поклала телефонну мережу на 9 годин, через що 60 тис. Людей залишилися без зв'язку.
- Причина була в недавньому оновленні ПЗ для АТС. Тепер, якщо ламався один з комутаторів, він посилав про це повідомлення сусіднім комутаторів, щоб ті перехоплювали його трафік. Тоді інші комутатори повинні були перезавантажуватися в новий режим роботи і періодично перевіряти, чи не заробив перший.
- Однак через баг в кодї несправний комутатор вислав по два повідомлення. Друге повідомлення досягало інших комутаторів прямо під час їх перезавантаження. Через це вони починали думати, що самі несправні і розсилали повідомлення далі по ланцюгу. Таким чином, немов доміно, впала ціла мережа. Загальні втрати компанії склали приблизно 60 млн. Доларів.

- Аварія ракети-носія «Аріан-5» сталася 4 червня 1996 Ракета зруйнувалася на 40-й секунді польоту через невірну роботи бортового ПО. Цей баг в системі - найдорожчий в історії - його збитки оцінюється від 360 до 500 млн. Доларів. В результаті аварії були також втрачені 4 супутника Cluster.
- Під час розробки бортового програмного забезпечення деякі фрагменти коду були взяті з ПО попередньої вдалою Ariane 4. Однак софт так і не протестували в новому оточенні. Після старту ракети її модуль спробував прорахувати певне значення, виходячи з горизонтальної швидкості ракети. Так як ця швидкість у Ariane 5 була набагато вище, ніж у попередньої версії, це викликало переповнення змінної, що, в свою чергу, спричинило помилки. На борту була запасна система управління, але т. К. В ній було ідентичне ПО, помилка повторилася.





# Котячий баг

## Bug Description

---

14.04, locked screen to go to lunch, upon return from lunch cat was sitting on keyboard, login screen was frozen & unresponsive.

To replicate: In unity hit ctrl-alt-l, place keyboard on chair. Sit on keyboard.

Resolution: Switched to virtual terminal, restarted lightdm, lost all open windows in X session.

What should have happened: lightdm not becoming unresponsive.



**Можно избежать релиза**



**если ничего не передавать  
в тестирование**

# Хто займається тестуванням

- Тестувальники
- Програмісти

# assert

- Інструкція `assert` призначена для порівняння значень. Працює вона по принципу `if`, але якщо умова не виконується то кидається екsepшин
- Приклад:  
`assert sum(a, b) == c, "Error a + b != c"`
- Дану інструкцію за частку 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 кодї 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 докестів це завдання програміста! Вони пишуться для самотестування написаної функції.
- В докест 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>





**-Видишь тестирование?**  
**-Нет**

- 
- 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.

**- И я нет... А оно есть**

# unittest

- Бібліотека Python призначена для 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

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

# 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 успішного 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 проваленного 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, додавши 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")`