

Ayudantía 4

Exceptions y Testing

¿Qué veremos hoy?

- Qué son las excepciones
- Tipos de error
- Cómo evitarlas
- Qué es testing
- Cómo testeo mi programa

Alumno durante un jueves cualquiera: ¿Ayudante, por qué se cae mi actividad? :c

Tipos de excepción:

- `SyntaxError` exception
- `NameError` exception
- `ZeroDivisionError` exception
- `IndexError` exception
- `TypeError` exception
- `AttributeError` exception
- `KeyError` exception
- `Exception`



```
try:
```

```
    # código
```

Aquí va nuestro código a probar

```
except:
```

```
    # Error
```

Aquí el error que esperamos

```
else:
```

```
    # Error
```

```
finally:
```

```
    # Error
```

```
try:
```

```
    # código
```

```
except:
```

```
    # Error
```

```
else:
```

```
    # Error
```

```
finally:
```

```
    # Error
```

**Esto se ejecutará siempre y cuando
no haya ninguna excepción**

**Se ejecutará siempre
¿de qué nos puede servir?**

Ejemplo 1

```
def dividir(num,den):  
    try:  
        return float(num)/float(den)  
    except (ZeroDivisionError) as err:  
        print('Error: {}'.format(err))  
        print('El denominador debe ser distinto de 0')
```

¿Cuál es el problema con esto? (Ejemplo 2)

```
def dividir(num,den):  
    try:  
        return float(num)/float(den)  
    except:  
        print('El denominador debe ser distinto de 0')
```

Ejemplo 3

```
while True:
    try:
        n = input("Ingrese un entero: ")
        n = int(n)
        break

    except Exception:
        print("Algo malo ocurrio...")

    except ValueError:
        print("Entero no valido, intente nuevamente...")

finally:
    print("Saliendo del programa")

print("Proceso terminado con exito")
```

¿Cuál será el output en los siguientes casos?

a) 'a'

b) 5

Testing

Python unittest

```
def dividir(num,den):  
    return float(num)/float(den)
```



¿Por qué hacer testing?

- Pone a prueba nuestro código
- Un testeo manual no es capaz de ponerse en **todos** los casos posibles
- Uso eficiente del tiempo
- *Untested code is broken code*



Importar unittest

```
import unittest
```

```
class Nombre(unittest.TestCase):
```

```
    def test_nombre_metodo(self):  
        self.assert...
```

```
unittest.main()
```

```
import unittest
```

```
class Nombre(unittest.TestCase):
```

```
    def test_nombre_metodo(self):  
        self.assert...
```

```
unittest.main()
```

Heredar de unittest.TestCase

```
import unittest
```

```
class Nombre(unittest.TestCase):
```

```
    def test_nombre_metodo(self):  
        self.assert...
```

**Los métodos para testear
deben comenzar con 'test'**

```
unittest.main()
```

```
import unittest
```

```
class Nombre(unittest.TestCase):
```

```
    def test_nombre_metodo(self):  
        self.assert...
```

True, Equals, IsInstance, In, etc.

```
unittest.main()
```

```
import unittest

class Nombre(unittest.TestCase):

    def test_nombre_metodo(self)
        self.assert...
```

Cómo correrlo :)

```
unittest.main()
```

Ejemplo

```
import unittest
```

```
class TestAlgunosAsserts(unittest.TestCase):
```

```
    def test_ejemplos_asserts(self):
```

```
        self.lista = [1, 2, 3, 4, 5, 6]
```

```
        a = 1
```

```
        b = "Entero no válido"
```

```
        self.assertEqual(1, 1.0)
```

```
        self.assertIn(a, self.lista)
```

```
        self.assertIsInstance(a, int)
```


Ejemplo

```
import unittest
```

```
class TestAlgunosAsserts(unittest.TestCase):
```

```
    def test_ejemplos_asserts(self):
```

```
        self.lista = [1, 2, 3, 4, 5, 6]
```

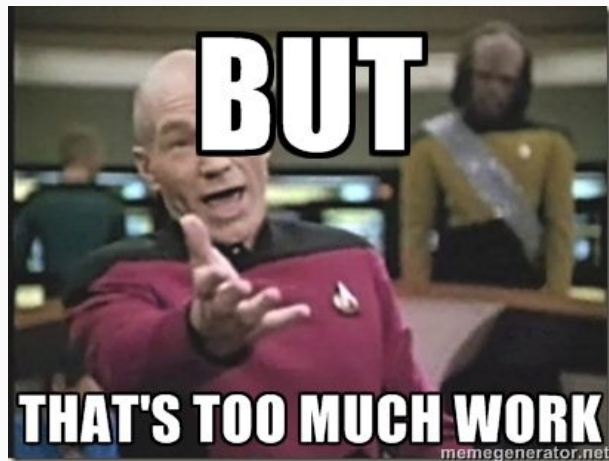
```
        a = 1
```

```
        b = "Entero no válido"
```

```
        self.assertEqual(1, 1.0)
```

```
        self.assertIn(a, self.lista)
```

```
        self.assertIsInstance(a, int)
```



Para hacer menos engorroso cada test, podemos setear y borrar variables con dos métodos que reconoce unittest

setUp

tearDown

(código)



¿Cómo ignoro tests?

```
@unittest.expectedFailure  
def test_...
```

```
    #asserts
```

```
@unittest.skip('Reason')    # skipUnless(condicion, 'Razón')  
def test_...                # skipIf(condicion, 'Razón')
```

```
    #asserts
```

¿Cuál es el resultado del test?

```
Testing started at 17:46 ...  
xuFs.s
```



¡Resolvamos una Actividad de Testing!

(código ac13)

¡Creando nuestras propias excepciones!

(código actividad 12)

Como sabemos que eres experto teniendo errores, hemos decidido que puedas crear tus propios errores

