Ayudantía 6

Exceptions y Testing

¿Qué veremos hoy?

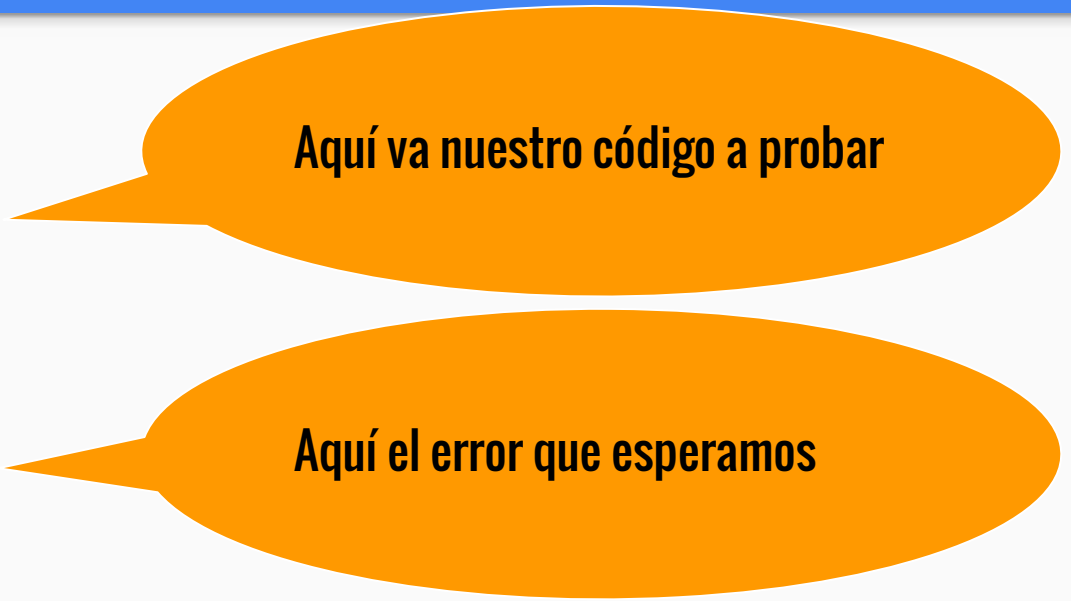
- Qué son las excepciones
- Tipos de error
- Cómo evitarlas
- Qué es testing
- ¿Cómo testeo mi programa?
- Repaso AC Metaclases

Alumno durante un jueves cualquiera: ¿Ayudante porque 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
except:
    # Error
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

```
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?

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

¡ Creando nuestras propias excepciones!

(código)

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

Ejercicio

```
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*


```
import unittest

class Nombre(unittest.TestCase):

    def test_nombre_metodo(self)
        self.assert…



suite =
unittest.TestLoader().loadTestsFromTestCase(IgnorarTests)
unittest.TextTestRunner().run(suite)
```

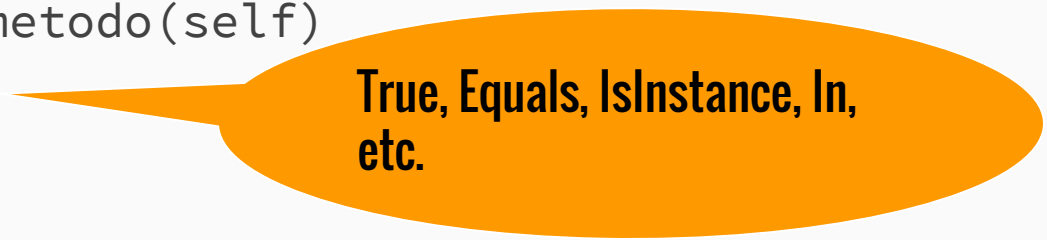
```
import unittest

class Nombre(unittest.TestCase):

    def test_nombre_metodo(self)
        self.assert…


suite =
unittest.TestLoader().loadTestsFromTestCase(IgnorarTests)
unittest.TextTestRunner().run(suite)
```

Heredar de unittest.TestCase

```
import unittest

class Nombre(unittest.TestCase):

    def test_nombre_metodo(self)
        self.assert…


suite =
unittest.TestLoader().loadTestsFromTestCase(IgnorarTests)
unittest.TextTestRunner().run(suite)
```

Los métodos para testear deben comenzar con ‘test’

```
import unittest

class Nombre(unittest.TestCase):

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




```
suite =
unittest.TestLoader().loadTestsFromTestCase(IgnorarTests)
unittest.TextTestRunner().run(suite)
```

```
import unittest

class Nombre(unittest.TestCase):

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

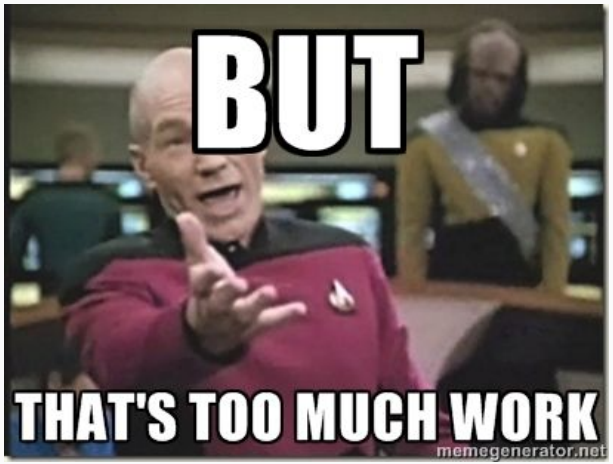

```
suite =
unittest.TestLoader().loadTestsFromTestCase(IgnorarTests)
unittest.TextTestRunner().run(suite)
```

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 valido”
        self.assertEquals(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 valido”
        self.assertEquals(1, 1.0)
        self.assertIn(a, self.lista)
        self.assertIsInstance(a, int)
```



setUp

tearDown

(código)

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

¿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
```

¡Probemos nuestro programa!

(código)

AC07 Metaclases

```
class MiMetaClase(type):

  def __new__(cls, name, bases, dic):
    return super().__new__(cls, name,
bases, dic)

  def __call__(cls, *args, **kwargs):
    return super().__call__(*args,
**kwargs)
```

```
class MiClase(metaclass= MiMetaClase):
    def __init__(self, *args):
        # código
```

¿Cuál es la diferencia? (código)

```
class MiMetaClase(type):

  def __new__(cls, name, bases, dic):
    return super().__new__(cls, name,
bases, dic)

  def __call__(cls, *args, **kwargs):
    return super().__call__(*args,
**kwargs)
```

```
class MiClase(metaclass= MiMetaClase):
    def __init__(self, *args):
        # código
```