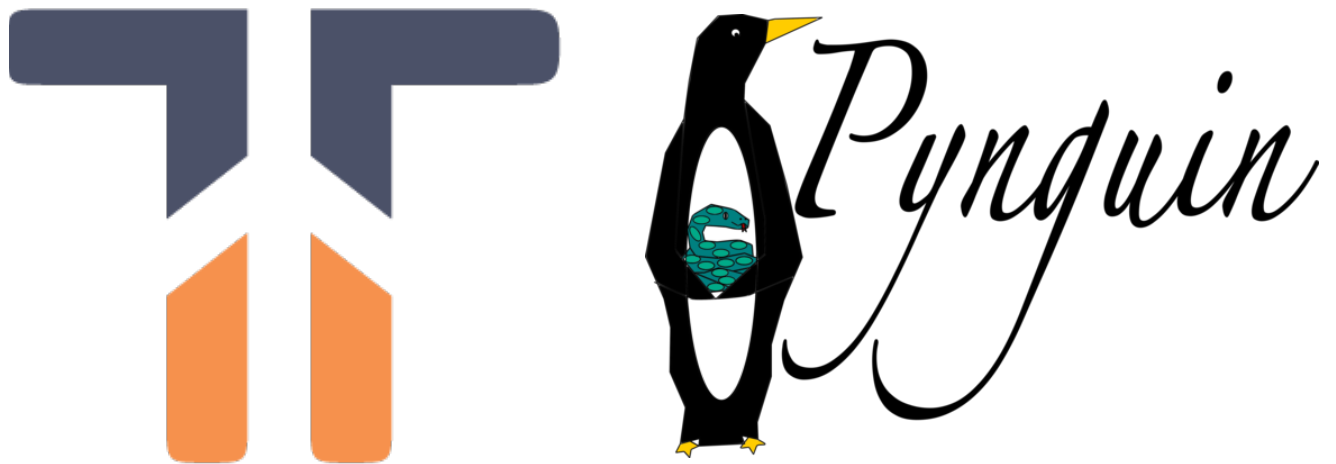




Tarea 4



Alumnos: Juan Vicuña Gebauer y Joaquín Errázuriz

Profesora: Alison Fernández Blanco

Screenshot Mutatest:

```
Overall mutation trial summary
=====
- DETECTED: 20
- TIMEOUT: 1
- SURVIVED: 4
- TOTAL RUNS: 25
- RUN DATETIME: 2023-04-25 16:07:05.668606

DETECTED
-----
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.Mod'>
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.Div'>
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.Add'>
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.Pow'>
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.FloorDiv'>
- src\clock_display.py: (1: 11, c: 25) - mutation from <class 'ast.Sub'> to <class 'ast.Mult'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.And'> to <class 'ast.Or'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.GtE'> to <class 'ast.Eq'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.GtE'> to <class 'ast.LtE'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.GtE'> to <class 'ast.Gt'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.GtE'> to <class 'ast.Lt'>
- src\clock_display.py: (1: 12, c: 14) - mutation from <class 'ast.GtE'> to <class 'ast.NotEq'>
- src\clock_display.py: (1: 13, c: 12) - mutation from AugAssign_Sub to AugAssign_Add
- src\clock_display.py: (1: 13, c: 12) - mutation from AugAssign_Sub to AugAssign_Mult
- src\clock_display.py: (1: 13, c: 12) - mutation from AugAssign_Sub to AugAssign_Div
- src\display_number.py: (1: 16, c: 11) - mutation from <class 'ast.Lt'> to <class 'ast.LtE'>
- src\display_number.py: (1: 16, c: 11) - mutation from <class 'ast.Lt'> to <class 'ast.NotEq'>
- src\display_number.py: (1: 16, c: 11) - mutation from <class 'ast.Lt'> to <class 'ast.Gt'>
- src\display_number.py: (1: 16, c: 11) - mutation from <class 'ast.Lt'> to <class 'ast.GtE'>
- src\display_number.py: (1: 16, c: 11) - mutation from <class 'ast.Lt'> to <class 'ast.Eq'>

2023-04-25 16:07:05,673: Timedout mutations:

TIMEOUT
-----
- src\display_number.py: (1: 8, c: 21) - mutation from <class 'ast.Mod'> to <class 'ast.Pow'>

2023-04-25 16:07:05,673: Surviving mutations:

SURVIVED
-----
- src\clock_display.py: (1: 7, c: 39) - mutation from None to True
- src\display_number.py: (1: 3, c: 41) - mutation from None to True
- src\display_number.py: (1: 8, c: 22) - mutation from <class 'ast.Add'> to <class 'ast.Mod'>
- src\display_number.py: (1: 21, c: 15) - mutation from <class 'ast.Lt'> to <class 'ast.NotEq'>
```

Justificacion Mutatest:

En la definición del método `__init__` de las clases `clock_display` y `display_number` se menciona mediante una sintaxis de flecha el resultado retornado al ejecutarse dicho método. En este escenario fue definido como `None`, sin embargo, cambiar esta sintaxis no influye en el funcionamiento del código en lo absoluto, por lo que los primeros 2 mutantes son justificados como insignificantes.

Luego tenemos que el método `increase()` de `NumberDisplay`, ocurre que en las siguientes expresiones:

- `self.value = (self.value + self.limit + 1) % self.limit`
- `self.value = (self.value % self.limit + 1) % self.limit`

El resultado final será el mismo, ya que sumar o modularizar `self.value` por `self.limit` son operaciones equivalentes cuando se modulariza nuevamente por `self.limit`. De esta manera justificamos como insignificantes el tercer mutante sobreviviente.

Finalmente, podemos notar como el valor de cualquier instancia de `display number` nunca va a ser igual o mayor al del límite por cómo está definido en el método que lo incrementa (modularizando por el límite, dejándolo en un rango de `[0, self.limit - 1]`, por lo que la operación de invariante puede evaluar que sean distintos y se obtendrá el mismo resultado. De esta manera justificamos como insignificantes el cuarto mutante sobreviviente.

Por último, podemos justificar el mutante tipo `TimeOut`, con la idea que reemplazar el modulo por un elevado en el método incrementar de `display_number`, lógicamente no resultaría en lo esperado, al tener un número que se escapa del rango `[0, self.limit - 1]`.

Test Smells:

- **ClockFactory:**

Eager test: El caso de prueba `test_case_0` solo crea una instancia del objeto `ClockFactory` y no realiza ninguna prueba. Se puede considerar como una prueba innecesaria o "eager" que solo está ahí para llenar espacio.

Assertion roulette: Ninguno de los casos de prueba realiza ninguna comprobación o aserción.

Ejemplos:

Eager test:

```
def test_case_0():  
    clock_factory_0 = module_0.ClockFactory()
```

Assertion roulette:

```
def test_case_0():  
    clock_factory_0 = module_0.ClockFactory()
```

- **DisplayNumber:**

Test run war: Los casos de prueba están marcados con el decorador `pytest.mark.xfail(strict=True)`, lo que significa que se espera que fallen. Sin embargo, es posible que estos casos de prueba se estén ejecutando y ralentizando la suite de pruebas innecesariamente.

Assertion roulette: Los casos de prueba no realizan suficientes aserciones para verificar que el módulo bajo prueba funciona correctamente.

Ejemplos:

```
@pytest.mark.xfail(strict=True)
```

Test run war:

```
def test_case_1():  
    bool_0 = False
```

```

number_display_0 = module_0.NumberDisplay(bool_0, bool_0)
dict_0 = {
    number_display_0: number_display_0,
    number_display_0: number_display_0,
    number_display_0: number_display_0,
}
number_display_1 = module_0.NumberDisplay(dict_0, number_display_0)
var_0 = number_display_1.reset()
assert number_display_1.value == 0
number_display_1.increase()

```

Assertion roulette:

```

def test_case_4():
    float_0 = 381.28695
    number_display_0 = module_0.NumberDisplay(float_0, float_0)
    var_0 = number_display_0.str()
    assert var_0 == "381.28695"
    var_0.reset()

```

- **ClockDisplay:**

Assertion Roulette: En algunos casos de prueba, como test_case_2, test_case_3, test_case_5, test_case_7 y test_case_9, no se realizan comprobaciones o aserciones para verificar el comportamiento del código probado.

Eager Test: El caso de prueba test_case_0 solo crea una instancia del objeto ClockDisplay y no realiza ninguna comprobación o aserción. Este caso de prueba se puede considerar como una prueba innecesaria o "eager" que solo está ahí para llenar espacio.

General Fixture: En algunos casos de prueba se utilizan estructuras de datos generales, como dict o tuple, que no están específicamente relacionadas con la funcionalidad del código probado, lo que hace que las pruebas sean menos específicas y más propensas a errores.

Test Code Duplication: En algunos casos de prueba se repite código innecesariamente, como en test_case_2, donde se llama a var_0 = clock_display_0.increment() dos veces seguidas.

Ejemplos:

Assertion Roulette:

```

def test_case_3():
    tuple_0 = ()
    clock_display_0 = module_0.ClockDisplay(tuple_0)

```

Eager Test:

```

def test_case_0():
    bool_0 = True
    dict_0 = {bool_0: bool_0, bool_0: bool_0}
    clock_display_0 = module_0.ClockDisplay(dict_0)

```

```
var_0 = clock_display_0.increment()
```

General Fixture:

```
def test_case_4():  
    bool_0 = True  
    tuple_0 = (bool_0, bool_0, bool_0)  
    clock_display_0 = module_0.ClockDisplay(tuple_0)  
    var_0 = clock_display_0.increment()  
    var_1 = clock_display_0.str()  
    assert var_1 == "00:00:00"
```

Test Code Duplication:

```
def test_case_2():  
    bool_0 = True  
    dict_0 = {bool_0: bool_0, bool_0: bool_0}  
    clock_display_0 = module_0.ClockDisplay(dict_0)  
    var_0 = clock_display_0.increment()  
    var_1 = clock_display_0.invariant()  
    var_0.str()
```