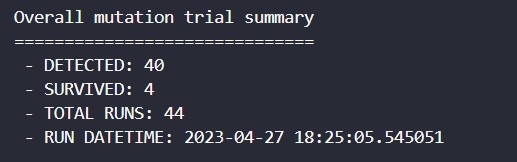
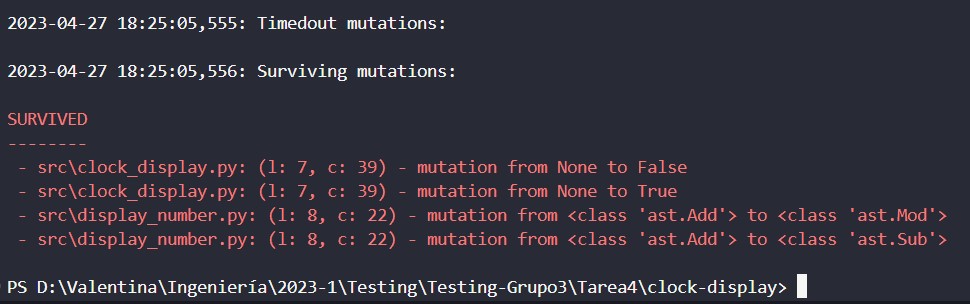
# Resporte Mutaciones





- src\clock\_factory.py: (l: 8, c: 26) - mutation from None to True/False y src\display\_number.py: (l: 3, c: 41) - mutation from None to True/False

Estas mutaciones se hacen en la línea de la definición del método init (def \_\_init\_\_(self) -> None). Se está indicando que el método retorna None, por lo que la mutación a True/False indicaría que el método retorna el booleano correspondiente. Por la forma en que está construido Python, esta mutación no es necesario eliminarla ya que cuando un método \_\_init\_\_() retorna algo, se levanta un error de TypeError lo que no permitiría la ejecución del código. <https://docs.python.org/3/reference/datamodel.html?highlight=__init__#object.__init__>

- src\display\_number.py: (l: 8, c: 22) - mutation from <class 'ast.Add'> to <class 'ast.Sub'>

Esta mutación ocurre en el método increase de la clase NumberDisplay indicando que el primer “+” es cambiado por un “-” en “(self.value + self.limit + 1) % self.limit”. Tests no fallan dada la naturaleza de la operación módulo (división con resto), tal que A%B = (A+B)%B = (A-B)%B, con el resto restringido a 0<= r < B. Así, considerando A = value + 1, se puede entender entonces el por qué la mutación no provoca que la prueba falle.

- src\display\_number.py: (l: 8, c: 22) - mutation from <class 'ast.Add'> to <class 'ast.Mod'>

Mutación indica que el primer “+” es cambiado por un “%”. Esto no provoca que la prueba falle en tanto sacar el módulo a un número, sumarle uno y a ese resultado volver a sacarle el módulo respecto del mismo número, es equivalente a tomar la suma de un número y uno, y a ese resultado sacarle el módulo. EJ: (3%7+1)%7 = 4 = (3+7+1)%7 = (3+1)%7 donde se usó la idea de la mutación anterior. Notar funciona para todo valor limit>0 respetando las propiedades de la división con resto.

**REPORTE TESTS SMELLS ACTIVDAD 3 USO PYNGUIN**

MÓDULO ‘display\_number.py’

MÓDULO ‘clock\_display.py’

* En test\_case\_0 y test\_case\_5 encontramos código en líneas siguientes a donde se declara assert. Tal que este código no tiene sentido en tanto no aporta nada a los tests ya que la correctitud de la performance de esas líneas no es evaluada.
* En test\_case\_1, test\_case\_2, test\_case\_3 y test\_case\_4 no poseen assert. Por lo tanto, no se está evaluando la correcta performance del código anidado en estos.

Además de lo anterior, analizando los módulos de forma particular podemos decir que:

* En test\_case\_0:
  + A ClockDisplay le entregan un número y no un iterable que no tiene mucho sentido en tanto espera un iterable, no importa si es de un elemento, pero debe ser un iterable. Que el número entregado sea en formato binario no es problema.
  + Correctitud de var\_0 y var\_2 jamás es evaluada.
  + Variable var\_0 jamás es ocupada.
  + De haberse entregado bytes\_0 dentro de una lista a ClockDisplay el test siempre falla en tanto si solo hay un iterable, el resultado será algo en el formato hh y no hh:mm que es con lo que están realizando la comparación con var\_1.
* En test\_case\_1:
  + Mismo problema que antes con dato entregado a ClockDisplay que en test\_case\_0.
  + Variable var\_0 jamás es ocupada.
* En test\_case\_2:
  + Mismo problema que antes con dato entregado a ClockDisplay que en test\_case\_0 y test\_case\_1.
  + Se entrega un objeto ClockDisplay como parámetro para instanciar otro ClockDisplay. Lo cual es incorrecto en tanto espera otro tipo de dato como entrada y no tiene sentido en el contexto de nuestro problema pues no es algo que vaya a ocurrir pues no es un caso de uso.
  + Variable var\_0 jamás es ocupada.
* En test\_case\_3:
  + A ClockDisplay se le entrega un diccionario vacío. Corresponde porque es un iterable, pero este diccionario va vacio. No obtenemos información de esto pues no se evalua que efectivamente no se retorne nada pues no hay sentencia assert.
  + No tiene sentido aplicar el método increment() sobre un booleano, que es el tipo de dato var\_0 al tomar el valor de retorno de una aplicación de increment() sobre un ClockDisplay.
* En test\_case\_4:
  + A ClockDisplay se le entrega un iterable de booleanos. Esto no es lo esperado en tanto no representa un caso de uso del código del proyecto y no tiene sentido para construir una instancia del objeto en cuestión.
  + Variable var\_0 jamás es ocupada.
* En test\_case\_5:
  + Mismo problema que antes con dato entregado a ClockDisplay que en test\_case\_0, test\_case\_1 y test\_case\_2. Por lo que caso esperado en assert jamás va ocurrir, pues el Clock siempre tendrá el formato “hh”
  + Variable var\_1 jamás es ocupada.
  + Se aplica método .str() sobre var\_2 no tiene sentido en tanto no espera que el objeto de entrada ya sea un string.