Introducción a la Programación

Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Testing - cubrimiento

Ejercicio 9

Sea la siguiente especificación del problema de `sumar` y una posible implementación en lenguaje imperativo:

`problema sumar` (in x: $\mathbb{Z}$, in y: $\mathbb{Z}$) : $\mathbb{Z}$ $\{$
  `requiere:` $\{True\}$
  `asegura:` $\{result = x + y\}$
$\}$

```
def sumar(x: int , y: int) -> int:
L1:    sumando: int = 0
L2:    abs_y: int = 0
L3:    if y < 0:
L4:      sumando = -1
L5:      abs_y = -y
       else:
L6:      sumando = 1
L7:      abs_y = y
L8:    result: int = x
L9:    count: int = 0
L10:   while(count < abs_y):
L11:      result = result + sumando
L12:      count = count + 1
L13:   return result
```

Ejercicio 9

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un test suite que ejecute todas las líneas del programa `sumar`.

Tests de unidad en Python

Para armar tests de unidad en Python debemos usar alguna biblioteca (como usábamos HUnit en Haskell). Algunas muy utilizadas son:

- `unittest`: viene con Python, es un poco más compleja la sintaxis (se usan clases). Es lo que ya usamos para testing de caja negra.
- `pytest`: es necesario instalarla, es más simple de utilizar

Cubrimiento

Para poder visualizar el cubrimiento de líneas de nuestro test suite, tendremos que instalar `coverage`.

En Linux: `pip install coverage`

Para ver como instalarlo en otros sistemas operativos y opciones de cómo ejecutarlo:
https://devguide.python.org/testing/coverage/#install-coverage

Cubrimiento

Es posible ver el resultado por consola o en un archivo html:

- Para calcular la cobertura, se debe indicar qué archivo queremos evaluar con --include y luego pasar el test suite a ejecutar, con el siguiente comando:
  coverage run –include=archivo-implementacion.py -m unittest archivo-testsuite.py
- Ejemplo: si tenemos nuestra implementación en el archivo ‘ejercicios.py‘ y los casos de test en test_ejercicios.py, debemos usar:
  `coverage run --include=ejercicios.py -m unittest test_ejercicios.py`
- Una vez ejecutado el comando anterior, podemos ver en consola el reporte con:
  coverage report

```
Name             Stmts   Miss  Cover
------------------------------------
ejercicios.py       14      0   100%
------------------------------------
TOTAL               14      0   100%
```

- `También podemos generar el reporte en un archivo html con:`
  `coverage html`

Cubrimiento

Podemos visualizar el cubrimiento de ramas (branch coverage) pasando el parámetro --branch:

- `coverage run -- branch --include=archivo-implementacion.py -m unittest archivo-testsuite.py`
- Ejemplo: `coverage run --branch --include=ejercicios.py -m unittest test_ejercicios.py`
- **Importante 1**: Cada vez que hagamos una modificación en el código o en los casos de test, debemos ejecutar `coverage run....` Luego podemos ejecutar `coverage html` para visualizar la información actualizada.
- **Importante 2**: Al definir cada caso de test en unittest, el nombre de la función debe comenzar con el string 'test'. Caso contrario, no se ejecutará ese caso, ya que unittest asume que no es un caso de test, sino una función auxiliar.

Cubrimiento

Veamos un ejemplo de cobertura con un solo caso de test:

test_ejercicios.py

```python
import unittest
from ejercicios import sumar

class EjerciciosTest(unittest.TestCase):

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_cero(self):
        self.assertEqual(sumar(3,0), 3, "test_sumar_cero")
```

Si ejecutamos:

```
coverage run --branch -m unittest test_ejercicios.py
coverage report -m ejercicios.py // -m permite ver las lineas no cubiertas
```

nos muestra:

```
Name             Stmts   Miss Branch BrPart  Cover   Missing
------------------------------------------------------------
ejercicios.py       14      4      4      2    67%   6-7, 15-16
------------------------------------------------------------
TOTAL               14      4      4      2    67%
```

¿Cubrimiento de líneas?

Cubrimiento

Veamos un ejemplo de cobertura con un solo caso de test:

test_ejercicios.py

```python
import unittest
from ejercicios import sumar

class EjerciciosTest(unittest.TestCase):

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_cero(self):
        self.assertEqual(sumar(3,0), 3, "test_sumar_cero")
```

Si ejecutamos:

```
coverage run --branch -m unittest test_ejercicios.py
coverage report -m ejercicios.py // -m permite ver las lineas no cubiertas
```

nos muestra:

```
Name             Stmts   Miss Branch BrPart  Cover   Missing
------------------------------------------------------------
ejercicios.py       14      4      4      2    67%   6-7, 15-16
------------------------------------------------------------
TOTAL               14      4      4      2    67%
```

¿Cubrimiento de líneas? 71.4 %
¿Cubrimiento de ramas?

Cubrimiento

Veamos un ejemplo de cobertura con un solo caso de test:

test_ejercicios.py

```python
import unittest
from ejercicios import sumar

class EjerciciosTest(unittest.TestCase):

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_cero(self):
        self.assertEqual(sumar(3,0), 3, "test_sumar_cero")
```

Si ejecutamos:

```
coverage run --branch -m unittest test_ejercicios.py
coverage report -m ejercicios.py // -m permite ver las lineas no cubiertas
```

nos muestra:

```
Name             Stmts   Miss Branch BrPart  Cover   Missing
------------------------------------------------------------
ejercicios.py       14      4      4      2    67%   6-7, 15-16
------------------------------------------------------------
TOTAL               14      4      4      2    67%
```

¿Cubrimiento de líneas? 71.4 %
¿Cubrimiento de ramas? 50 %

Cubrimiento

Veamos un ejemplo de cobertura con un solo caso de test:

test_ejercicios.py

```python
import unittest
from ejercicios import sumar

class EjerciciosTest(unittest.TestCase):

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_cero(self):
        self.assertEqual(sumar(3,0), 3, "test_sumar_cero")
```

Si ejecutamos:

```
coverage run --branch -m unittest test_ejercicios.py
coverage report -m ejercicios.py // -m permite ver las lineas no cubiertas
```

nos muestra:

```
Name             Stmts   Miss Branch BrPart  Cover   Missing
------------------------------------------------------------
ejercicios.py       14      4      4      2    67%   6-7, 15-16
------------------------------------------------------------
TOTAL               14      4      4      2    67%
```

¿Cubrimiento de líneas? 71.4 %
¿Cubrimiento de ramas? 50 % Deben tener al menos 95 % en ambos para el TP!

Cubrimiento

Agregamos un nuevo caso de test, corremos el coverage y pedimos el reporte:

```python
import unittest
from ejercicios import sumar

class EjerciciosTest(unittest.TestCase):

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_cero(self):
        self.assertEqual(sumar(3,0), 3, "test_sumar_cero")

    Tabnine | Edit | Test | Explain | Document
    def test_sumar_neg(self):
       self.assertEqual(sumar(3,-2), 1, "test_sumar_neg")
```

```
Name             Stmts   Miss Branch BrPart  Cover   Missing
------------------------------------------------------------
ejercicios.py       14      0      4      0   100%
------------------------------------------------------------
TOTAL               14      0      4      0   100%
```

Ejercicio 11

Sea el siguiente programa que retorna diferentes valores dependiendo si a, b y c, definen lados de un triángulo inválido, equilátero, isósceles o escaleno.

```
def triangle(a: int , b: int, c: int) -> int:
L1:     if(a <= 0 or b <= 0 or c <= 0):
L2:         return 4 # invalido
L3:     if(not ((a + b > c) and (a + c > b) and (b + c > a))):
L4:         return 4 # invalido
L5:     if(a == b and b == c):
L6:         return 1 # equilatero
L7:     if(a == b or b == c or a == c):
L8:         return 2 # isosceles
L9:     return 3  # escaleno
```

- Describir el diagrama de control de flujo (control-flow graph) del programa `triangle`.
- Escribir un test suite que ejecute todas las líneas y todos los branches del programa.
- usar la herramienta coverage para visualizar las lineas cubiertas