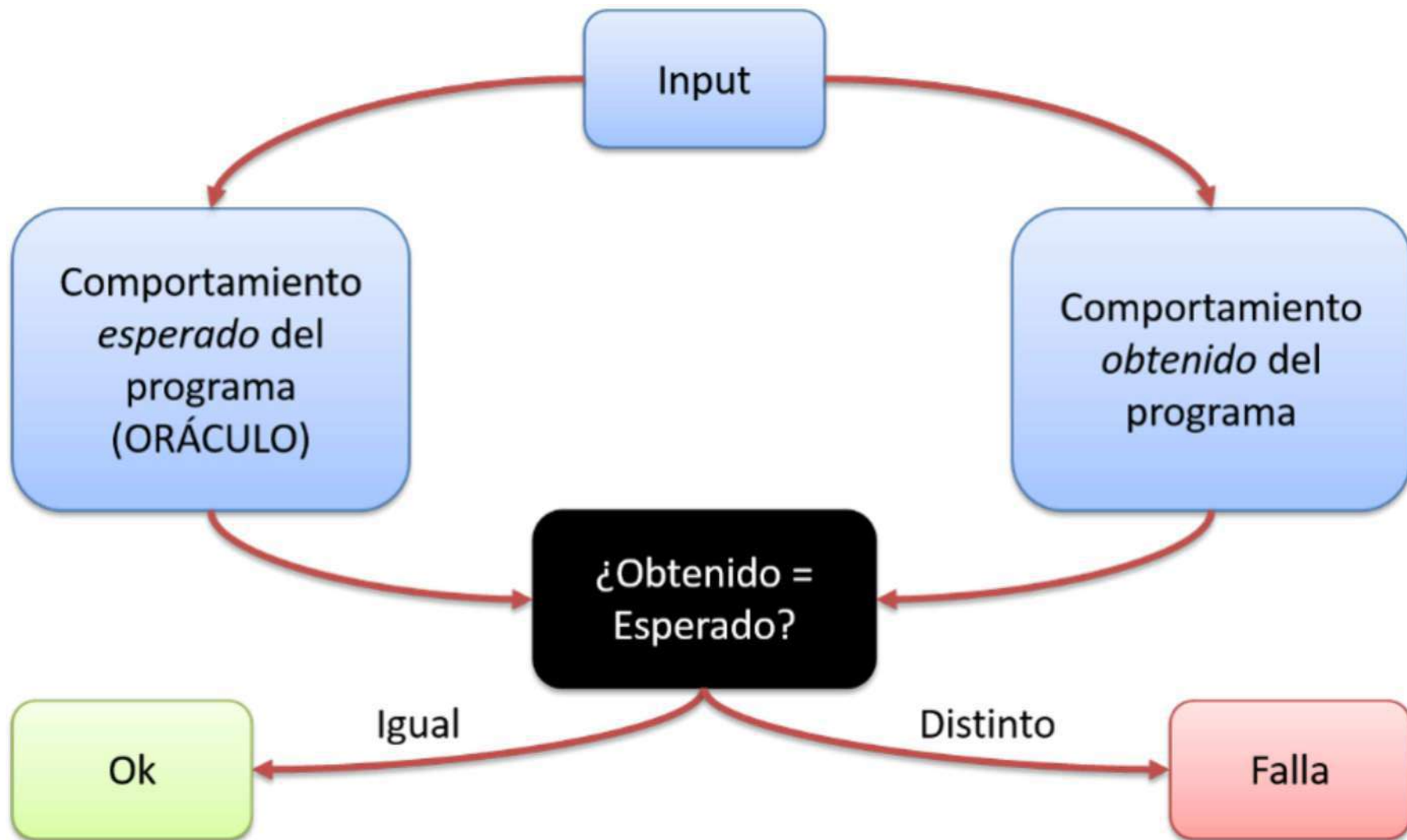


Testing Estructural o Caja Blanca

Introducción a la Programación

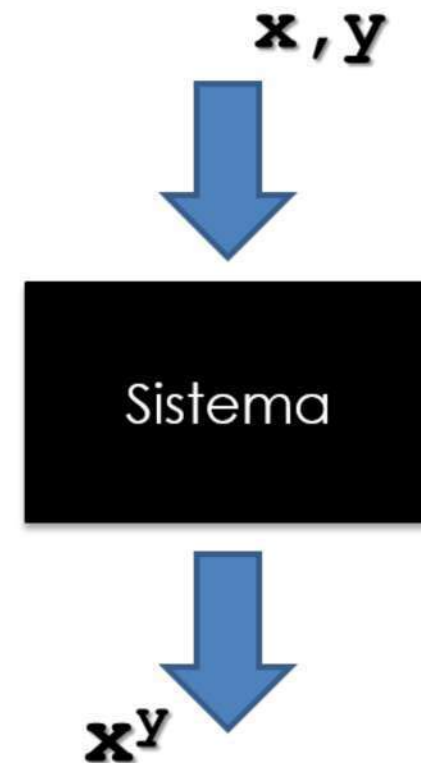
¿Cómo se hace testing?



Criterios de **caja negra** o funcionales

- Los datos de test se derivan a partir de la descripción del programa sin conocer su implementación.

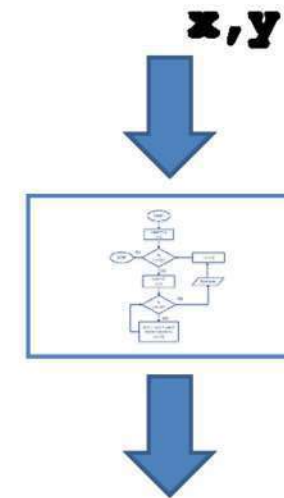
```
problema fastexp( $x : \mathbb{Z}, y : \mathbb{Z}$ ) :  $\mathbb{Z}$ {  
  requiere:  $\{0 \leq x \wedge 0 \leq y\}$   
  asegura:  $\{res = x^y\}$   
}
```



Criterios de **caja blanca** o estructurales

- Los datos de test se derivan a partir de la estructura interna del programa.

```
def fastexp(x: int, y: int) -> int:  
    z: int = 1  
    while(y != 0):  
        if(esImpar(y)):  
            z = z * x  
            y = y - 1  
  
            x = x * x  
            y = y / 2  
  
    return z
```



¿Qué pasa si y es potencia de 2?

¿Qué pasa si $y = 2^n - 1$?

Criterios de **caja blanca** o estructurales

Los criterios de caja blanca permiten identificar casos especiales según el flujo de control de la aplicación.

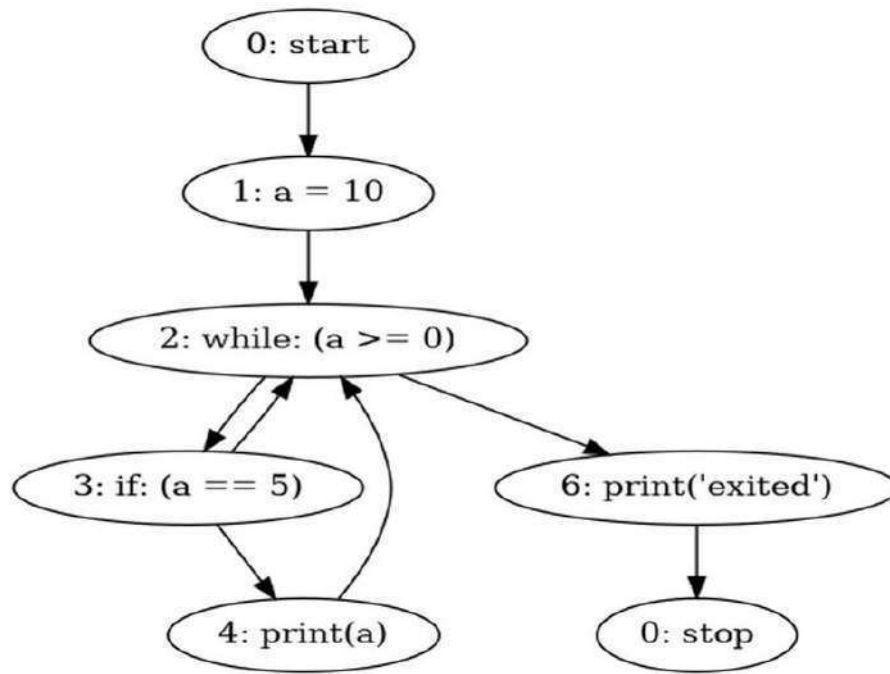
- ▶ Ver que sucede si entra o no en un IF
- ▶ Ver que sucede si entra o no a un ciclo
- ▶ Etc

Pero tiene una tremenda dificultad: determinar el resultado esperado de un programa sin una especificación no es para nada trivial.

Por este motivo, el test de caja blanca se suele utilizar como:

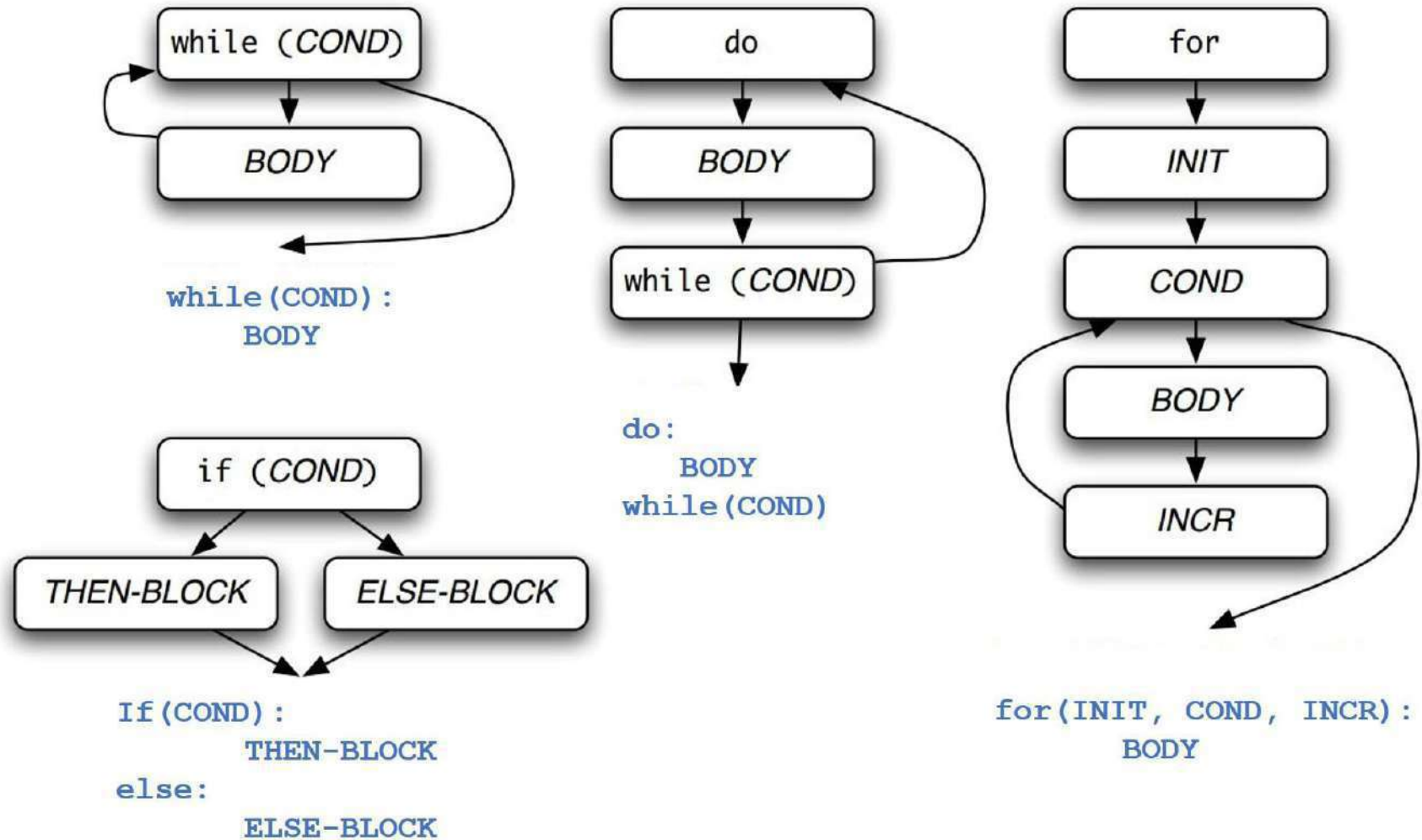
- ▶ **Complemento al Test de Caja Negra:** permite encontrar más casos o definir casos más específicos
- ▶ Como **criterio de adecuación** del Test de Caja Negra: brinda herramientas que nos ayuden a determinar cuán bueno o confiable resultaron ser los test suites definidos.
 - ▶ En este contexto hablaremos de **Criterios de Cubrimiento**

Control-Flow Graph



- ▶ El control flow graph (CFG) de un programa es sólo una **representación gráfica del programa**.
- ▶ El CFG es independiente de las entradas (su definición es estática)
- ▶ Se usa (entre otras cosas) para definir criterios de adecuación para test suites.
- ▶ Cuanto más *partes* son ejercitadas (cubiertas), mayores las chances de un test de descubrir una falla
- ▶ *partes* pueden ser: nodos, arcos, caminos, decisiones...

Control Flow Patterns

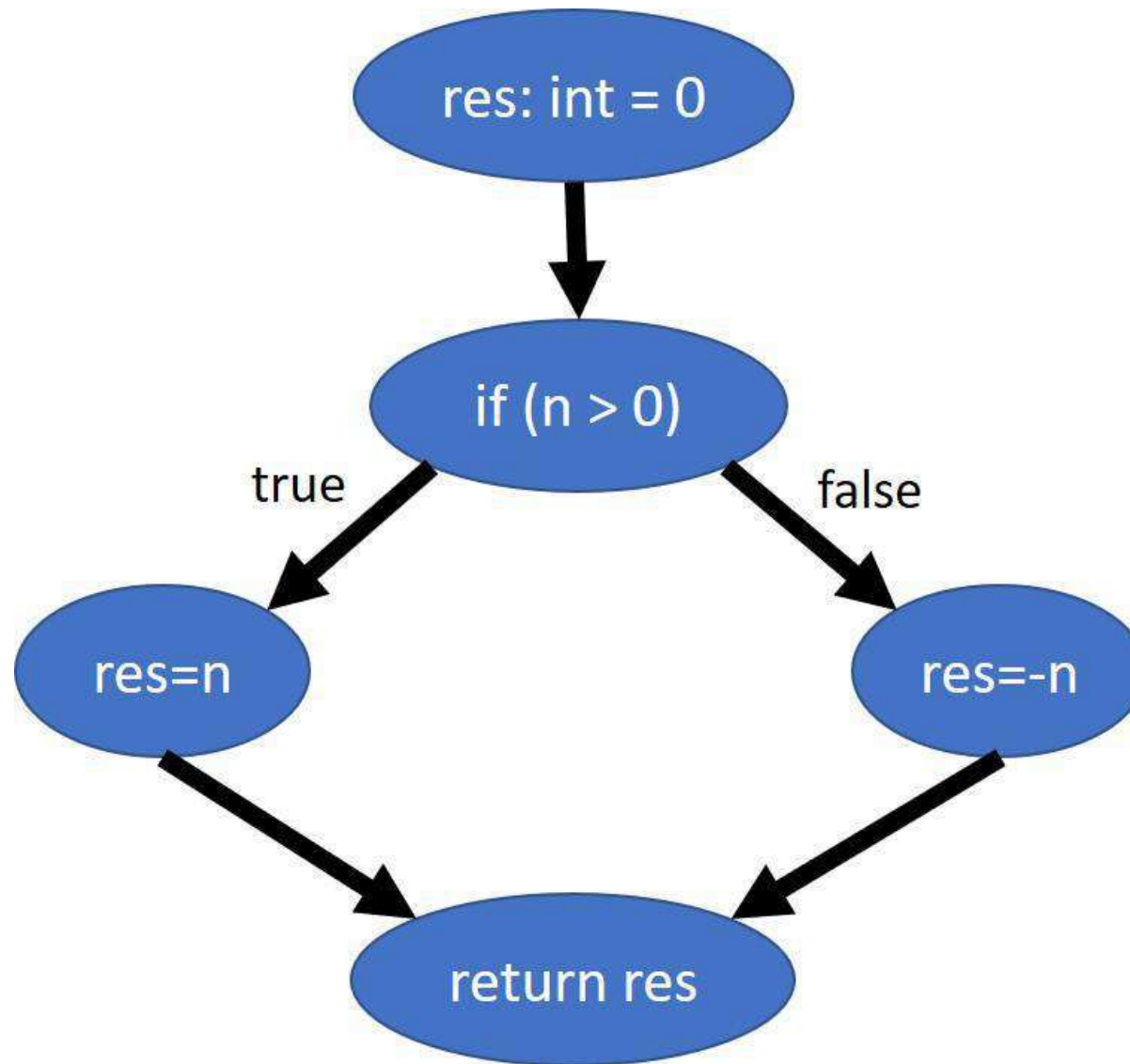


Ejemplo #1

```
problema valorAbsoluto(in  $x : \mathbb{Z}$ ) :  $\mathbb{Z}$ {  
  Pre {True}  
  Post {res =  $\|x\|$ }  
}
```

```
def valorAbsoluto(n: int) -> int:  
  res: int = 0  
  
  if( n > 0 ):  
    res = n  
  else:  
    res = -n  
  
  return res
```


CFG de valorAbsoluto

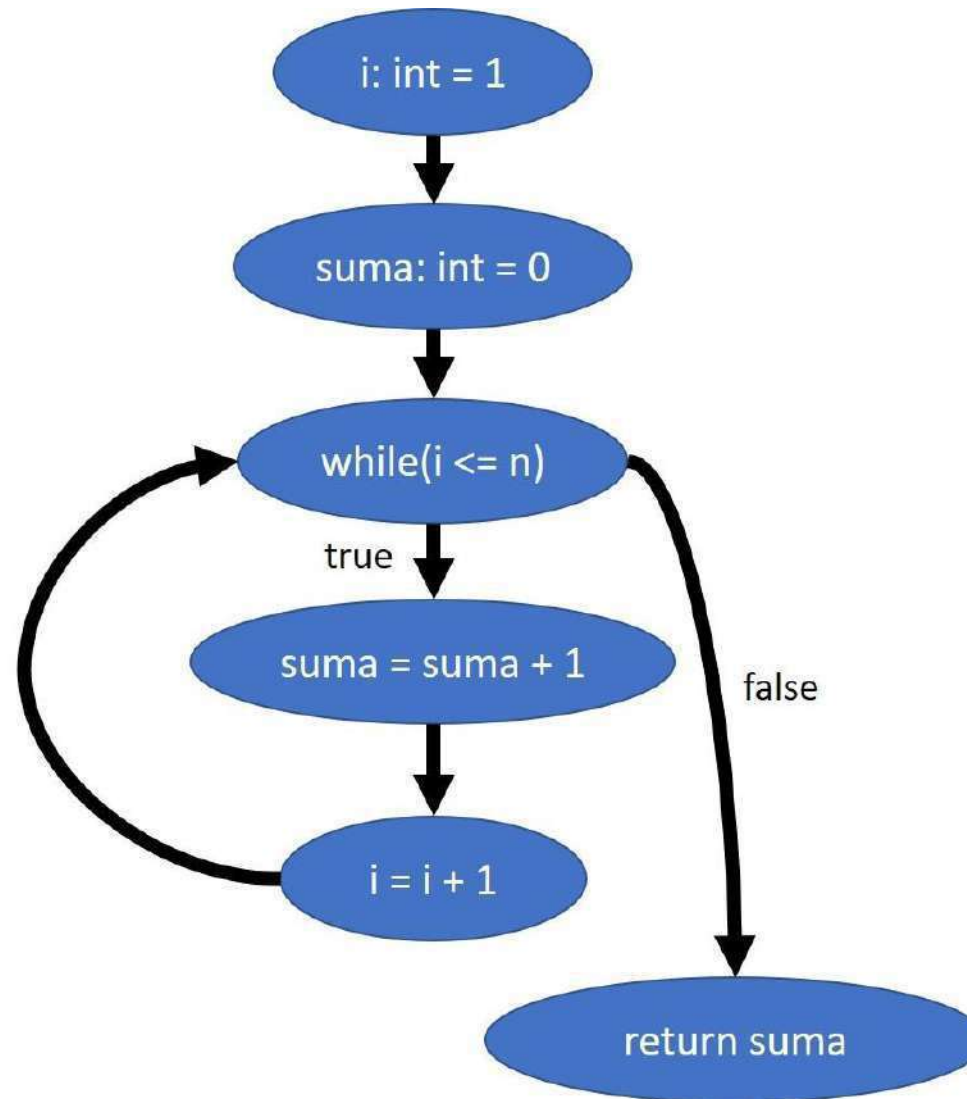


Ejemplo #2

problema *sumar*(in $n : \mathbb{Z}$) : \mathbb{Z} {
 Pre { $n \geq 0$ }
 Post { $res = \sum_{i=1}^n i$ }
}

```
def sumar(n: int) -> int:  
  i:int = 1  
  suma:int = 0  
  
  while( i <= n ):  
    suma = suma + i  
    i = i + 1  
  
  return suma
```

CFG de sumar

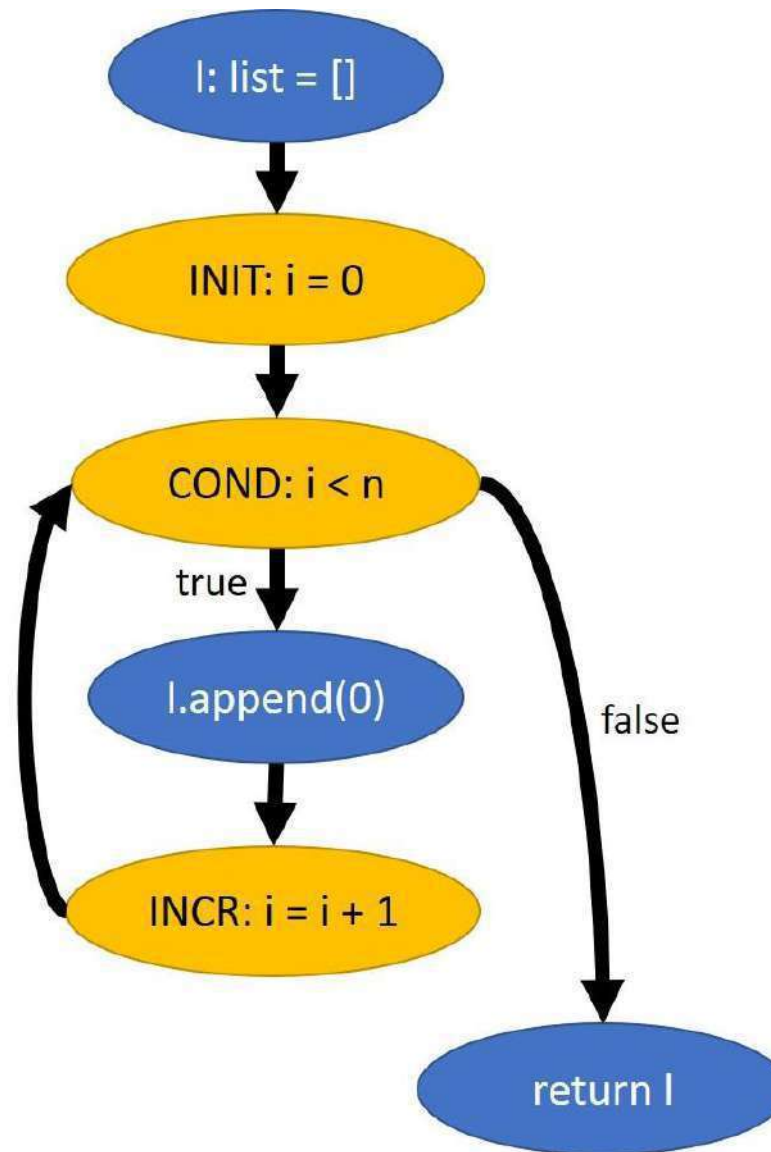


Ejemplo #3

```
problema crearListaN(in  $n : \mathbb{Z}$ ) : seq $\langle \mathbb{Z} \rangle$ {  
    Pre { $n \geq 0$ }  
    Post { $|res| = n \wedge \#apariciones(res, 0) = n$ }  
}
```

```
def crearListaN(int n) -> list:  
    l: list = []  
  
    for i in range(0, n, 1):  
        l.append(0)  
  
    return l
```

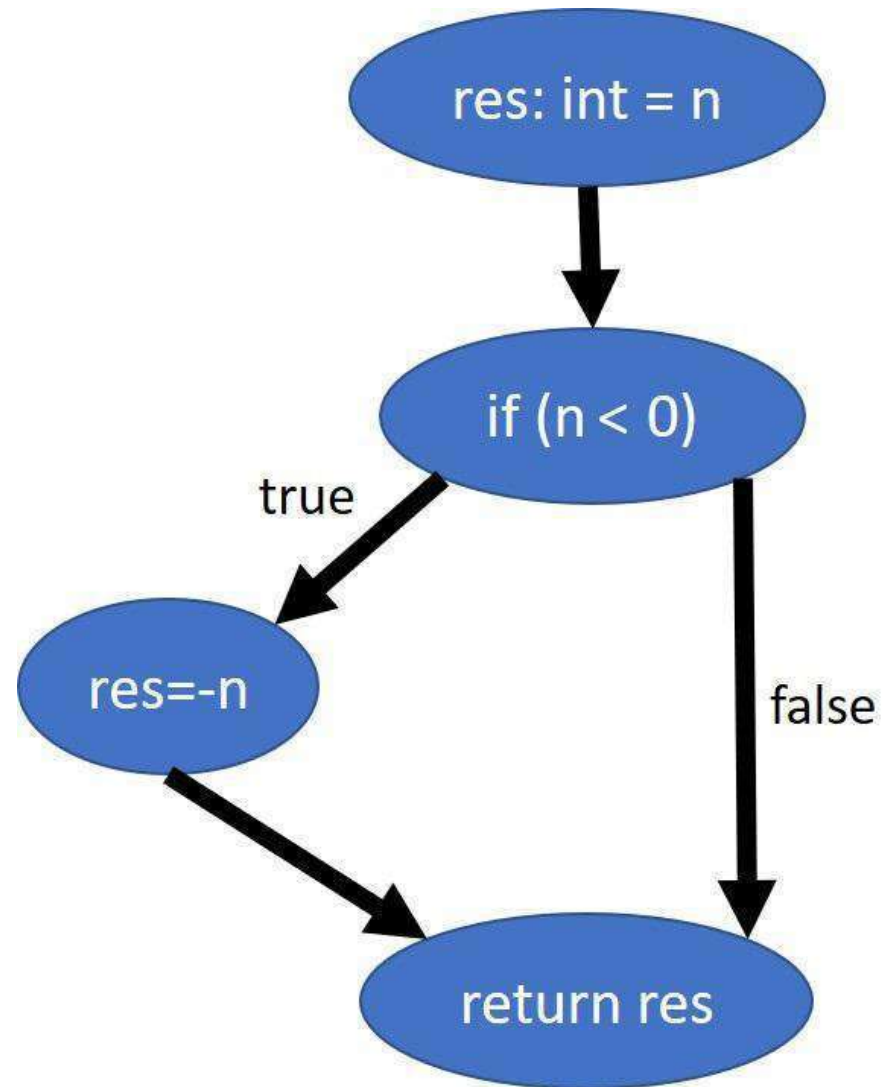
CFG de crearListaN



Ejemplo #4

```
def valorAbsoluto(n: int) -> int:  
    res: int = n  
  
    if( n < 0 ):  
        res = -n  
  
    return res
```

CFG de valorAbsoluto



Criterios de Adecuación

- ▶ ¿Cómo sabemos que un *test suite* es *suficientemente bueno*?
- ▶ Un criterio de adecuación de test es un predicado que toma un valor de verdad para una tupla $\langle \textit{programa}, \textit{test suite} \rangle$
- ▶ Usualmente expresado en forma de una regla del estilo:
todas las sentencias deben ser ejecutadas

Cubrimiento de Sentencias

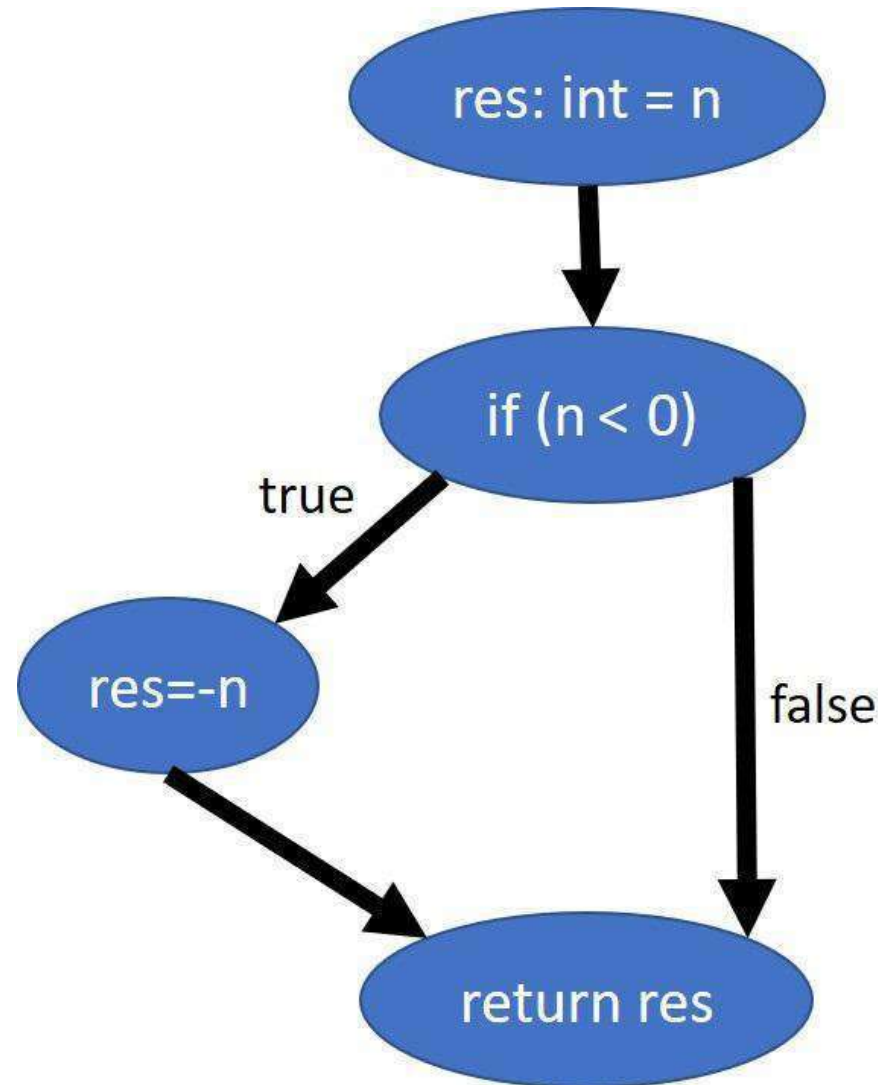
- ▶ Criterio de Adecuación: cada nodo (sentencia) en el CFG debe ser ejecutado al menos una vez por algún test case.
- ▶ Idea: un defecto en un sentencia sólo puede ser revelado ejecutando el defecto.
- ▶ Cobertura:
$$\frac{\text{cantidad nodos ejercitados}}{\text{cantidad nodos}}$$

Cubrimiento de Arcos

- ▶ Criterio de Adecuación: todo arco en el CFG debe ser ejecutado al menos una vez por algún test case.
- ▶ Si recorremos todos los arcos, entonces recorremos todos los nodos. Por lo tanto, el cubrimiento de arcos incluye al cubrimiento de sentencias.
- ▶ Cobertura:
$$\frac{\text{cantidad arcos ejercitados}}{\text{cantidad arcos}}$$
- ▶ El cubrimiento de sentencias (nodos) no incluye al cubrimiento de arcos. ¿Por qué?

Cubrimiento de Nodos no incluye cubrimiento de Arcos

Sea el siguiente CFG:



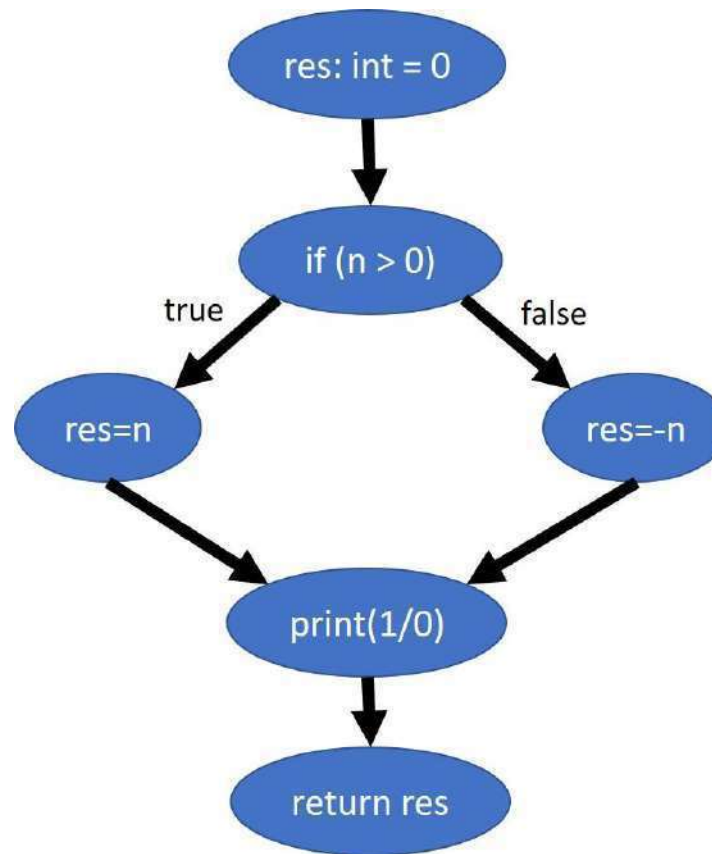
En este ejemplo, puedo construir un test suite que cubra todos los nodos pero que no cubra todos los arcos.

Cubrimiento de Decisiones (o Branches)

- ▶ Criterio de Adecuación: cada decisión (arco True o arco False) en el CFG debe ser ejecutado.
- ▶ Por cada arco **True** o arco **False**, debe haber al menos un test case que lo ejercite.
- ▶ Cobertura:
$$\frac{\text{cantidad decisiones ejercitadas}}{\text{cantidad decisiones}}$$
- ▶ El cubrimiento de decisiones **no implica** el cubrimiento de los arcos del CFG. ¿Por qué?

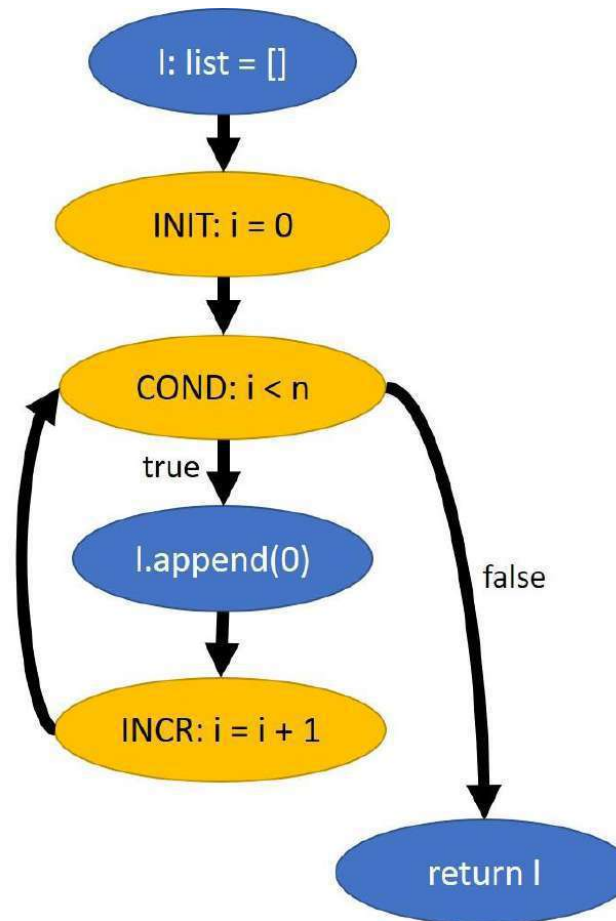
Cubrimiento de Branches no incluye cubrimiento de Arcos

Sea el siguiente CFG:



En este ejemplo, puedo construir un test suite que cubra todos los branches pero que no cubra todos los arcos.

CFG de crearListaN



- ¿Cuántos nodos (sentencias) hay? 6
- ¿Cuántos arcos (flechas) hay? 6
- ¿Cuántas decisiones (arcos True y arcos False) hay? 2

Cubrimiento de Condiciones Básicas

- ▶ Una condición básica es una fórmula atómica (i.e. no divisible) que componen una decisión.
 - ▶ Ejemplo: `(digitHigh==1 || digitLow==-1) && len>0`
 - ▶ Condiciones básicas:
 - ▶ `digitHigh==1`
 - ▶ `digitLow==-1`
 - ▶ `len>0`
 - ▶ No es condición básica: `(digitHigh==1 || digitLow==-1)`
- ▶ Criterio de Adecuación: cada condición básica de cada decisión en el CFG debe ser evaluada a verdadero y a falso al menos una vez
- ▶ Cobertura:

$$\frac{\text{cantidad de valores evaluados en cada condicion}}{2 * \text{cantidad condiciones basicas}}$$

Cubrimiento de Condiciones Básicas

- Sea una única decisión:

`(digitHigh==1 || digitLow==-1) && len>0`

- Y el siguiente test case:

Entrada	digitHigh==1?	digitLow==-1?	len>0?
digitHigh=1, digitLow=0 len=1,	True	False	True

- ¿Cuál es el cubrimiento de condiciones básicas?

$$C_{\text{cond.básicas}} = \frac{3}{2 * 3} = \frac{3}{6} = 50\%$$

Cubrimiento de Condiciones Básicas

- Sea una única decisión:

`(digitHigh==1 || digitLow==-1) && len>0`

- Y el siguiente test case:

Entrada	digitHigh==1?	digitLow==-1?	len>0?
digitHigh=1, digitLow=0 len=1,	True	False	True
digitHigh=0, digitLow=-1 len=0,	False	True	False

- ¿Cuál es el cubrimiento de condiciones básicas?

$$C_{\text{cond.básicas}} = \frac{6}{2 * 3} = \frac{6}{6} = 100\%$$

Cubrimiento de Branches y Condiciones Básicas

- ▶ **Observación** Branch coverage no implica cubrimiento de Condiciones Básicas
 - ▶ Ejemplo: **if(a && b)**
 - ▶ Un test suite que ejercita solo $a = true, b = true$ y $a = false, b = true$ logra cubrir ambos branches de **if(a && b)**
 - ▶ **Pero:** no alcanza cubrimiento de decisiones básica ya que falta $b = false$
- ▶ El criterio de cubrimiento de Branches y condiciones básicas necesita 100 % de cobertura de branches y 100 % de cobertura de condiciones básicas
- ▶ Para ser aprobado, todo software que controla un avión necesita ser testeado con cubrimiento de branches y condiciones básicas (RTCA/DO-178B y EUROCAE ED-12B).

Cubrimiento de Caminos

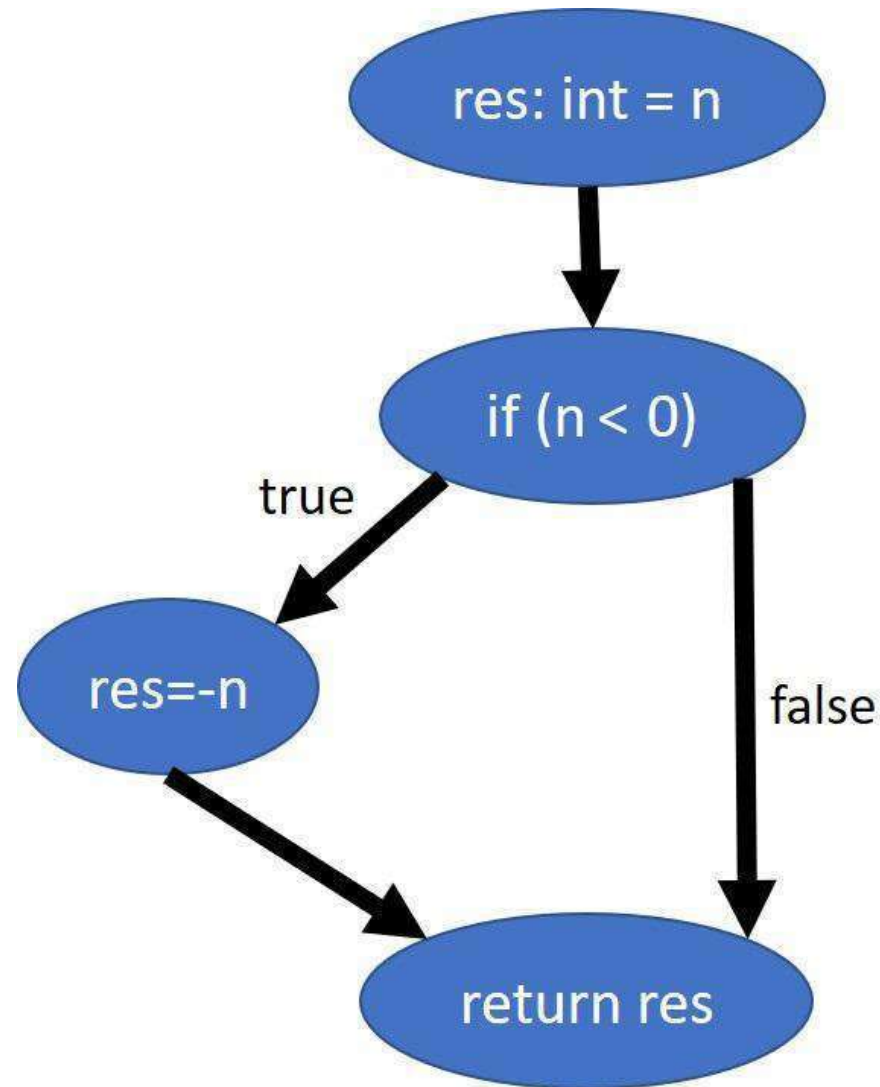
- ▶ Criterio de Adecuación: cada camino en el CFG debe ser transitado por al menos un test case.

- ▶ Cobertura:

$$\frac{\text{cantidad caminos transitados}}{\text{cantidad total de caminos}}$$

Caminos para el CFG de valorAbsoluto

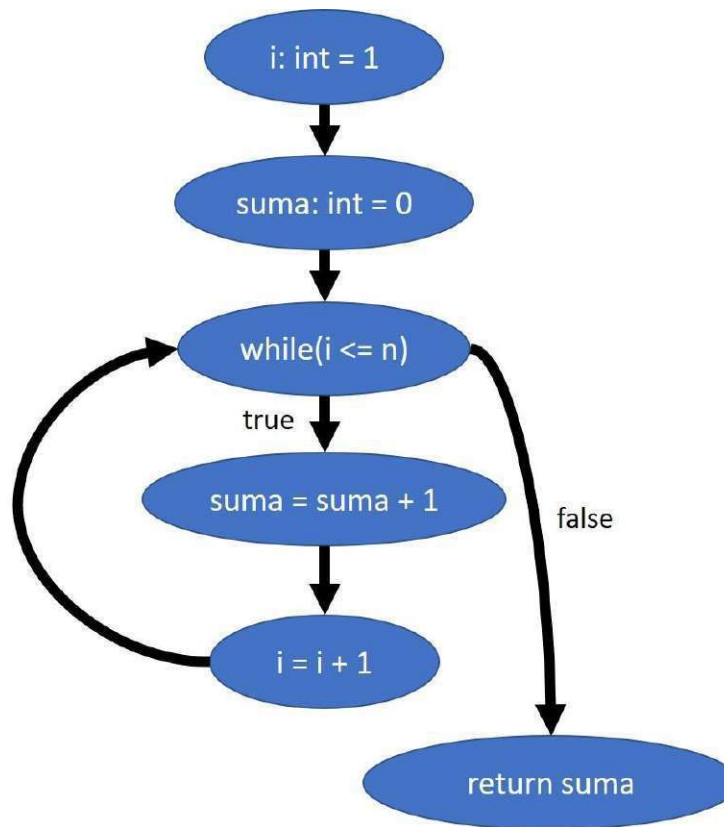
Sea el siguiente CFG:



¿Cuántos caminos hay en este CFG? 2

Caminos para el CFG de sumar

Sea el siguiente CFG:



¿Cuántos caminos hay en este CFG? La cantidad de caminos no está acotada (∞)

Recap: Criterios de Adecuación Estructurales

- ▶ En todos estos criterios se usa el CFG para obtener una métrica del test suite
- ▶ **Sentencias:** cubrir todos los nodos del CFG.
- ▶ **Arcos:** cubrir todos los arcos del CFG.
- ▶ **Decisiones (Branches):** Por cada if, while, for, etc., la guarda fue evaluada a verdadero y a falso.
- ▶ **Condiciones Básicas:** Por cada componente básico de una guarda, este fue evaluado a verdadero y a falso.
- ▶ **Caminos:** cubrir todos los caminos del CFG. Como no está acotado o es muy grande, se usa muy poco en la práctica.

esPrimo()

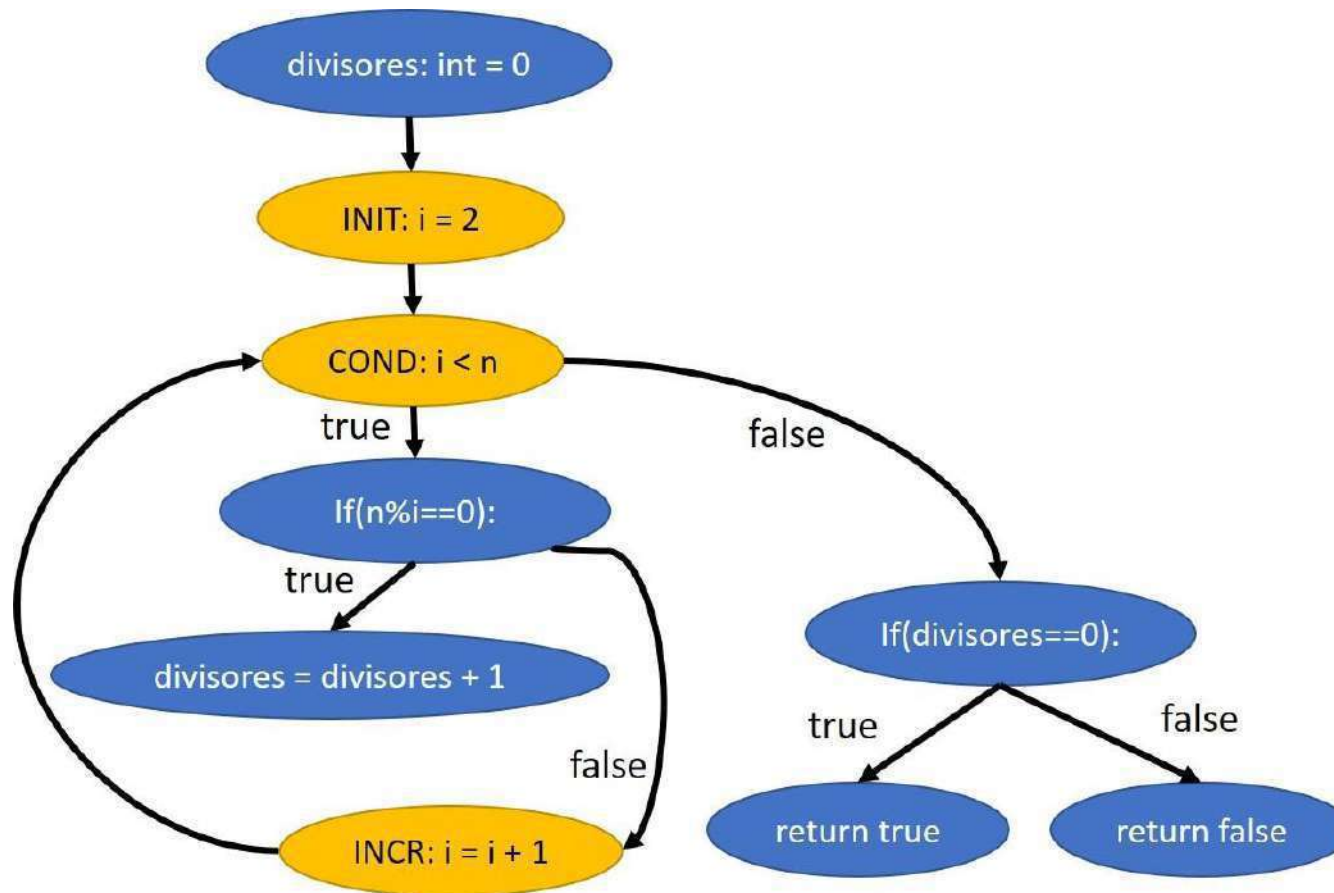
Sea la siguiente implementación que decide si un número $n > 1$ es primo:

```
def esPrimo(n : int) → bool:
    divisores: int = 0
    for i in range(2, n, 1):
        if (n \% i == 0 ):
            divisores = divisores + 1

    if (divisores == 0):
        return true
    else:
        return false
```

Graficar el CFG de la función esPrimo().

esPrimo()



Cubrimientos

Sea el siguiente test suite para `esPrimo()`:

- ▶ Test Case #1: `valorPar`
 - ▶ Entrada: $n = 2$
 - ▶ Salida esperada: *result = true*
- ▶ Test Case #2: `valorImpar`
 - ▶ Entrada: $n = 3$
 - ▶ Salida esperada: *result = true*
- ▶ ¿Cuál es el cubrimiento de sentencias (nodos) del test suite?

$$Cov_{sentencias} = \frac{7}{9} \sim 77\%$$

- ▶ ¿Cuál es el cubrimiento de decisiones (brances) del test suite?

$$Cov_{branches} = \frac{4}{6} \sim 66\%$$

Cubrimientos

Sea el siguiente test suite para `esPrimo()`:

- ▶ Test Case #1: `valorPrimo`
 - ▶ Entrada: $n = 3$
 - ▶ Salida esperada: *result = true*
- ▶ Test Case #2: `valorNoPrimo`
 - ▶ Entrada: $n = 4$
 - ▶ Salida esperada: *result = false*
- ▶ ¿Cuál es el cubrimiento de sentencias (nodos) del test suite?

$$Cov_{sentencias} = \frac{9}{9} = 100\%$$

- ▶ ¿Cuál es el cubrimiento de decisiones (brances) del test suite?

$$Cov_{branches} = \frac{6}{6} = 100\%$$

Discusión

- ▶ ¿Puede haber partes (nodos, arcos, branches) del programa que no sean alcanzables con **ninguna** entrada válida (i.e. que cumplan la precondition)?
- ▶ ¿Qué pasa en esos casos con las métricas de cubrimiento?
- ▶ Existen esos casos (por ejemplo: código defensivo o código que sólo se activa ante la presencia de un estado inválido)
- ▶ El 100 % de cubrimiento suele ser no factible, por eso es una medida para analizar con cuidado y estimar en función al proyecto (ejemplo: 70 %, 80 %, etc.)