

# Tecnologías de la Información II

## Clase 04: Ciclos

Daniela Opitz, Diego Caro, Ismael Botti  
[dopitz@udd.cl](mailto:dopitz@udd.cl)



Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

Disponible en <https://introcs.cs.princeton.edu/python>

# Clase de Hoy

- Ciclo **for**
- Comparación **while** vs **for**
- Diagramas lógicos

# Ciclo **for**

- **for**: Permite repetir un conjunto de instrucciones un numero determinado de veces. La secuencia de instrucciones se recorre en orden.
- Sintaxis:  
**for** <variable> in <elemento iterable>:  
    <instrucciones>
- Ejemplo: Imprime el texto “Hola número n veces seguido del valor de n donde n va desde 0 a 3”.

```
for i in range(4):  
    print('Hola número', i)
```

Variable usada  
para recorrer la  
secuencia

Secuencia de  
enteros de 0  
hasta n - 1

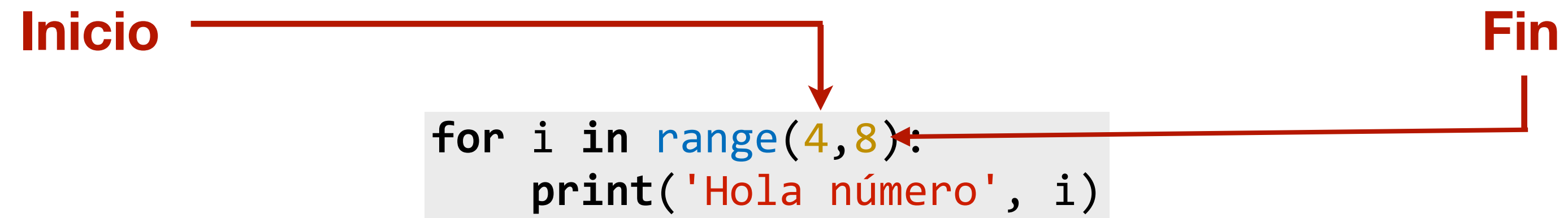
```
$ python3 holas.py  
Hola número 0  
Hola número 1  
Hola número 2  
Hola número 3
```

# Ciclo **for**

- Imprime el texto “Hola número n veces seguido del valor de n donde n va desde 4 a 7”.

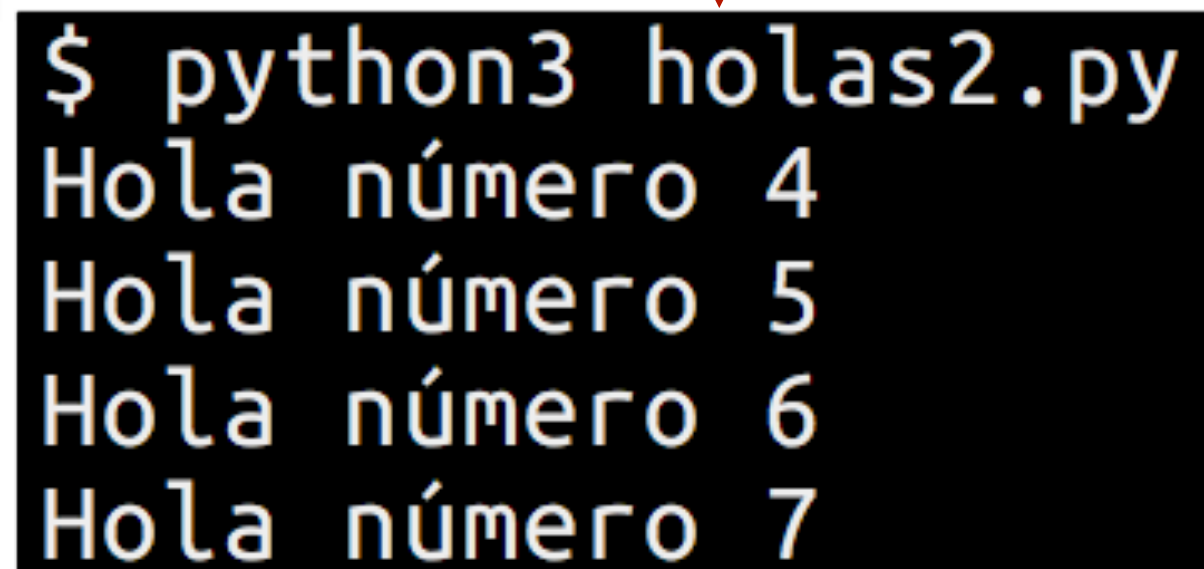
**Inicio** ————— **Fin**

```
for i in range(4,8):  
    print('Hola número', i)
```



**Salida**

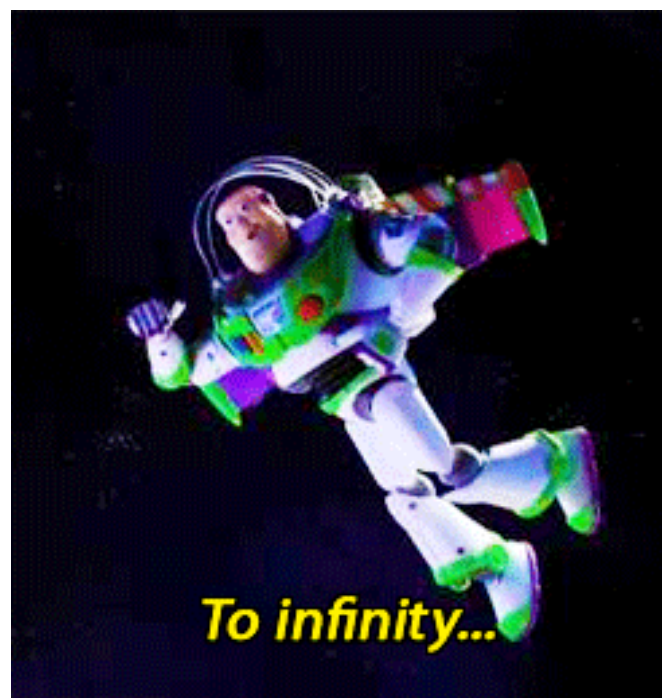
```
$ python3 holas2.py  
Hola número 4  
Hola número 5  
Hola número 6  
Hola número 7
```



# while vs for

Ejemplo de un ciclo **while** que **nunca termina**. La condición de detención siempre es **True**!

```
x = 1
while True:
    print("Al infinito y más allá! Ya vamos en {:d}!".format(x))
    x += 1
```



```
1. bash
Al infinito y más allá! Ya vamos en 93523!
Al infinito y más allá! Ya vamos en 93524!
Al infinito y más allá! Ya vamos en 93525!
Al infinito y más allá! Ya vamos en 93526!
Al infinito y más allá! Ya vamos en 93527!
Al infinito y más allá! Ya vamos en 93528!
Al infinito y más allá! Ya vamos en 93529!
Al infinito y más allá! Ya vamos en 93530!
Al infinito y más allá! Ya vamos en 93531!
Al infinito y más allá! Ya vamos en 93532!
Al infinito y más allá! Ya vamos en 93533!
Al infinito y más allá! Ya vamos en 93534!
Al infinito y más allá! Ya vamos en 93535!
Al infinito y más allá! Ya vamos en 93536!
Al infinito y más allá! Ya vamos en 93537!
Al infinito y más allá! Ya vamos en 93538!
Al infinito y más allá! Ya vamos en 93539!
Al infinito y más allá! Ya vamos en 93540!
Al infinito y más allá! Ya vamos en 93541!
Al infinito y más allá! Ya vamos en 93542!
Al infinito y más allá! Ya vamos en 93543!
```



# Numeros Pares e Impares

- Números pares: números que son divisibles en 2

$$i \% 2 == 0$$

- Números impares: números que no son divisibles en 2

$$i \% 2 != 0$$

$$i \% 1 == 1$$



# while vs for

- Ejemplo: Imprima todos los números impares menores que n mayores o iguales a cero.

## Solución 1

```
1 n = int(input('ingrese n: '))
2 if n <= 0:
3     print('Debe ingresar un número mayor a cero')
4 i = 0
5 while i < n:
6     if i % 2 == 1:
7         print(i)
8     i = i+1
```

## Solución 2

```
1 n = int(input('ingrese n: '))
2 for i in range(n):
3     if i % 2 == 1:
4         print(i)
```

## Solución 3

```
1 n = int(input('ingrese n: '))
2 for i in range(1, n, 2):
3     print(i)
```

# Diagramas Lógicos (Ismael Botti)

Los diagramas lógicos incluyen:

- condiciones
- estructuras iterativas

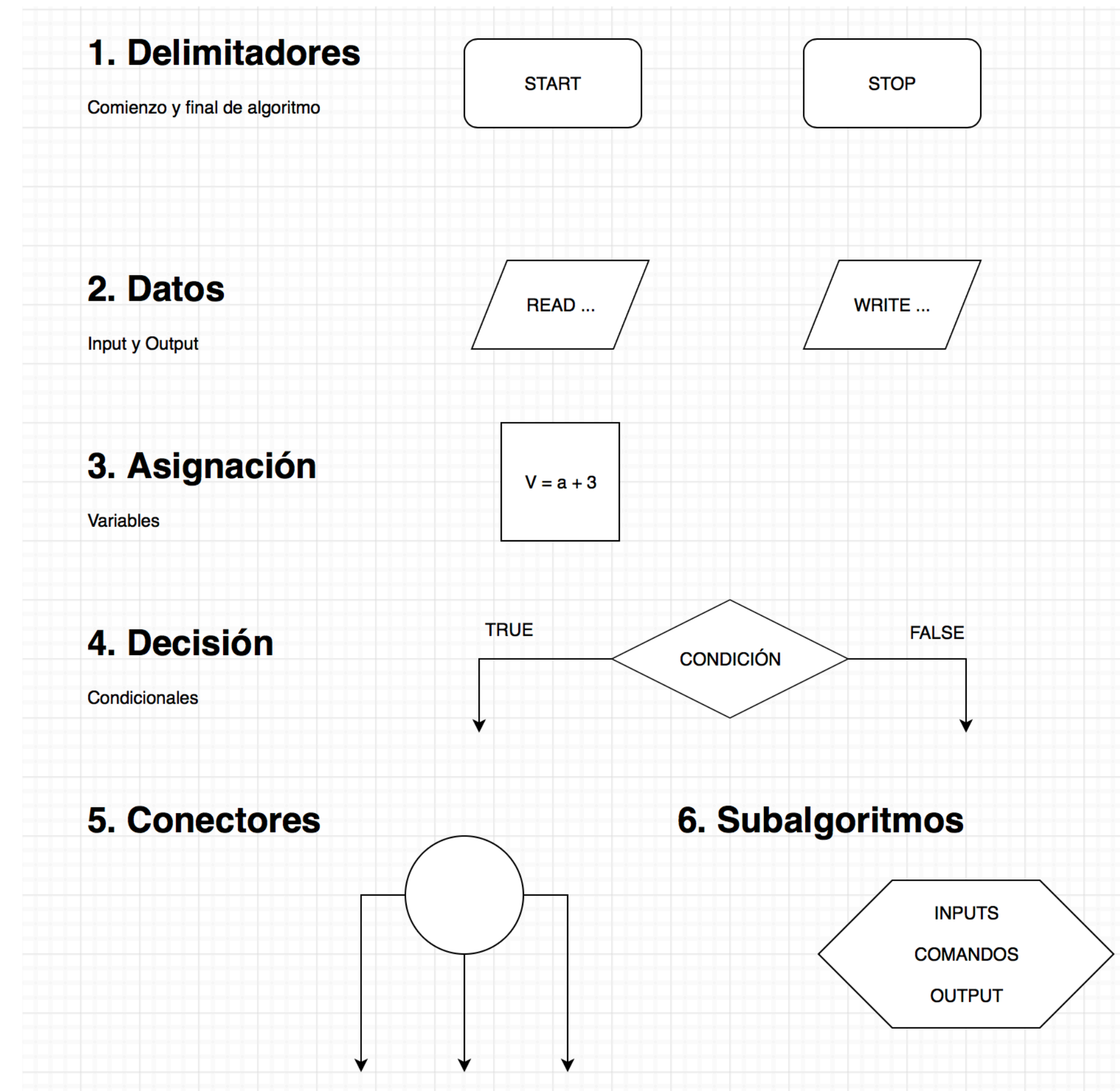
Es importante darse cuenta de que la programación no es para nada algo especial, es una forma alternativa de resolver problemas, como los que tenemos en los cursos matemáticos.



# Reglas para un Diagrama Lógico (Ismael Botti)

- Cada esquema debe comenzar con un COMIENZO (**START**) y terminar en un FINAL (**STOP**). Sólo debe haber un bloque START y un STOP en un esquema.
- No se debe improvisar, sólo usar bloques. Es por eso que hay que pensar muy bien cómo conectar los bloques.
- Flechas deben tocar los bloques de inicio y final.

# Diagramas Lógicos (Ismael Botti)

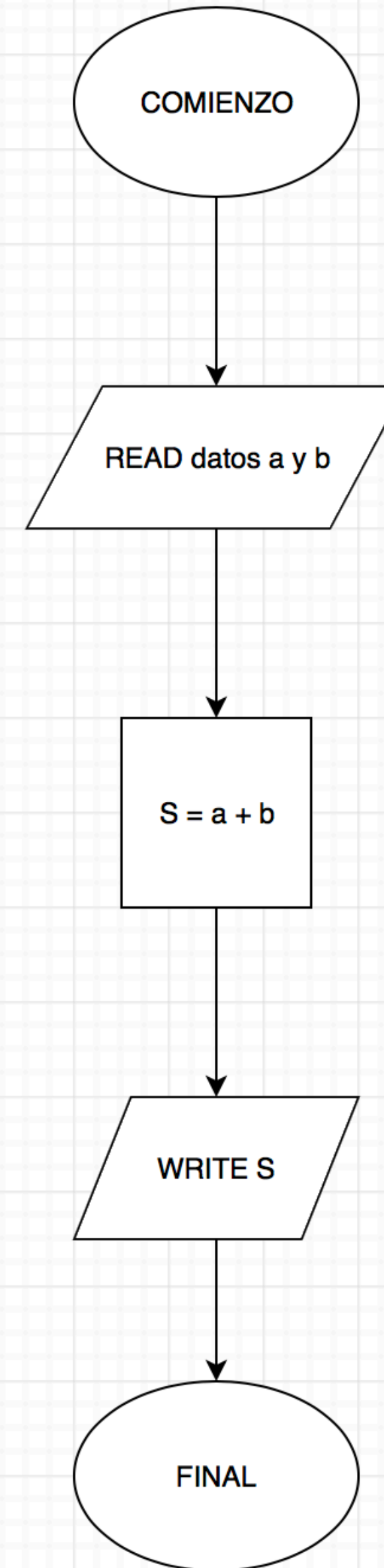


**Elementos de un Diagrama de Flujo**

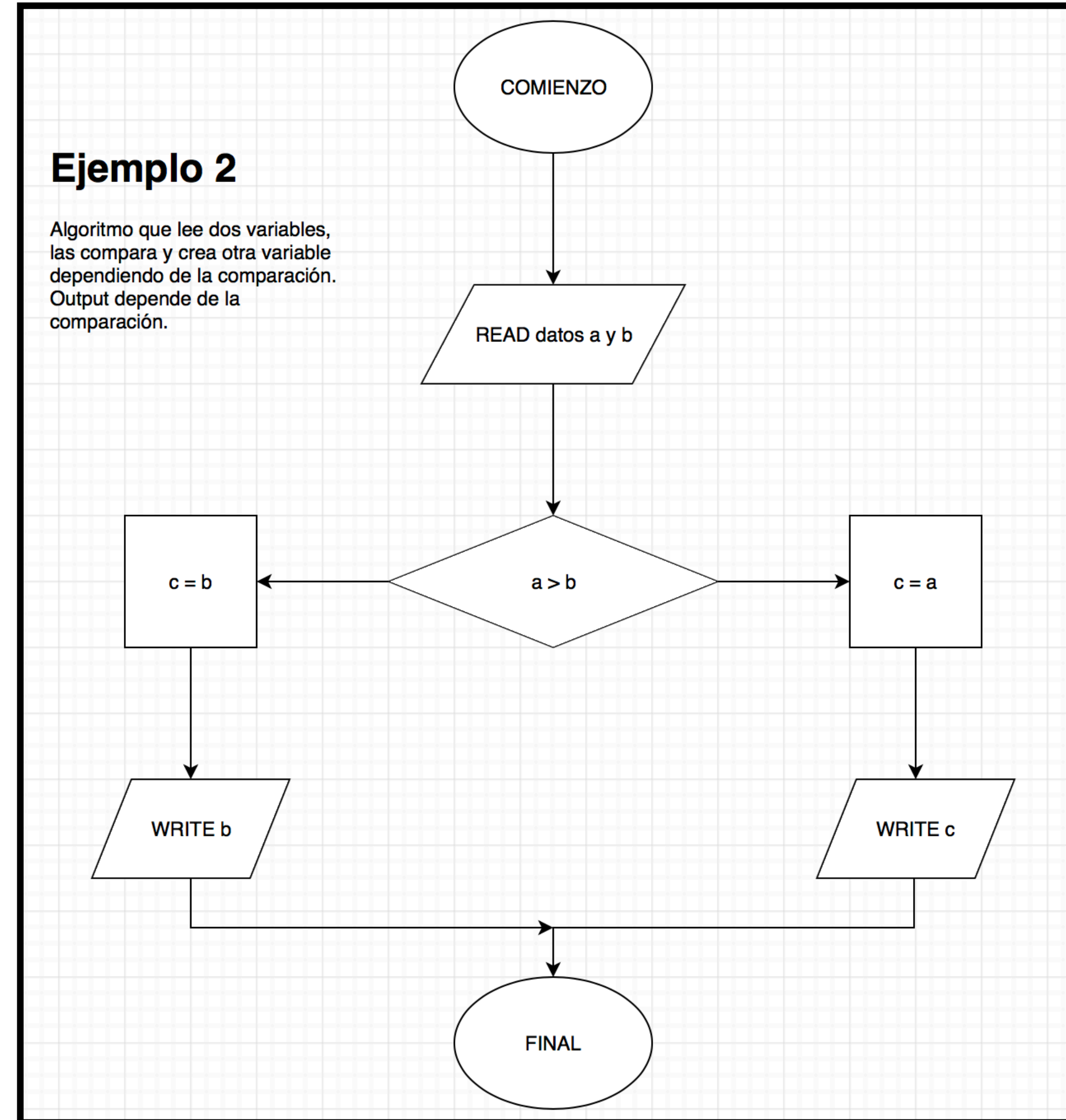
# Diagramas Lógicos (Ismael Botti)

## Ejemplo 1

Algoritmo simple y lineal, donde se leen dos variables, se crea una tercera que es la que se entrega como output.



# Diagramas Lógicos (Ismael Botti)



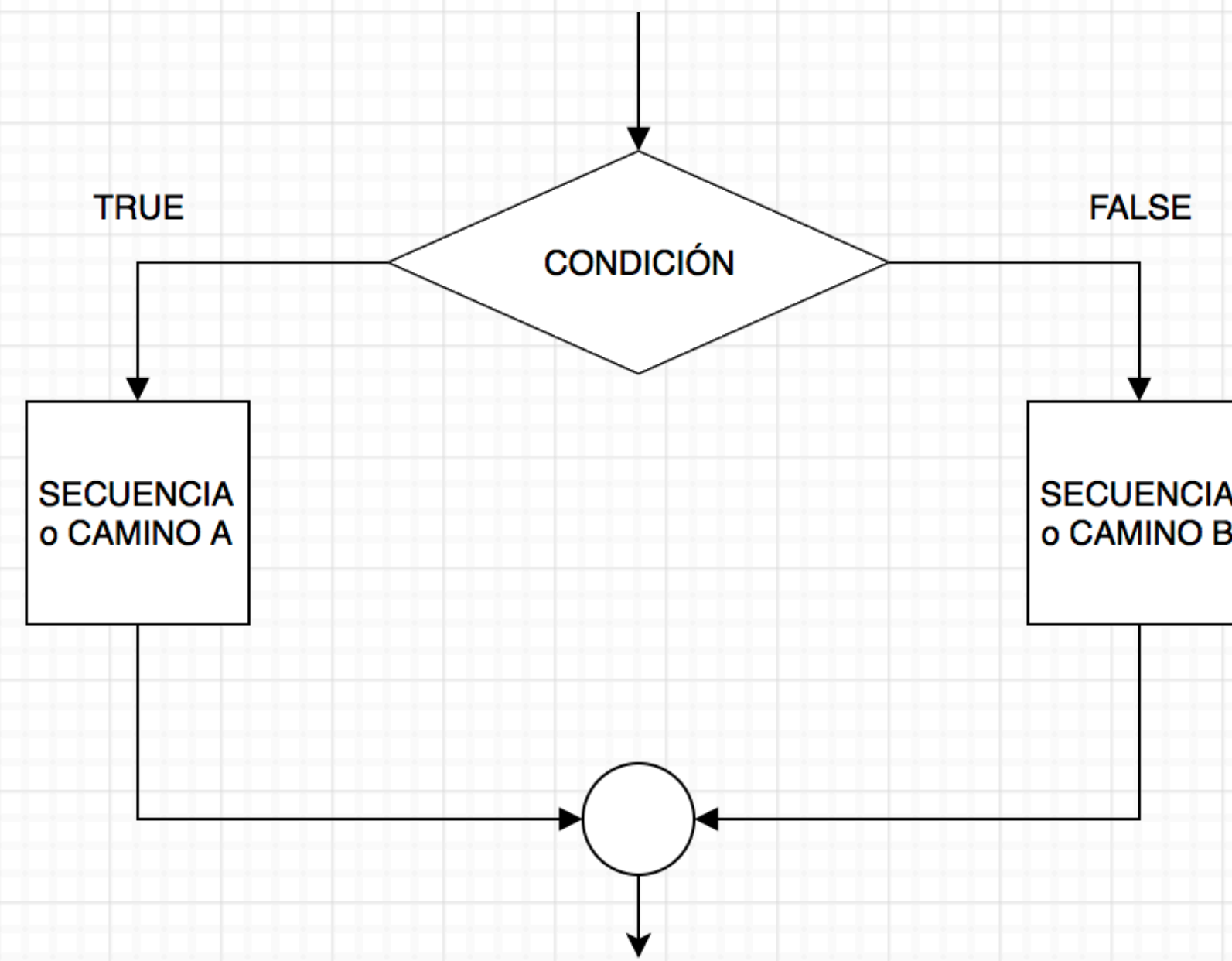
# Condiciones en Diagramas Lógicos

- Se usa cuando tenemos dos alternativas y sólo podemos escoger una.
- Es importante tener un criterio (condición matemática).
- Una vez que se haya optado por una opción o la otra, el algoritmo seguirá por un camino donde:
  - No puede volver atrás
  - No puede cambiar de alternativa
- Se pueden usar para:
  - Validar inputs
  - Validar outputs
  - Manejar excepciones



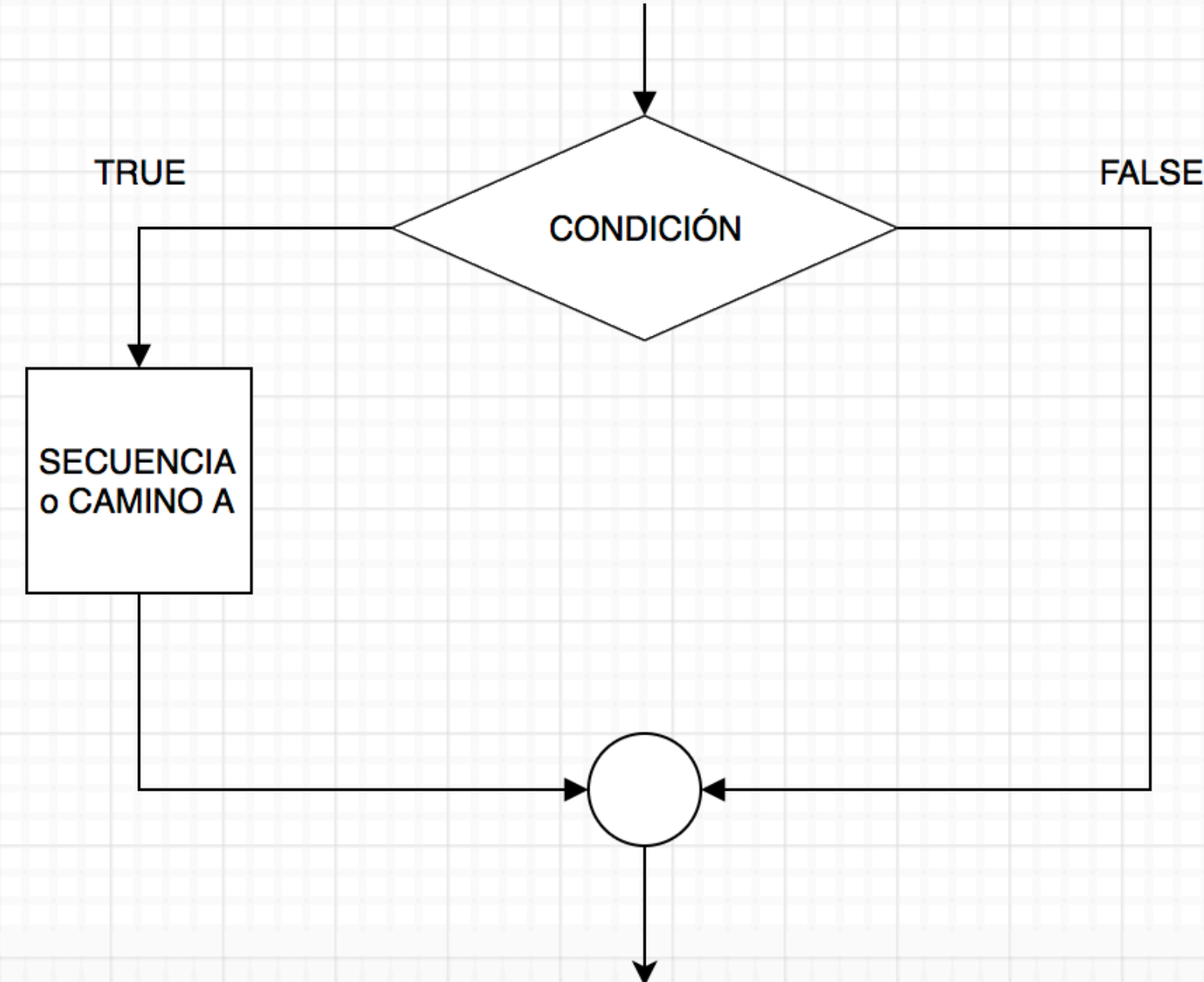
## Caso General

Al evaluar una condición se debe optar por un camino A o el B.



## Caso Particular

Uno de los caminos no tiene ninguna secuencia a ejecutar.



# Estructuras Iterativas

Se componen de:

- Un contador
- Una condición de salida
- Secuencia de comandos

## Importancia

Todas las partes son igualmente importantes. Si **NO** hay un:

- **Contador:** algoritmo nunca sale del loop (loop infinito)
- **Condición de salida:** algoritmo nunca sale del loop
- **Comandos:** el algoritmo no hace nada



# Tipos de Loops

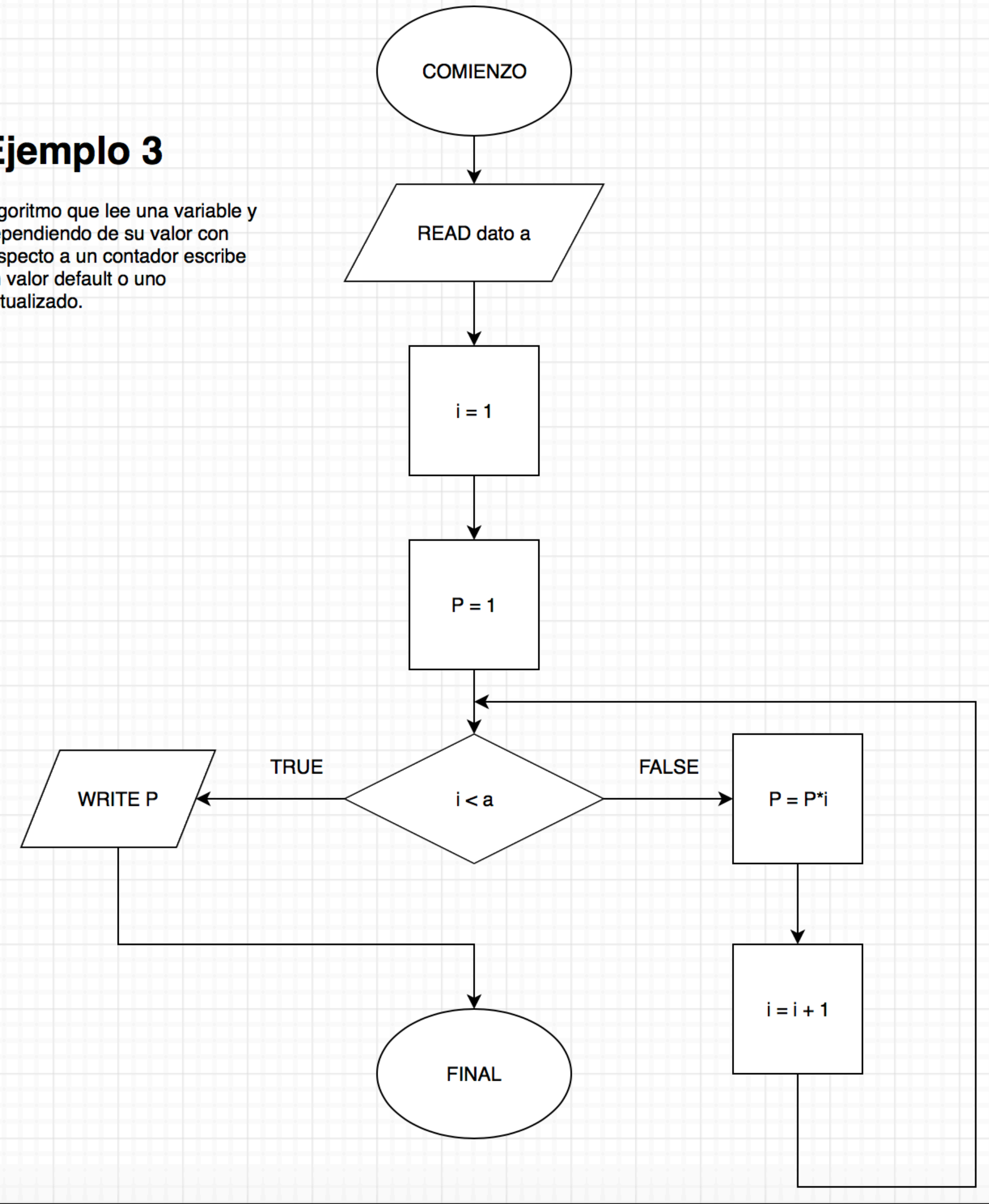
- Iteraciones que usan un test inicial
- Iteraciones que usan un test final
- Iteraciones que utilizan un contador

Primer y segundo tipo se diferencian en la posición de la  
condición de término del loop.

# Diagramas Lógicos (Ismael Botti)

## Ejemplo 3

Algoritmo que lee una variable y dependiendo de su valor con respecto a un contador escribe un valor default o uno actualizado.



# Resumen

## Conceptos

- **while**: ejecutar código mientras una condición se cumple
- **for**: ejecutar código al recorrer una secuencia. La secuencia se puede generar con la función `range(...)`

## Funciones

- **range(stop)**: secuencias de enteros hasta `stop-1`
- **range(start, stop[, step])**: secuencia de enteros desde `start` hasta `stop-1`, saltándose `step` pasoss



# Resumen

## ¿En dónde estamos?

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

[https://docs.python.org/3/reference/lexical\\_analysis.html](https://docs.python.org/3/reference/lexical_analysis.html)

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<https://docs.python.org/3/library/functions.html>