

# Taller de Programación

## 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

- Acumuladores y contadores
- Ciclo **for**
- Comparación **while** vs **for**
- Diagramas lógicos

# Acumuladores y Contadores

Dos de las utilidades más comunes en las iteraciones son la acumulación y el conteo de números.

**Ejemplo:** Sume los primeros **n** números y contar cuántos números hay entre 1 y n (trivial).

inicio variable **suma**  
y **contador**  
con valor 0

```
#Sumo numeros desde 1 a 3
#Cuento numeros desde 1 a 3
n = int(input("Ingrese un numero: "))
suma = 0 #acumulador
contador = 0 #contador

while contador <= n:
    suma = suma + contador
    contador = contador + 1

print(suma)
print(contador-1) #Si no cuento uno más
```

# Acumuladores y Contadores

```
#Sumo números desde 1 a 3
#Cuento números desde 1 a 3

n = int(input("Ingrese un numero: "))
suma = 0 #acumulador
contador = 0 #contador

while contador < n:
    suma += contador
    contador += 1

print(suma)
```

```
suma = suma + contador
contador = contador + 1
```

son equivalentes!

# Numeros Pares e Impares

- Números pares: números que son divisibles en 2  
 $i \% 2 == 0$
- Números impares: números que no son divisibles en 2  
 $i \% 2 != 0$  o bien  $i \% 2 == 1$



- **Ejemplo:** Imprimir y contar los número pares entre 1 y un numero n

contador

variable usada para  
recorrer la secuencia

```
numero = int(input("Ingrese un numero: "))
i=1
contador = 0

while i <= numero:
    if i % 2 == 0:
        print(i, 'es par')
        contador += 1
    i += 1
print('Numeros de pares entre 1 y 10:', contador)
```

Voy contando los pares



# 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”.

Variable usada  
para recorrer la  
secuencia

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

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)
```

`range(start, stop[, step])`

secuencia de enteros desde start  
hasta stop-1, saltándose step  
pasos

**Salida**

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

# while vs for

while	for
número <b>desconocido</b> de iteraciones	número <b>conocido</b> de iteraciones
<b>no siempre</b> puede ser sustituido por un ciclo for	<b>puede</b> ser sustituido por un ciclo while
necesita un contador que se inicie <b>antes</b> del loop y que se incremente <b>dentro</b> del loop	usa una variable (contador) para recorrer la secuencia



# 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

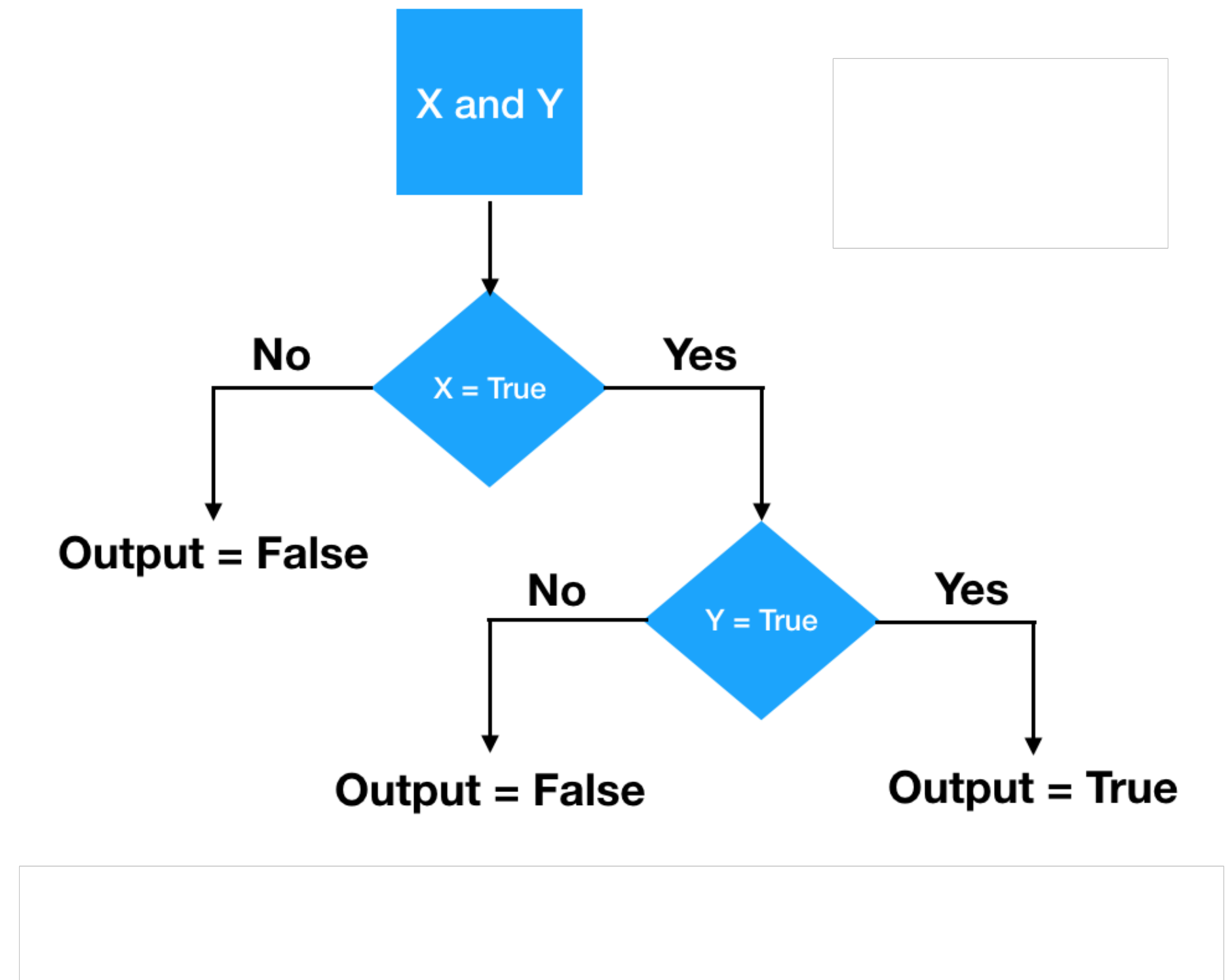
```
1 n = int(input('ingrese n: '))
2 for i in range(n):
3     if i % 2 == 1:
4         print(i)
```

## Solución 3

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

# Diagramas Lógicos

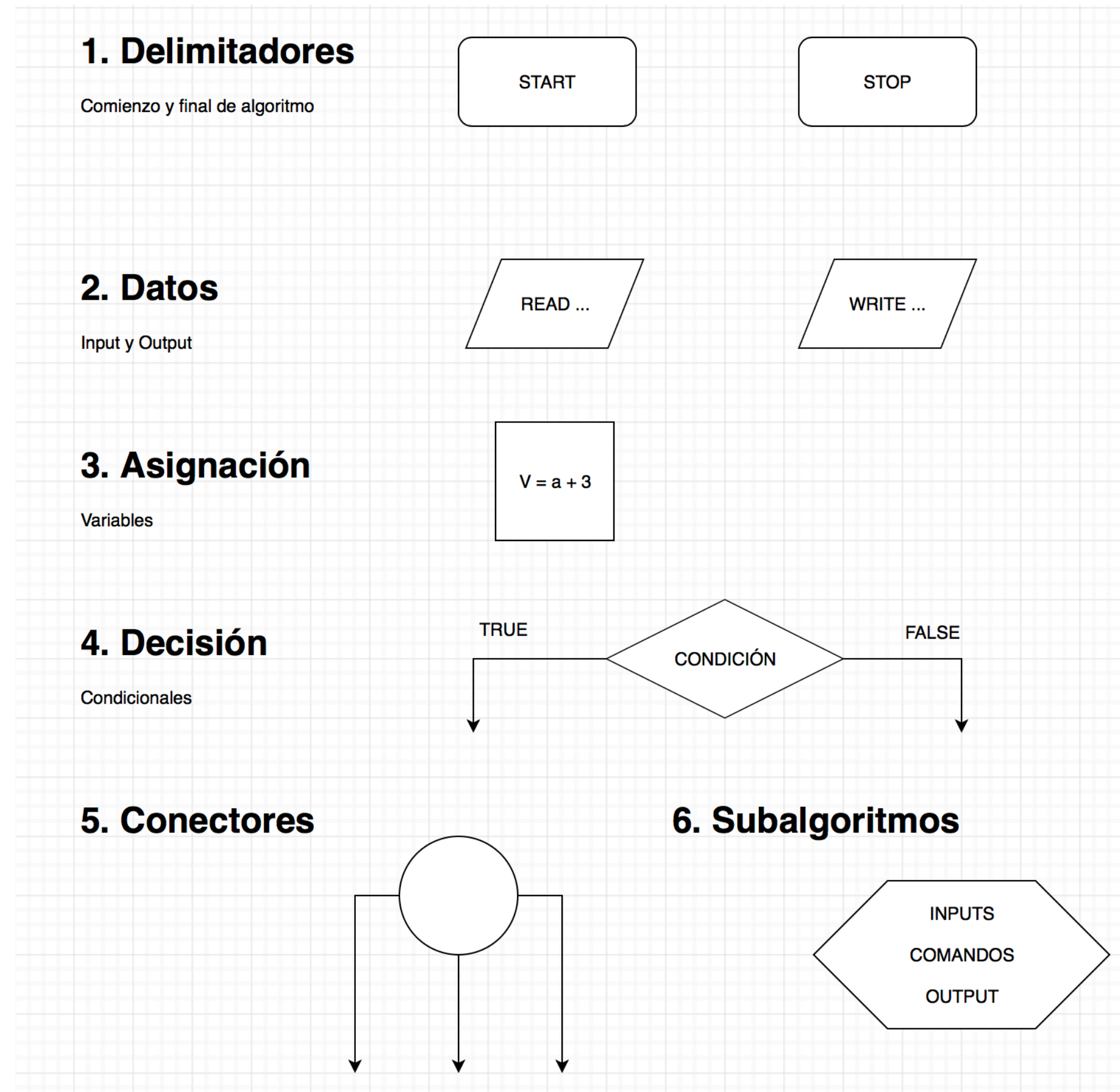
- Los diagramas lógicos o de flujo incluyen:
  - condiciones
  - estructuras iterativas
- Cada esquema debe tener un COMIENZO (**Start**) y un FINAL (**Stop**). Sólo debe haber un bloque **Start** y un **Stop** en un esquema.
- No se debe improvisar, sólo usar bloques.
- Hay que pensar cómo conectar los bloques.
- Flechas deben tocar los bloques de inicio y final.



Ejemplo de diagrama lógico

Creditos: Profesor Ismael Botti

# Diagramas Lógicos



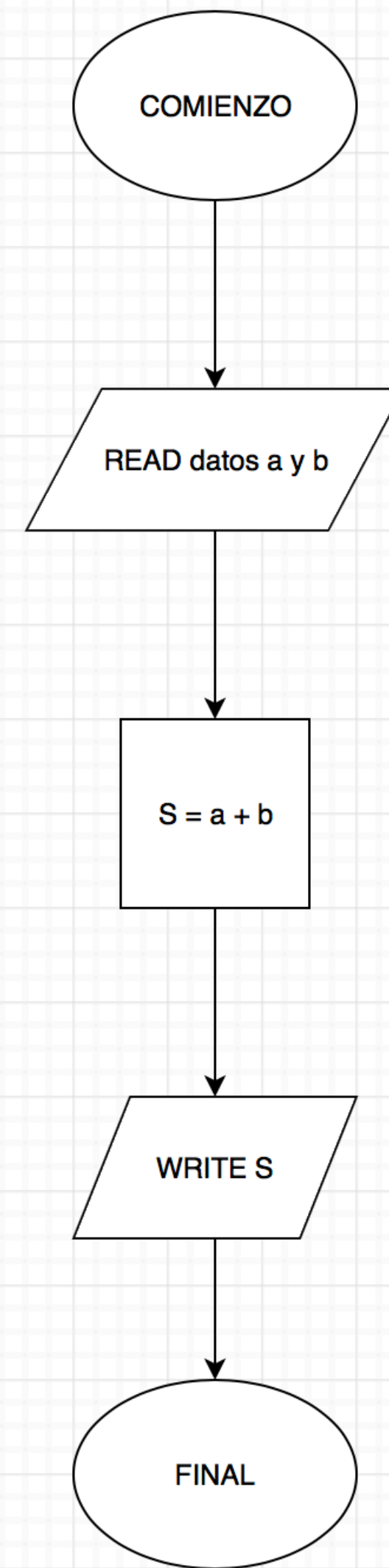
Elementos de un Diagrama de Flujo

Creditos: Profesor Ismael Botti

# Diagramas Lógicos

## Ejemplo 1

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

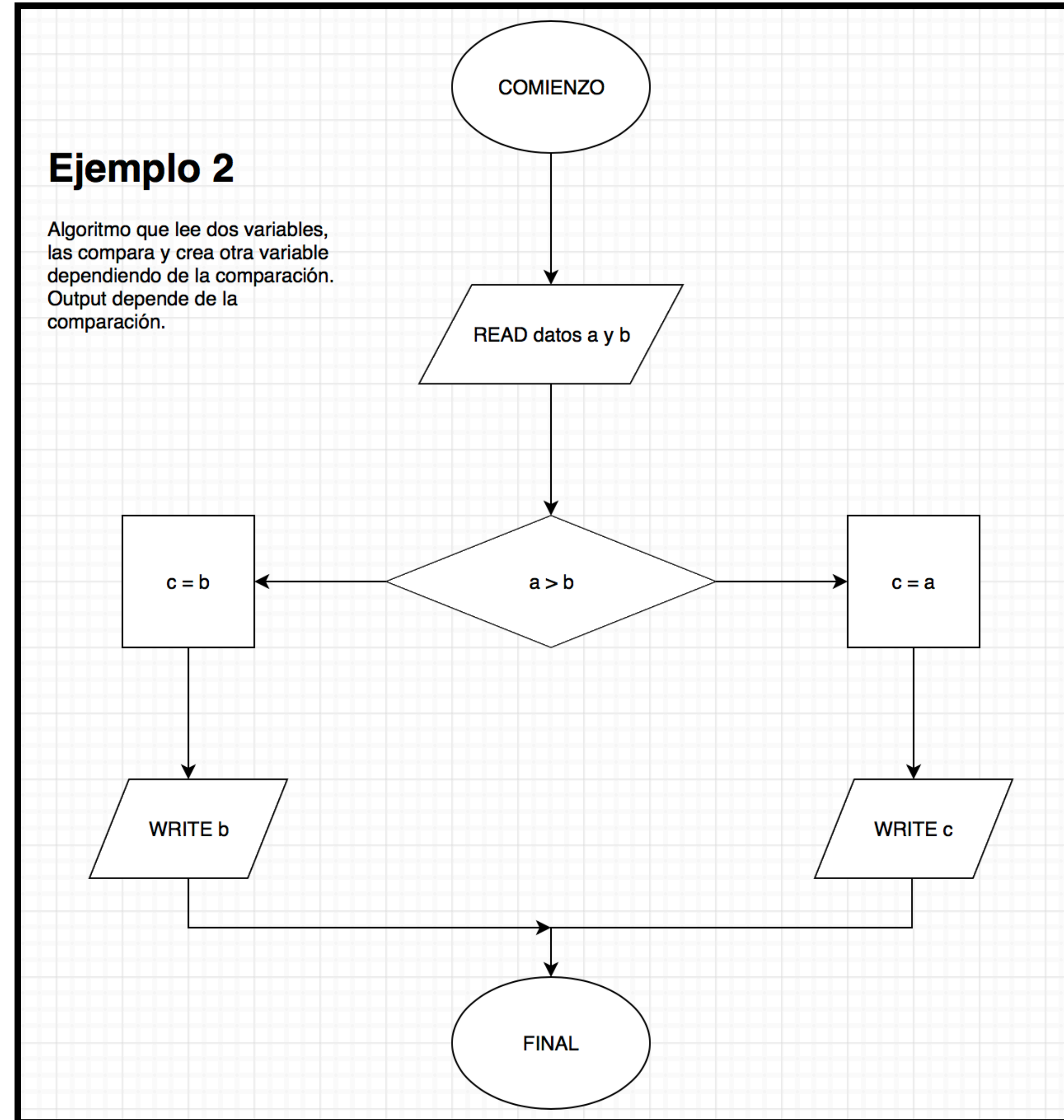


Creditos: Profesor Ismael Botti

# Diagramas Lógicos

## Ejemplo 2

Algoritmo que lee dos variables, las compara y crea otra variable dependiendo de la comparación. Output depende de la comparación.



# Condiciones

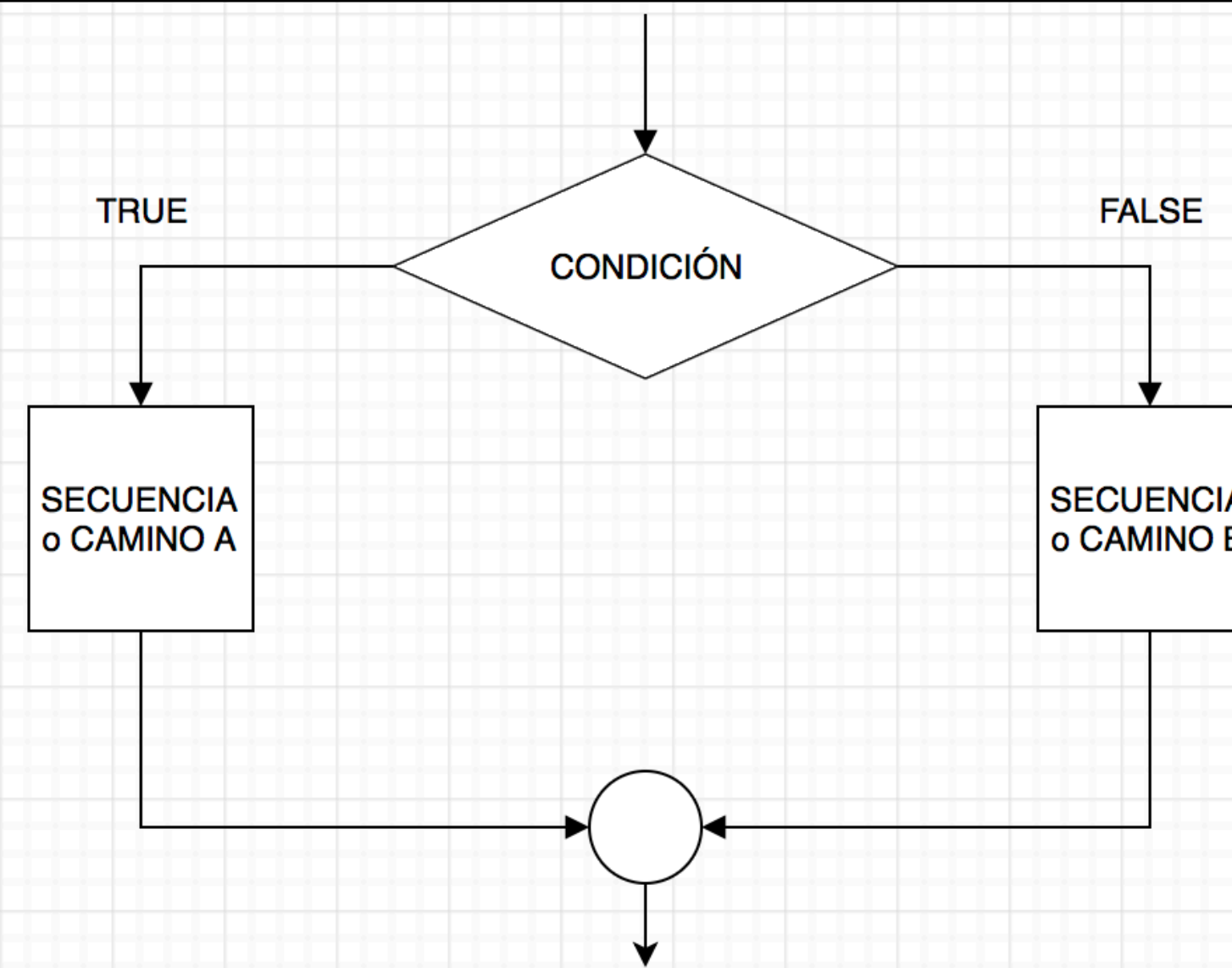
- Se usa cuando tenemos dos alternativas y sólo podemos escoger una.
- Es importante tener un criterio (condición matemática).
- Una vez que se haya optado por una opción o la otra, el algoritmo seguirá por un camino donde:
  - No puede volver atrás
  - No puede cambiar de alternativa
- Se pueden usar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

**Creditos: Profesor Ismael Botti**



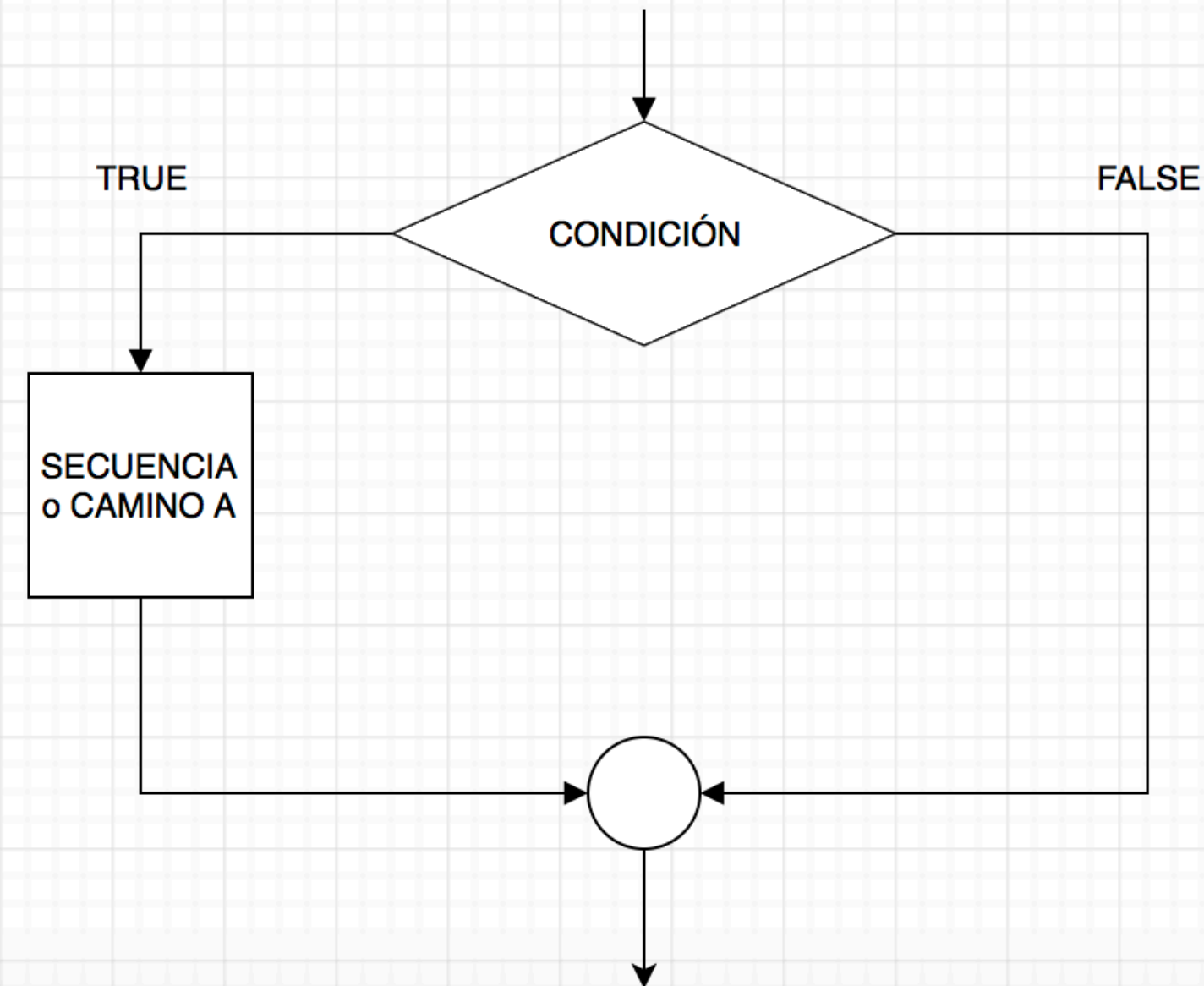
## Caso General

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.



Creditos: Profesor Ismael Botti

# 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
- **Secuencia de comandos:** el algoritmo no hace nada

Creditos: Profesor Ismael Botti



# Tipos de Loops

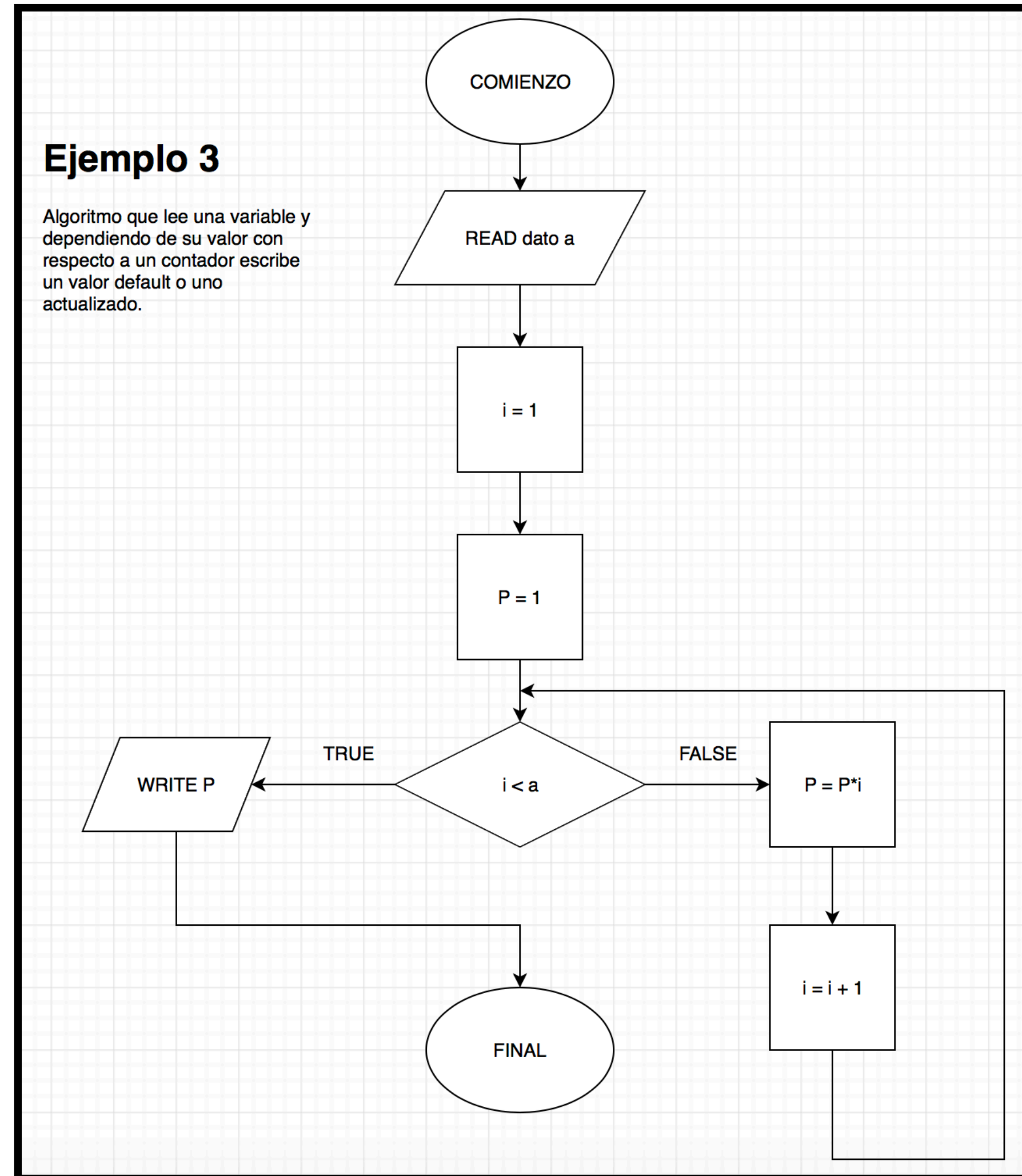
- Iteraciones que usan un test inicial
- Iteraciones que usan un test final
- Iteraciones que utilizan un contador

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

# Diagramas Lógicos

## Ejemplo 3

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



Creditos: Profesor Ismael Botti

# 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` pasos

# 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>