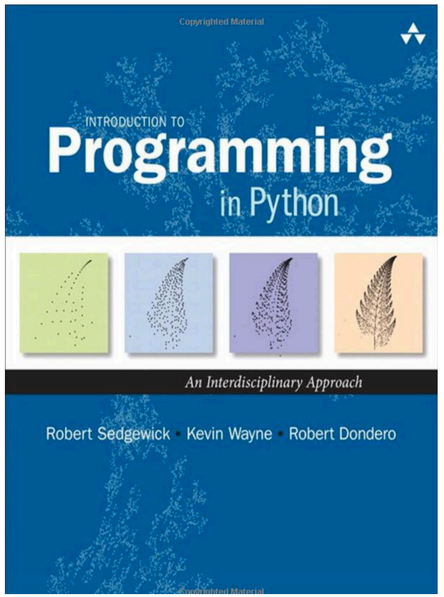
Parte I: Intro pensamiento computacional

Clase 04: Ciclo while for, break y continue

Diego Caro, Daniela Opitz e Ismael Botti
dcaro@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`
- Diagrámas Lógicos o de Flujo

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: "I))
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
```

$$\sum_{i=0} i$$

Acumuladores y Contadores

$$\sum_{i=0} i$$

```
#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)
```

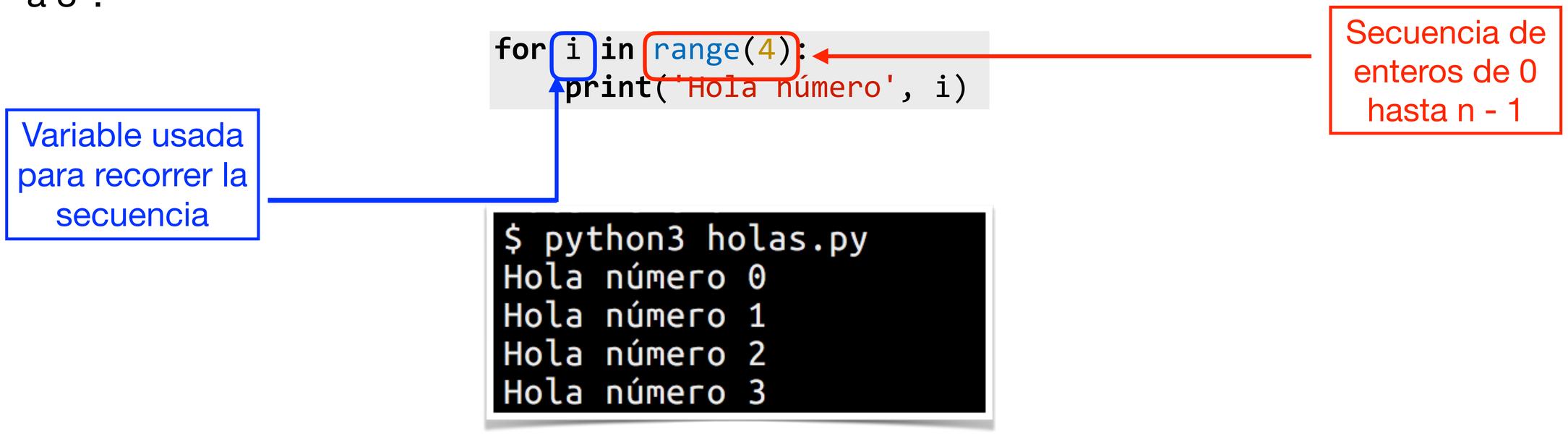
Voy contando los pares

Ciclo for

- **for**: Permite repetir un conjunto de instrucciones un numero determinado de veces. La secuencia de instrucciones se recorre en orden.
- Sintaxis:

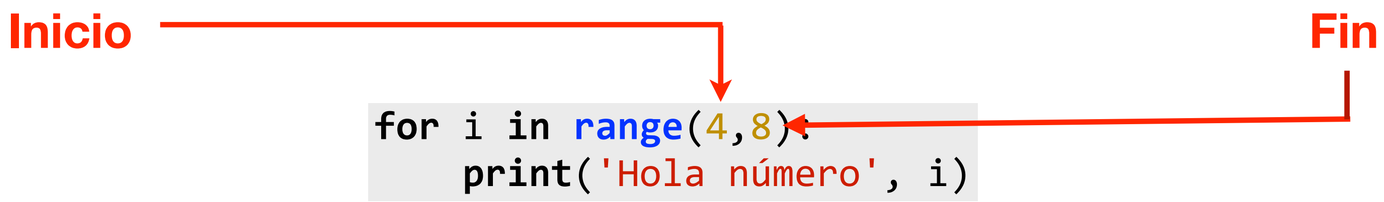
```
for <variable> in <elemento iterable>:
        <intrucciones>
```

- Ejemplo: Imprime el texto “Hola número n veces seguido del valor de n donde n va desde 0 a 3”.



Ciclo for

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



`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 **desconocido** de iteraciones	número **conocido** de iteraciones
no siempre puede ser sustituido por un ciclo for	**puede** ser sustituido por un ciclo while
necesita un contador que se inicie **antes** del loop y que se incremente **dentro** 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

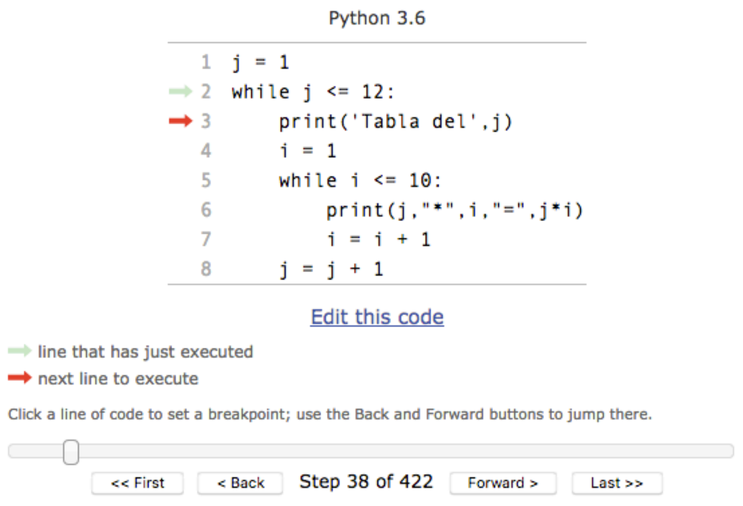
```
1 n = int(input('ingrese n: '))
2 for i in range(1, n, 2):
3     print(i)
```

Ejercicio

```
1 i = 1
2 while i <= 10:
3     print("7 *",i,"=",7*i)
4     i = i + 1
```

```
1 j = 1
2 while j <= 12:
3     print('Tabla del',j)
4     i = 1
5     while i <= 10:
6         print(j,"*",i,"=",j*i)
7         i = i + 1
8     j = j + 1
```

Chequear traza en https://goo.gl/cdGQx8



Human-based python interpreter™

- ¿Qué hace este programa?

```
1 a = 5
2 b = int(input())
3 if a + b < b:
4     print('Si')
5 else:
6     print('No')
```

Típico caso de loop

- Preguntar indefinidamente al usuario.
- Ejemplo: x es un número múltiplo de 7?

```
1 while True:
2     x = int(input('ingrese número entero: '))
3     if x % 7 == 0:
4         print(x, 'es múltiplo de 7')
5     else:
6         print(x, 'NO es múltiplo de 7')
```

Ciclo `for`

- Ejecutar código mientras se recorre una secuencia de elementos.
  - La secuencia se recorre en orden.
  - El término está garantizado.

Variable usada para recorrer la secuencia

Secuencia de enteros hasta n - 1

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

Salida:

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

Inicio **Fin**

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

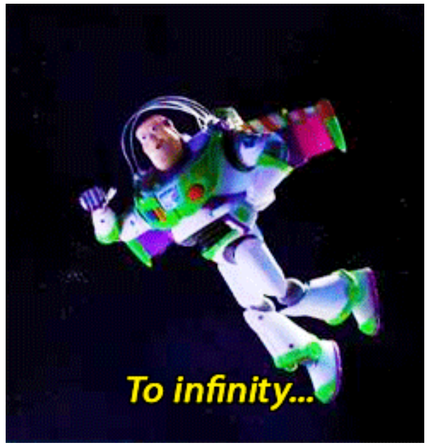
Salida:

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

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!
```



`while` vs `for`

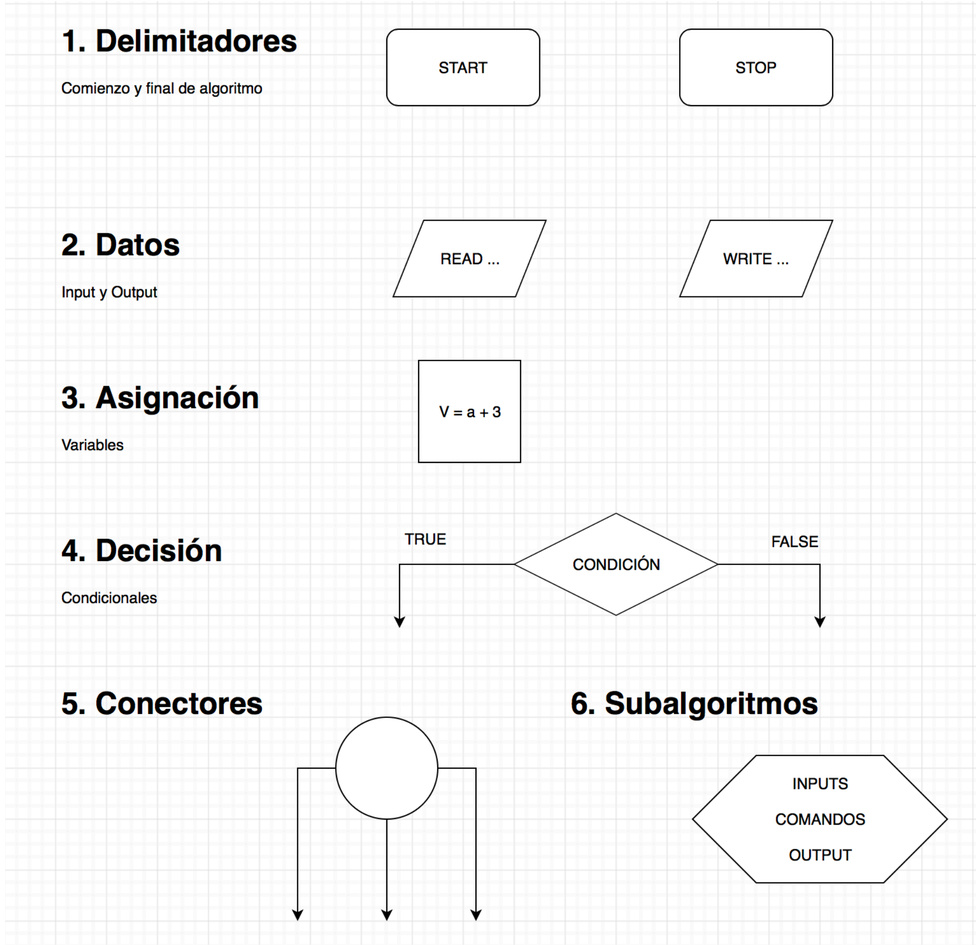
- Imprima todos los números impares menores que n mayores o iguales a cero.

```
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
```

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

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

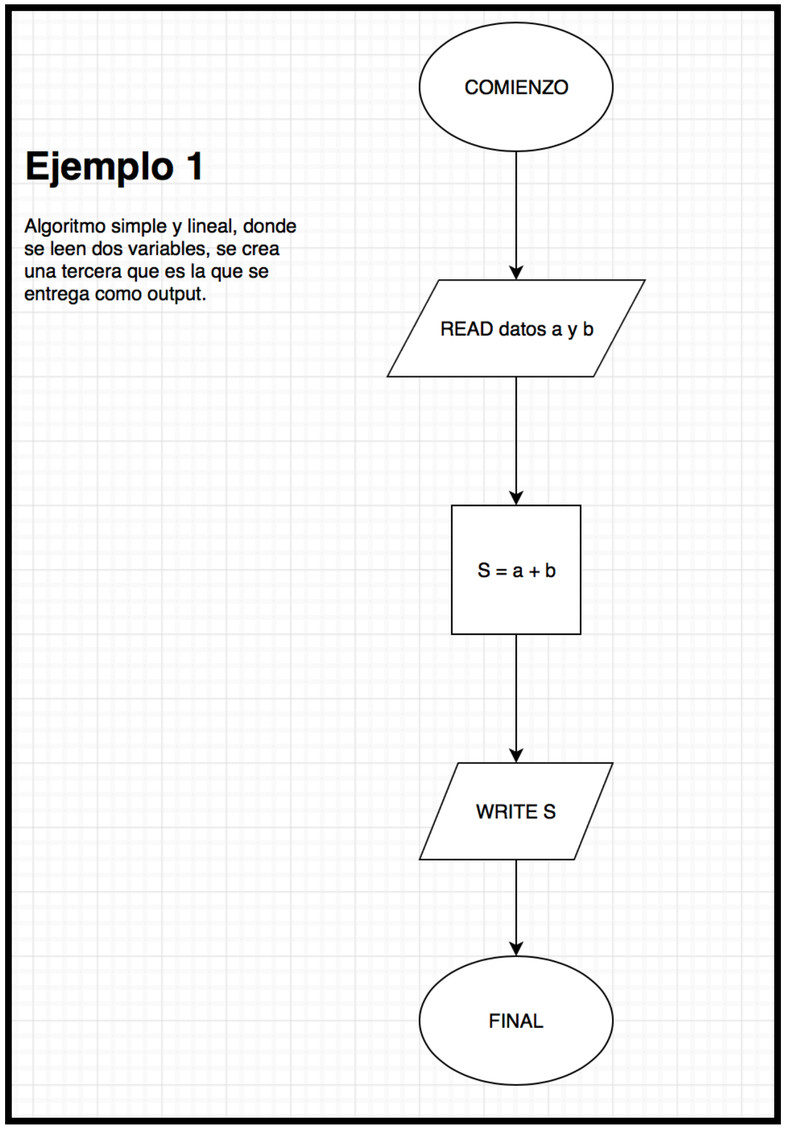
Diagramas Lógicos



Elementos de un Diagrama de Flujo

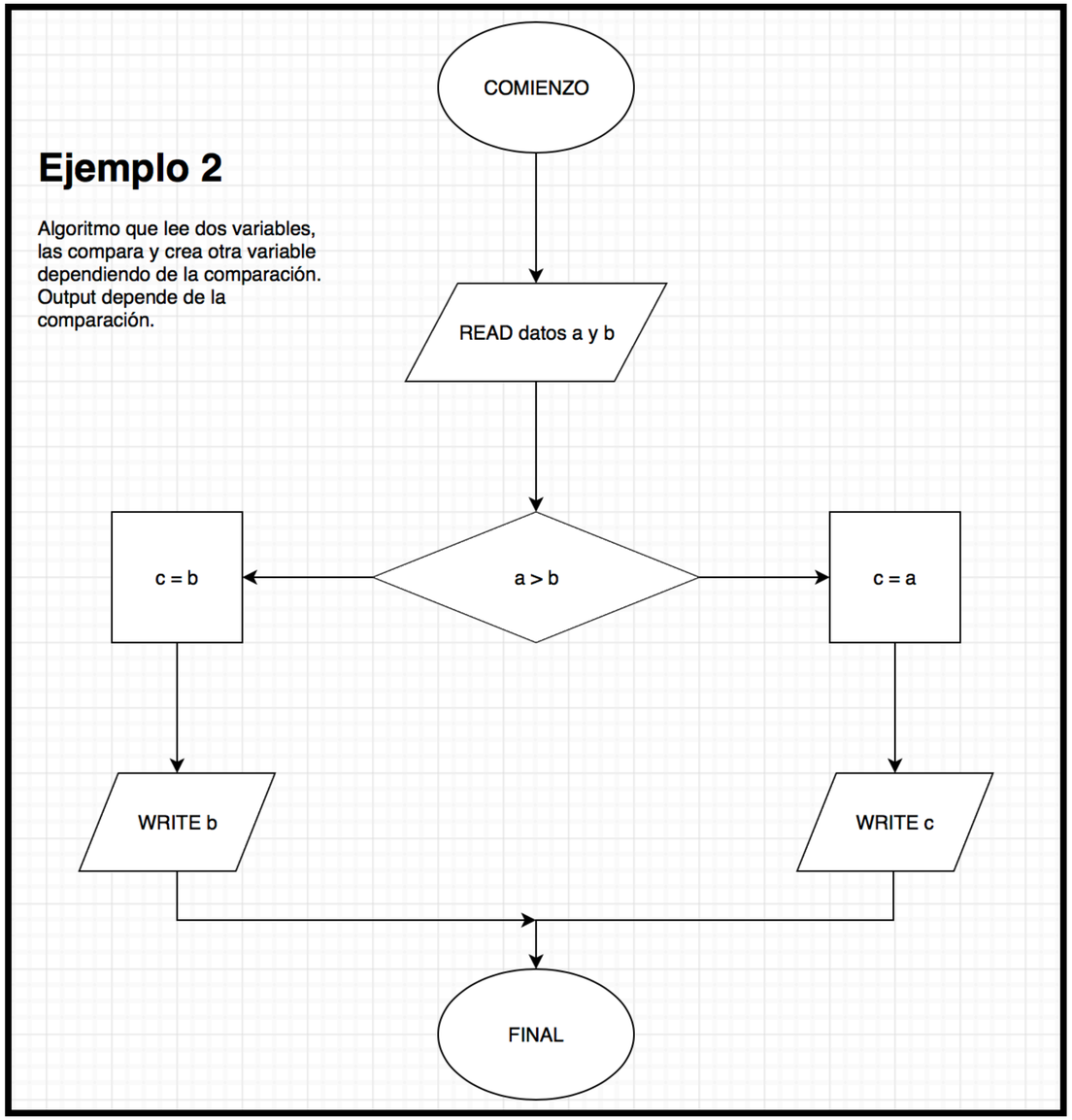
Creditos: Profesor Ismael Botti

Diagramas Lógicos



Creditos: Profesor Ismael Botti

Diagramas Lógicos

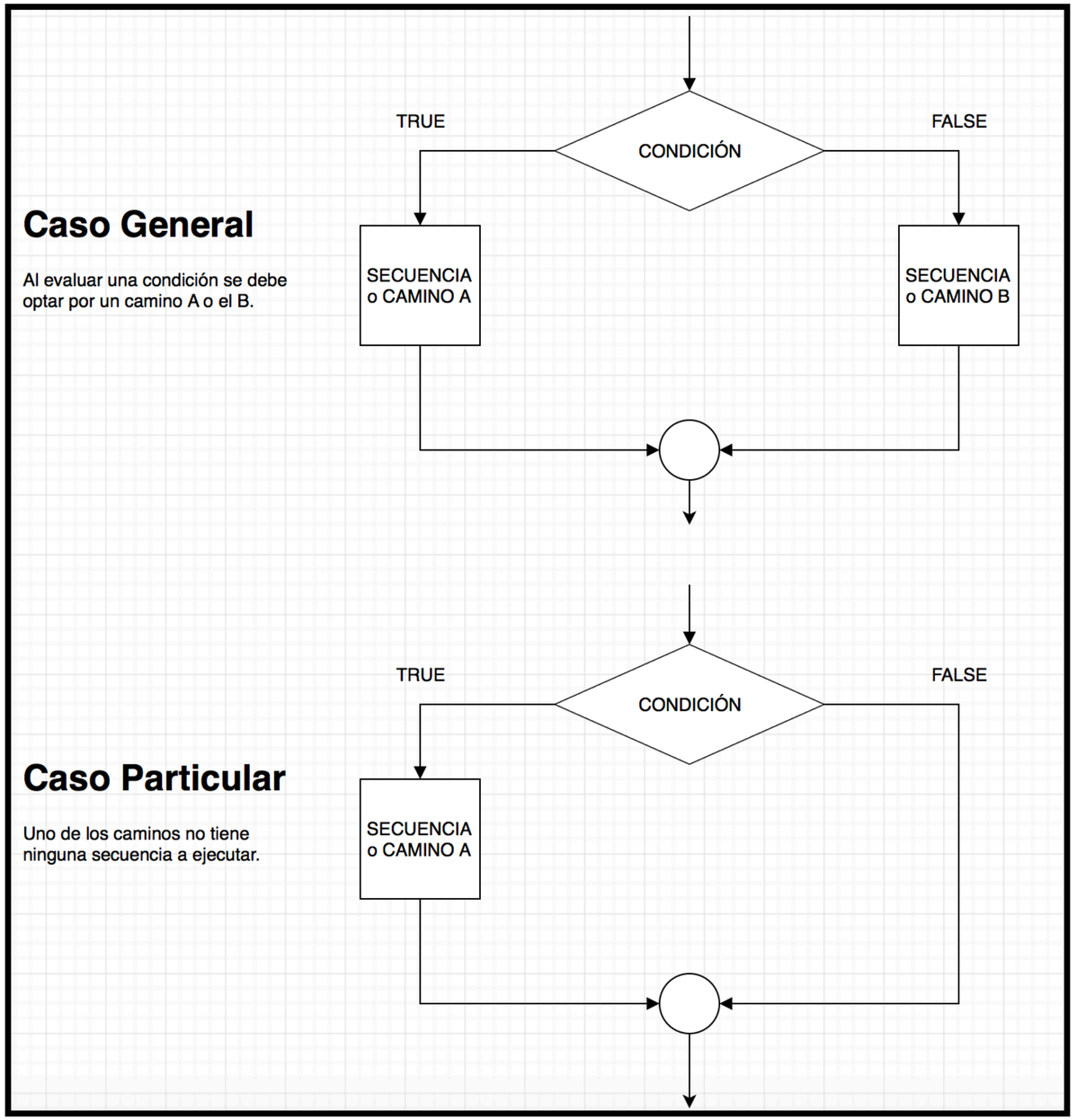


Creditos: Profesor Ismael Botti

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



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

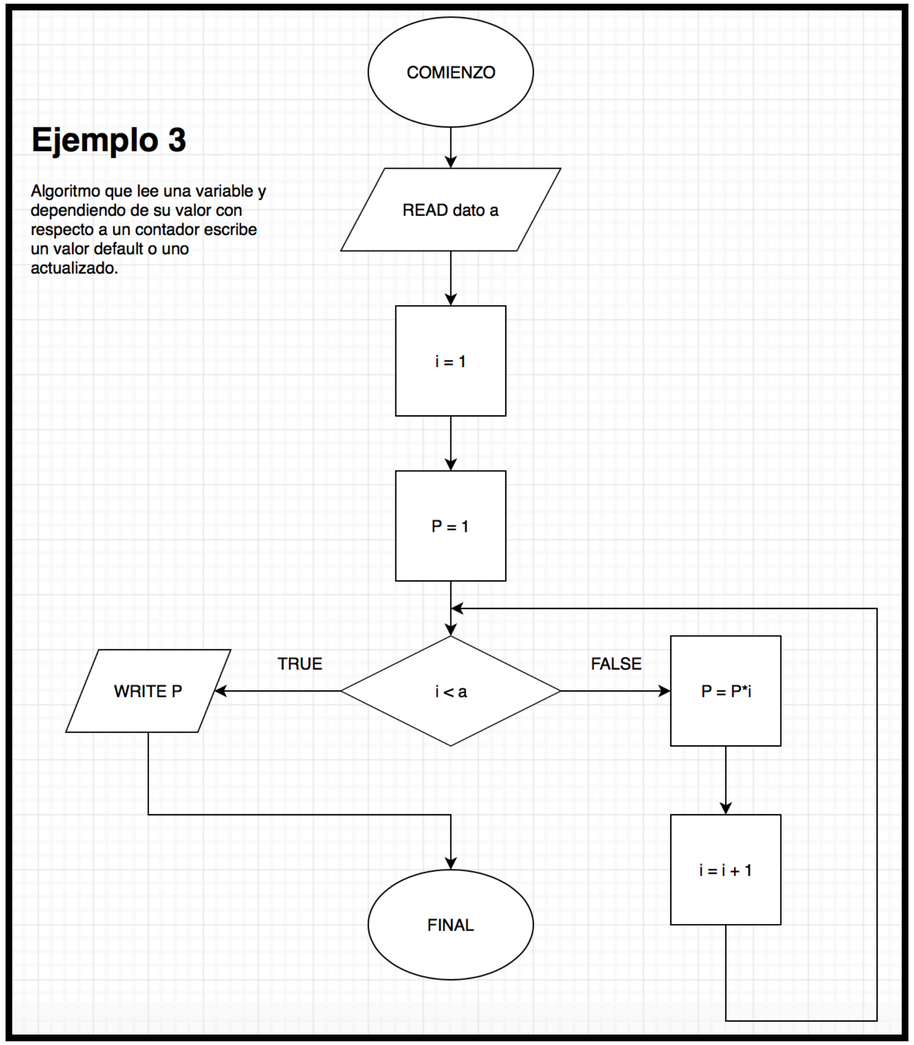
Tipos de Loops

- 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.

Creditos: Profesor Ismael Botti

Diagramas Lógicos



Creditos: Profesor Ismael Botti

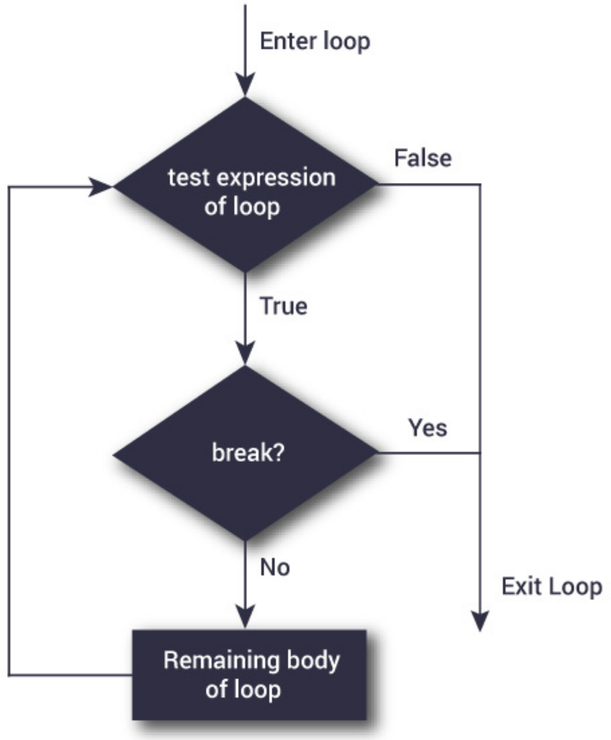
break: detener un ciclo

- Los ciclos se pueden detener antes de que recorran la secuencia o la condición en while no se cumpla. Keyword: **`break`**
- **Ventaja**: podemos ahorrar tiempo de procesador (muuuuuy poco).
- **Desventaja**: código más complejo.

```
for var in secuencia:
    # código dentro del ciclo for
    if condicion:
        break # detiene el ciclo for
    # código dentro del ciclo for
#código fuera del ciclo for

--

while test expresión:
    # código dentro del ciclo while
    if condicion:
        break # detiene el ciclo while
    # código dentro del ciclo while
#código fuera del ciclo while
```




```
1 for e in 'hola':
2     if e == 'l':
3         break
4     print(e)
```

```
$ python3 simple-break.py
h
o
```

Nota: si necesitas usar **`break`**, verifica que sea la alternativa más sencilla.

Imagen extraída desde https://www.programiz.com/python-programming/break-continue

continue: saltar a siguiente iteración

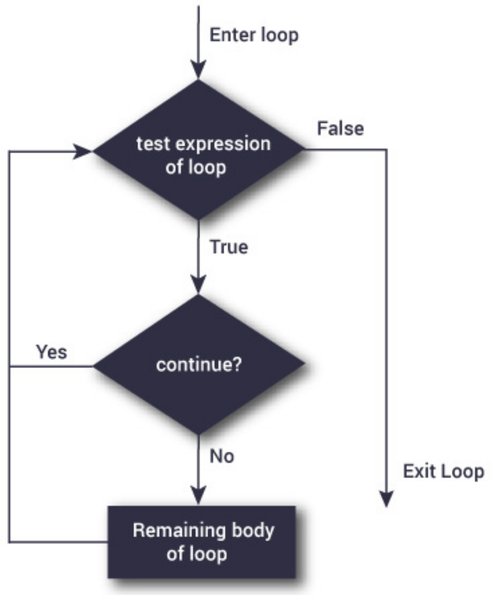
- A veces queremos saltarnos alguna iteración (ej.: ignorar elementos negativos). Puedes saltarlos usando **`continue`**.
- **Ventaja**: podemos ahorrar tiempo de procesador (muuuuuy poco).
- **Desventaja**: código más complejo.

```
for var in secuencia:
    # código dentro del ciclo for
    if condicion:
        continue # salta a siguiente iteración
    # código dentro del ciclo for

#código fuera del ciclo for

while test expresión:
    # código dentro del ciclo while
    if condicion:
        continue # salta a siguiente iteración
    # código dentro del ciclo while

#código fuera del ciclo while
```




```
1 for e in 'hola':
2     if e == 'l':
3         continue
4     print(e)
```

```
$ python3 simple-continue.py
h
o
a
```

Nota: si necesitas usar **`continue`**, verifica que sea la alternativa más sencilla.

Imagen extraída desde https://www.programiz.com/python-programming/break-continue

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

		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