

CÓNQUER BLOCKS

PYTHON

LISTAS Y ESTRUCTURAS ITERATIVAS (PARTE 1)

REPASO CLASE ANTERIOR

1. Test condicionales simples
2. Test condicionales múltiples
3. Expresión If simple
4. Expresión If anidada
5. Sustitución del switch case por expresiones If

LISTAS

¿QUÉ ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

```
Lista1 = [ "a", "b", "c", "d", "e", "f"]
```

```
Lista2 = [ 1, 2, 3, 4, 5, 6 ]
```

```
Lista3 = [ 1, 3, 7, 2, 5, 1, 13 ]
```

```
Lista4 = [ 1, "a", "b", 3, 7, "e"]
```

LISTAS

¿QUÉ ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

```
Lista1 = [ "a", "b", "c", "d", "e", "f" ]
```

```
Lista2 = [ 1, 2, 3, 4, 5, 6 ]
```

```
Lista3 = [ 1, 3, 7, 2, 5, 1, 13 ]
```

```
Lista4 = [ 1, "a", "b", 3, 7, "e" ]
```

LISTAS

¿QUE ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones)
```

✓ 0.1s

```
['bote', 'yate', 'velero', 'catamarán']
```

LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones[0])
```

✓ 0.7s

bote

LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones[0])
```

✓ 0.7s

bote

¡Es un string!

LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(type(embarcaciones))  
print(type(embarcaciones[0]))
```

✓ 0.8s

```
<class 'list'>
```

```
<class 'str'>
```


LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones[0].title())
```

✓ 0.6s

Bote

LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones[1], embarcaciones[2])
```

✓ 0.8s

yate velero

LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
print(embarcaciones[-1])
```

✓ 0.1s

catamarán

LISTAS

USAR UN ELEMENTO DE LA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']  
mensaje = 'Si me comprase una embarcación, elegiría un ' + embarcaciones[-1].title()  
print(mensaje)
```

✓ 0.9s

Si me comprase una embarcación, elegiría un Catamarán

LISTAS

MODIFICAR ELEMENTOS

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coches[2] = 'mercedes'  
print(coches)
```

✓ 0.6s

```
['bmw', 'audi', 'seat']
```

```
['bmw', 'audi', 'mercedes']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Append() nos permite añadir elementos a una lista en la última posición.

APPEND()

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coches.append('mercedes')  
print(coches)
```

✓ 0.1s

```
['bmw', 'audi', 'seat']
```

```
['bmw', 'audi', 'seat', 'mercedes']
```

Las listas son objetos dinámicos. Se les puede asignar más memoria de manera dinámica.

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Append() nos permite añadir elementos a una lista de manera dinámica.

```
coches = []  
coches.append('bmw')  
coches.append('audi')  
coches.append('mercedes')  
print(coches)
```

✓ 0.9s

```
['bmw', 'audi', 'mercedes']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Insert() nos permite añadir elementos a una lista en cualquier posición

INSERT()

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coches.insert(0, 'mercedes')  
print(coches)
```

✓ 0.1s

```
['bmw', 'audi', 'seat']
```

```
['mercedes', 'bmw', 'audi', 'seat']
```


LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Insert() nos permite añadir elementos a una lista en cualquier posición

INSERT()

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coches.insert(1, 'mercedes')  
print(coches)
```

✓ 0.8s

```
['bmw', 'audi', 'seat']
```

```
['bmw', 'mercedes', 'audi', 'seat']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coches.pop()  
print(coches)
```

✓ 0.1s

```
['bmw', 'audi', 'seat']  
['bmw', 'audi']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
coches = ['bmw', 'audi', 'seat']  
print(coches)  
coche_eliminado = coches.pop()  
print(coches)  
print(coche_eliminado)
```

✓ 0.1s

```
['bmw', 'audi', 'seat']
```

```
['bmw', 'audi']
```

```
seat
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
✓  
coches = ['bmw', 'audi', 'seat']  
coche_eliminado = coches.pop(1)  
print(coche_eliminado)  
print(coches)
```

✓ 0.1s

audi

['bmw', 'seat']

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

REMOVE()

```
coches = ['bmw', 'audi', 'seat']  
coches.remove('audi')  
print(coches)
```

✓ 0.7s

```
['bmw', 'seat']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

REMOVE()

```
coches = ['bmw', 'audi', 'seat', 'audi']  
coches.remove('audi')  
print(coches)
```

✓ 0.1s

```
['bmw', 'seat', 'audi']
```

Elimina la primera aparición del elemento indicado

LISTAS

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

DEL - Palabra Clave (Keyword)

```
coches = ['bmw', 'audi', 'seat']  
del coches[1]  
print(coches)
```

✓ 0.1s

```
['bmw', 'seat']
```

LISTAS

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

DEL - Palabra Clave (Keyword)

```
coches = ['bmw', 'audi', 'seat']
del coches
print(coches)
```

⊗ 0.8s

```
NameError                                Traceback (most recent call last)
Cell In[25], line 3
      1 coches = ['bmw', 'audi', 'seat']
      2 del coches
----> 3 print(coches)

NameError: name 'coches' is not defined
```


LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *permanente*

SORT()

```
coches = ['bmw', 'audi', 'seat']  
coches.sort()  
print(coches)
```

✓ 0.7s

```
['audi', 'bmw', 'seat']
```

```
coches = [23, 11, 5]  
coches.sort()  
print(coches)
```

✓ 0.6s

```
[5, 11, 23]
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *permanente*

SORT()

```
coches = ['bmw', 'audi', 'seat']  
coches.sort()  
print(coches)
```

✓ 0.7s

```
['audi', 'bmw', 'seat']
```

```
coches = ['23', '11', '5']  
coches.sort()  
print(coches)
```

✓ 0.9s

```
['11', '23', '5']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *temporal*

SORTED()

```
coches = ['bmw', 'audi', 'seat']  
print(sorted(coches))
```

✓ 0.9s

```
['audi', 'bmw', 'seat']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *temporal*

SORTED()

```
coches = ['bmw', 'audi', 'seat']  
autos = sorted(coches)  
print(autos)  
print(sorted(coches))
```

✓ 0.1s

```
['audi', 'bmw', 'seat']
```

```
['audi', 'bmw', 'seat']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos imprimir la lista en orden inverso.

REVERSE()

```
coches = ['bmw', 'audi', 'seat']  
coches.reverse()  
print(coches)
```

✓ 0.1s

```
['seat', 'audi', 'bmw']
```

LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

LONGITUD DE UNA LISTA

LEN()

```
coches = ['bmw', 'audi', 'seat']  
print(len(coches))
```

✓ 0.5s

3

LISTAS

DEBUGGING

```
coches = ['bmw', 'audi', 'seat']  
print(coches[3])
```

⊗ 0.9s

IndexError

Traceback (most recent call last)

Cell In[43], line 2

```
1 coches = ['bmw', 'audi', 'seat']  
----> 2 print(coches[3])
```

IndexError: list index out of range

LISTAS

DEBUGGING

```
coches = []  
print(coches[0])
```

⊗ 0.1s

IndexError

Traceback (most recent call last)

Cell In[45], line 2

```
1 coches = []  
----> 2 print(coches[0])
```

IndexError: list index out of range

LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si un elemento está en la lista

```
coches = ['bmw', 'audi', 'seat']  
coche_elegido = 'audi'  
if coche_elegido in coches:  
    print('Has escogido un', coche_elegido )  
else:  
    print('No tenemos esa marca de coche')
```

✓ 0.1s

Has escogido un audi

LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si un elemento está en la lista

```
coches = ['bmw', 'audi', 'seat']  
coche_elegido = 'mercedes'  
if coche_elegido in coches:  
    print('Has escogido un', coche_elegido )  
else:  
    print('No tenemos esa marca de coche')
```

✓ 0.9s

No tenemos esa marca de coche

LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si la lista tiene contenido

```
coches = []  
if coches:  
    print('La lista tiene el siguiente contenido ', coches)  
else:  
    print('La lista está vacía')
```

✓ 0.1s

La lista está vacía

LISTAS Y ESTRUCTURA CONDICIONAL

```
coches = ['seat']  
if coches:  
    print('La lista tiene el siguiente contenido ', coches)  
else:  
    print('La lista está vacía')
```

✓ 0.3s

La lista tiene el siguiente contenido ['seat']

ESTRUCTURAS ITERATIVAS

¿QUÉ ES UNA ESTRUCTURA ITERATIVA O BUCLE?

Es una secuencia de instrucciones que se ejecuta repetidamente mientras se cumple cierta condición establecida previamente.

ESTRUCTURAS ITERATIVAS EN PYTHON



WHILE = MIENTRAS

FOR = PARA

ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Mientras expr_logica Hacer
.....
ejecucion_codigo
FinMientras

Mientras (expr_logica_1) OP_LOG (expr_logica_2) Hacer
.....
ejecucion_codigo
FinMientras
```

PYTHON

```
while expr_logica:
|   ejecucion_codigo
while (expr_logica_1) and/or (expr_logica_2)
|   ejecucion_codigo
```

Para entrar en el ciclo WHILE debe cumplirse la condición de la expresión lógica.

Pero debe haber alguna opción para que la expresión no se cumpla y se salga del bucle.



Debe cambiarse el valor de la expresión lógica

ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Definir temporizador Como Entero
temporizador = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Mientras (temporizador > 0) Hacer
    Escribir "Quedan ", temporizador, " segundos"
    Esperar 1 Segundos
    temporizador = temporizador - 1
FinMientras
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de segundos del temporizador'))

print('Comienza el temporizador')

while (temporizador > 0):
    print('Quedan ', temporizador, ' segundos')
    time.sleep(1)
    temporizador = temporizador - 1

print('¡El temporizador ha finalizado!')
```

ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Definir temporizador Como Entero
temporizador = 0

Escribir "Introduce el número de
Leer temporizador

Escribir "Comienza el temporizad
Mientras (temporizador > 0) Hace
    Escribir "Quedan ", temporiz
    Esperar 1 Segundos
    temporizador = temporizador - 1
FinMientras
Escribir "El temporizador ha finalizado!!"
```

```
Introduce el número de
segundos del temporizador 3
Comienza el temporizador
Quedan 3 segundos
Quedan 2 segundos
Quedan 1 segundos
¡El temporizador ha finalizado!
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de
                        zador'))

temporizador')

> 0):
    , temporizador, ' segundos')
    temporizador - 1

print('¡El temporizador ha finalizado!')
```


ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Para i = valor_inicial Hasta valor_final Con Paso incr_decr Hacer
    ejecucion_codigo
FinPara
```

PYTHON

```
for i in range(valor_inicial, valor_final, paso):
    ejecucion_codigo
```

El número de repeticiones viene dado por el valor de la variable i

No es necesario incluir una instrucción para cambiar la condición del bucle.

ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de segundos del temporizador'))

print('Comienza el temporizador')

for i in range(0, temporizador):
    print('Quedan ', temporizador, ' segundos')
    time.sleep(1)
    temporizador = temporizador - 1

print('¡El temporizador ha finalizado!')
```

ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de segundos del temporizador'))

print('Comienza el temporizador ')

for i in range(temporizador,0,-1):
    print('Quedan ', i, ' segundos')
    time.sleep(1)

print('¡El temporizador ha finalizado!')
```

ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de s
Leer temporizador

Escribir "Comienza el temporizador
Para i = temporizador Hasta 1 Con
    Escribir "Quedan ", i, " segun
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

```
Introduce el número de
segundos del temporizador 3
Comienza el temporizador
Quedan 3 segundos
Quedan 2 segundos
Quedan 1 segundos
¡El temporizador ha finalizado!
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de
'))

izador')

zador):
orizador, ' segundos')

izador - 1

print('¡El temporizador ha finalizado!')
```


ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
for i in range(0,3):
    print(i)
```

✓ 0.9s

0

1

2

i va de 0 a 2

ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
for i in range(1,3):
    print(i)
```

✓ 0.7s

1
2

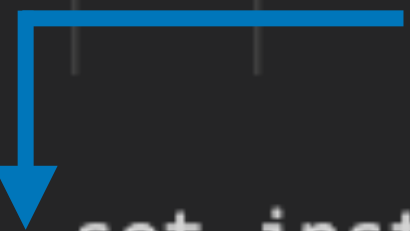
i va de 1 a 2

ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*

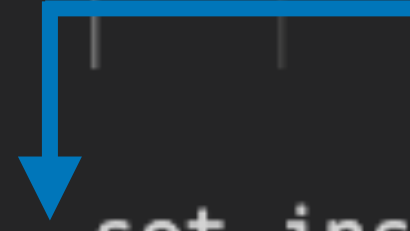
Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
set_instrucciones
```



Exit point

```
while condicion
    set_instrucciones
    if condicion_a:
        break
set_instrucciones
```



ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
set_instrucciones
```

Exit point

```
for i in range(5):
    if i == 3:
        break
    print(i)
print("Estamos fuera del bucle")
```

✓ 0.2s

0
1
2
Estamos fuera del bucle

ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
set_instrucciones
```

Exit point

```
for i in range(5):
    if i == 3:
        print("Estoy dentro del if")
        break
    print("Estoy dentro del if y", \
          "despues del break")
    print(i)
print("Estamos fuera del bucle")
```

✓ 0.1s

0

1

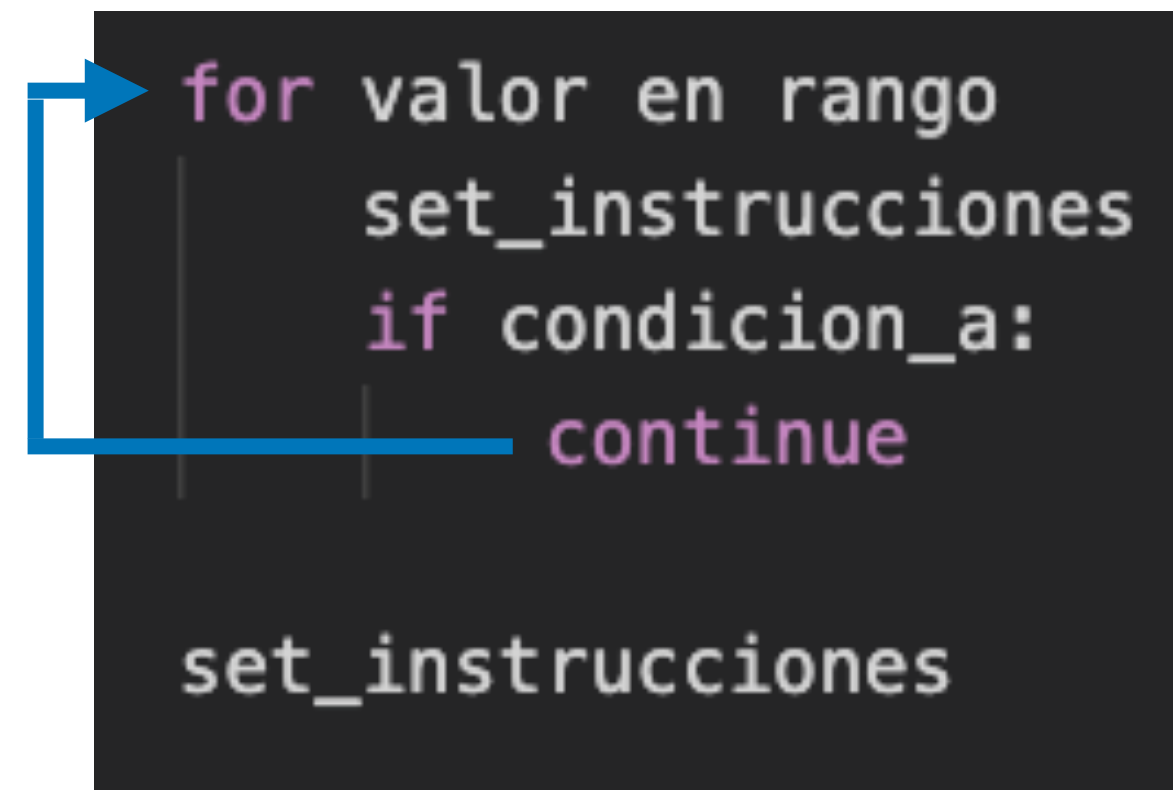
2

Estoy dentro del if

Estamos fuera del bucle

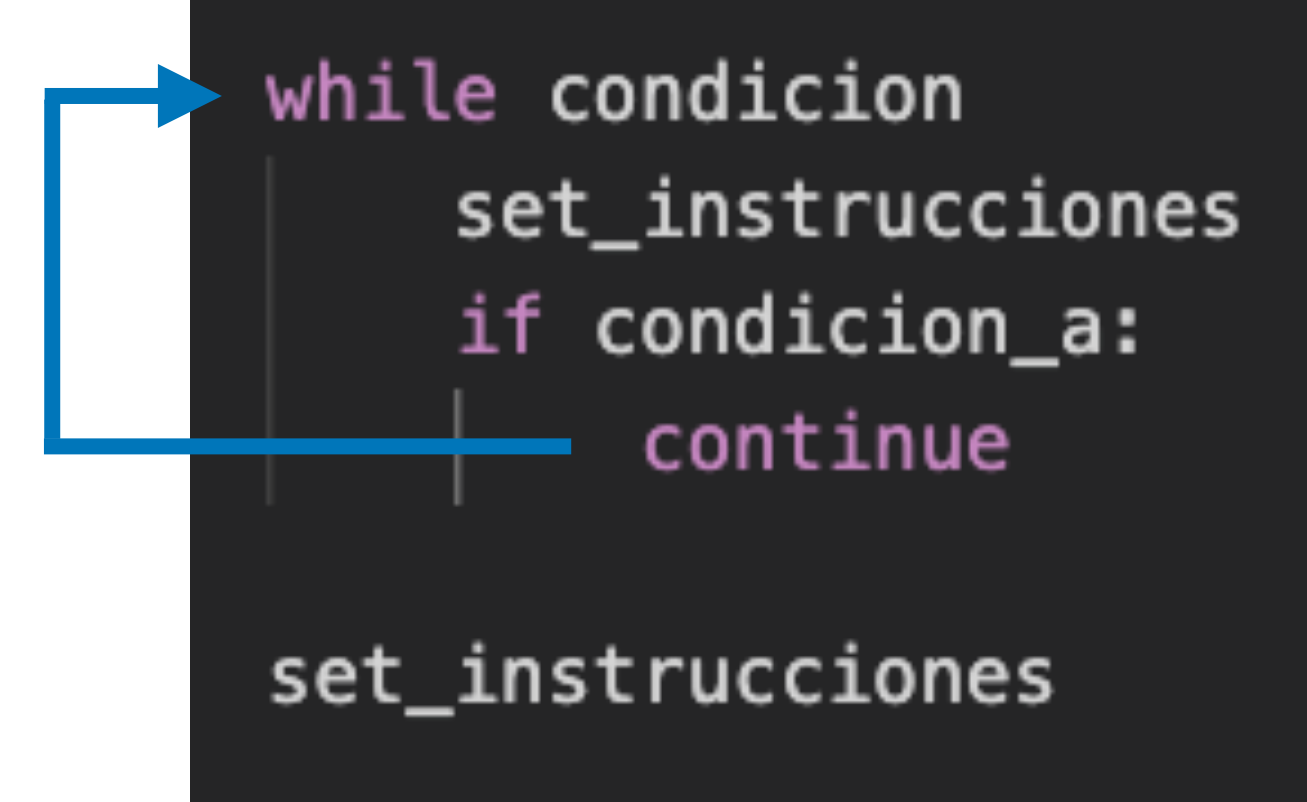
ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *CONTINUE*



```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
set_instrucciones
```

A blue arrow points to the start of the loop. Another blue arrow points from the `continue` statement to the start of the loop, indicating that the loop restarts from the beginning, skipping the rest of the code in the loop body.



```
while condicion
    set_instrucciones
    if condicion_a:
        continue
set_instrucciones
```

A blue arrow points to the start of the loop. Another blue arrow points from the `continue` statement to the start of the loop, indicating that the loop restarts from the beginning, skipping the rest of the code in the loop body.

ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *CONTINUE*

```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
    set_instrucciones
```

```
for i in range(5):
    if i == 3:
        print("Estoy dentro del if")
        continue
    print("Estoy dentro del if y", \
          "despues del break")
    print(i)

print("Estamos fuera del bucle")
```

✓ 0.1s

0

1

2

Estoy dentro del if

4

Estamos fuera del bucle

ESTRUCTURAS ITERATIVAS

CONTINUE y *BREAK*

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.



Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
set_instrucciones
```

```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
set_instrucciones
```

ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

→ Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

Exit point

```
for valor en rango
  set_instrucciones
  if condicion_a:
    break
set_instrucciones
```

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.


```
for valor en rango
  set_instrucciones
  if condicion_a:
    continue
set_instrucciones
```

ESTRUCTURAS ITERATIVAS


CONTINUE y BREAK

Exit point

```
for valor en rango
  set_instrucciones
  if condicion_a:
    break
set_instrucciones
```



```
for valor en rango
  set_instrucciones
  if condicion_a:
    continue
set_instrucciones
```



Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

→ Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.

Casos de uso:

Para crear comprobaciones rápidas al comienzo del programa → actual como pre-condiciones.

ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Exit point

```
for valor en rango
  set_instrucciones
  if condicion_a:
    break
set_instrucciones
```

```
for valor en rango
  set_instrucciones
  if condicion_a:
    continue
set_instrucciones
```

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.



Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.

Casos de uso:

Para crear comprobaciones rápidas al comienzo del programa —> actual como pre-condiciones.

Regla de estilo a seguir:

Echa un ojo al código. Piensa si hay alguna manera de escribirlo sin usar *break* o *continue* y que mantenga el código sencillo y elegante. Si ves que esto no es posible entonces usa *break* o *continue*.

CÔNQUER BLOCKS