

Programación en Python

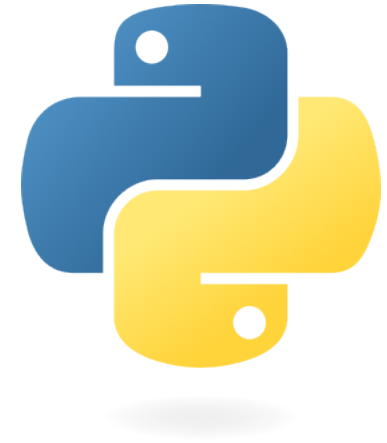
Clase 1: Conociendo Python

18/03/2026

Qué es Python

Python en una *slide*

- Lenguaje interpretado



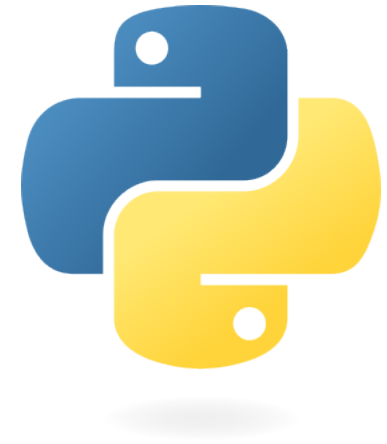
Python en una *slide*

- Lenguaje interpretado
- De propósito general



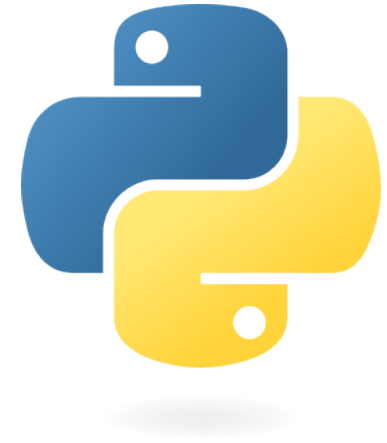
Python en una *slide*

- Lenguaje interpretado
- De propósito general
- De alto nivel



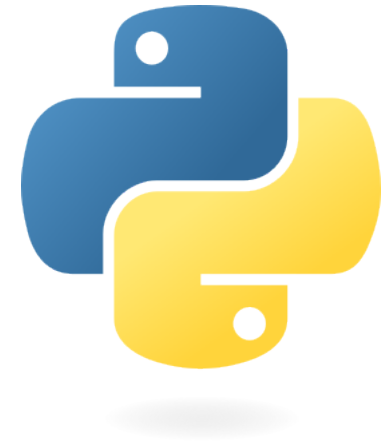
Python en una *slide*

- Lenguaje interpretado
- De propósito general
- De alto nivel
- Gratis



Python en una *slide*

- Lenguaje interpretado
- De propósito general
- De alto nivel
- Gratis
- Código abierto



Python en una *slide*

- Lenguaje interpretado
- De propósito general
- De alto nivel
- Gratis
- Código abierto
- Ecosistema de librerías



Python en una *slide*

- Lenguaje interpretado
- De propósito general
- De alto nivel
- Gratis
- Código abierto
- Ecosistema de librerías
- Comunidad



Positron

Positron en una *slide*

- Editor de código



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code
- Diseñado para ciencia de datos



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code
- Diseñado para ciencia de datos
- Enfocado en R y Python



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code
- Diseñado para ciencia de datos
- Enfocado en R y Python
- Gratis



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code
- Diseñado para ciencia de datos
- Enfocado en R y Python
- Gratis
- Código abierto



Positron en una *slide*

- Editor de código
- Creado por Posit (antes RStudio)
- Basado en Visual Studio Code
- Diseñado para ciencia de datos
- Enfocado en R y Python
- Gratis
- Código abierto
- Rstudio ❤️ VS Code



A laburar

Tipos de datos elementales

Tipos de datos elementales

Python cuenta con 5 tipos de datos elementales:
int, float, str, bool, NoneType.

```
13                # <class 'int'>
2.71              # <class 'float'>
13.0             # <class 'float'>
13.              # <class 'float'>
.7              # <class 'float'>
"Hola hola"     # <class 'str'>
'Hola hola'     # <class 'str'>
True            # <class 'bool'>
False          # <class 'bool'>
None           # <class 'NoneType'>
```

i Escalares de verdad

Python no es un lenguaje vectorizado, a diferencia de R. Los valores enteros, flotantes, etc. son **escalares** de verdad, no vectores de longitud 1.

i Escalares de verdad

Python no es un lenguaje vectorizado, a diferencia de R. Los valores enteros, flotantes, etc. son **escalares** de verdad, no vectores de longitud 1.

✓ Conversión entre tipos

Es posible convertir entre tipos de datos, siempre que tenga sentido.

```
float(12)
bool(0)
float("1.25")
int(12.5)
str(False)
```

Variables y asignaciones

Variables y asignaciones

Las variables se asignan con el símbolo = y la sintaxis es:

```
<nombre> = <expresión>
```

Variables y asignaciones

Las variables se asignan con el símbolo = y la sintaxis es:

```
<nombre> = <expresión>
```

Por ejemplo:

```
mensaje = "¡Hola, curso!"
```

Variables y asignaciones

Las variables se asignan con el símbolo = y la sintaxis es:

```
<nombre> = <expresión>
```

Por ejemplo:

```
mensaje = "¡Hola, curso!"
```

O:

```
a = 1
b = -5
c = 6
x1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
x2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
```

Asignacion múltiple

En vez de hacer esto:

```
a = 1  
b = -5  
c = 6
```

Asignacion múltiple

En vez de hacer esto:

```
a = 1  
b = -5  
c = 6
```

Python nos permite hacer esto:

```
a, b, c = 1, -5, 6
```

Asignacion múltiple

En vez de hacer esto:

```
a = 1  
b = -5  
c = 6
```

Python nos permite hacer esto:

```
a, b, c = 1, -5, 6
```



! La realidad siempre es más compleja

La asignación múltiple no es un simple atajo sintáctico, es consecuencia del **empaquetado** y **desempaquetado**, mecanismo que veremos más adelante.





Nombres permitidos

- Solo pueden contener **letras, números y guiones bajos**





Nombres permitidos

- Solo pueden contener **letras, números y guiones bajos**
- Deben comenzar con letra o guión bajo, pero no con un número.
 - mensaje_1 
 - 1_mensaje 

Nombres permitidos

- Solo pueden contener **letras, números y guiones bajos**
- Deben comenzar con letra o guión bajo, pero no con un número.
 - mensaje_1 
 - 1_mensaje 
- No pueden contener espacios
 - el_mensaje 
 - el mensaje 

Nombres permitidos

- Solo pueden contener **letras, números y guiones bajos**
- Deben comenzar con letra o guión bajo, pero no con un número.
 - mensaje_1 
 - 1_mensaje 
- No pueden contener espacios
 - el_mensaje 
 - el mensaje 
- No pueden ser palabras reservadas de Python

Nombres permitidos

i Consejos

- Usar nombres breves pero descriptivos
 - mensaje mejor que mj
 - equipo_favorito mejor que eq_fav
- No usar tildes ni caracteres específicos del castellano (ñ).

Expresiones

Expresiones

! ¿Qué es una expresión?

Una expresión es una combinación de **operadores** y **operandos** que se puede evaluar para producir un valor.

Expresiones

! ¿Qué es un operador?

Un símbolo o palabra que define una operación.

- Aritméticos: +, -, *, /, **, //, %.
- Lógicos: not, and, or.
- Comparación: is, is not, ==, !=, >, >=, <, <=.

Expresiones

! ¿Qué es un operador?

Un símbolo o palabra que define una operación.

- Aritméticos: +, -, *, /, **, //, %.
- Lógicos: not, and, or.
- Comparación: is, is not, ==, !=, >, >=, <, <=.

! ¿Y un operando?

Cada uno de los elementos sobre los que actúa el operador.

- Valores literales: 5, "posit", True, 1.8, etc.
- Variables: x, y, mi_objeto, etc.
- Llamadas a funciones: sum(x, 5), fun(a, b, c, d), etc.

Ejemplos

```
1 2 ** 8
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

Ejemplos

```
1 2 ** 8
2 False or True
3
4
5
6
```

Ejemplos

```
1 2 ** 8
2 False or True
3 int(((a + b) ** 2) / 1.8)
4
5
6
```

Ejemplos

```
1 2 ** 8
2 False or True
3 int(((a + b) ** 2) / 1.8)
4 12 + sum(x, y, z)
5
6
```

Ejemplos

```
1 2 ** 8
2 False or True
3 int(((a + b) ** 2) / 1.8)
4 12 + sum(x, y, z)
5 1
6
```

Ejemplos

```
1 2 ** 8
2 False or True
3 int(((a + b) ** 2) / 1.8)
4 12 + sum(x, y, z)
5 1
6 "foo"
```

Ejemplos

```
1 2 ** 8
2 False or True
3 int(((a + b) ** 2) / 1.8)
4 12 + sum(x, y, z)
5 1
6 "foo"
```

Comparación de objetos

Python permite comparar objetos utilizando dos operadores:

- `==`
- `is`

Comparación de objetos

Python permite comparar objetos utilizando dos operadores:

- `==`
- `is`

La diferencia entre ellos es que `==` compara en **valor**, mientras que `is` compara en **identidad**.

Comparación de objetos

Python permite comparar objetos utilizando dos operadores:

- `==`
- `is`

La diferencia entre ellos es que `==` compara en **valor**, mientras que `is` compara en **identidad**.

La identidad de un objeto se obtiene llamando a `id()`:

```
1 >>> id("bar")
2 139233362987312
```

Veremos ejemplos más adelante.

Operaciones con `str`

Hay objetos que admiten ser utilizados con ciertos operadores.

Por ejemplo, la suma (+) y la multiplicación (*) tienen sentido con las cadenas de caracteres.

Operaciones con `str`

Hay objetos que admiten ser utilizados con ciertos operadores.

Por ejemplo, la suma (+) y la multiplicación (*) tienen sentido con las cadenas de caracteres.

Con cadenas de texto, sumar es concatenar.

```
>>> "f" + "a"  
'fa'
```

Operaciones con str

Hay objetos que admiten ser utilizados con ciertos operadores.

Por ejemplo, la suma (+) y la multiplicación (*) tienen sentido con las cadenas de caracteres.

Con cadenas de texto, sumar es concatenar.

```
>>> "f" + "a"  
'fa'
```

Y donde vale sumar, vale que multiplicar por n es sumar el valor n veces:

```
>>> "z" * 3  
'zzz'
```

Métodos

Los objetos también pueden ofrecer acciones que no se implementan a través de operadores, sino de métodos.

Métodos

Los objetos también pueden ofrecer acciones que no se implementan a través de operadores, sino de métodos.

Por ahora, conformémonos con decir que un método es una función a la que se puede acceder a través de un objeto.

Métodos

Los objetos también pueden ofrecer acciones que no se implementan a través de operadores, sino de métodos.

Por ahora, conformémonos con decir que un método es una función a la que se puede acceder a través de un objeto.

Las cadenas de texto implementan una gran variedad de métodos, útiles para realizar diversas tareas.

Métodos de `str`

Cambiar la capitalización:

```
>>> texto = "Hola, qué tal?"
```

Métodos de `str`

Cambiar la capitalización:

```
>>> texto = "Hola, qué tal?"  
>>> texto.upper()  
'HOLA, QUÉ TAL?'
```

Métodos de `str`

Cambiar la capitalización:

```
>>> texto = "Hola, qué tal?"
>>> texto.upper()
'HOLA, QUÉ TAL?'
>>> texto.lower()
'hola, qué tal?'
```

Métodos de `str`

Cambiar la capitalización:

```
>>> texto = "Hola, qué tal?"
>>> texto.upper()
'HOLA, QUÉ TAL?'
>>> texto.lower()
'hola, qué tal?'
>>> texto.title()
'Hola, Qué Tal?'
```

Métodos de `str`

Cambiar la capitalización:

```
>>> texto = "Hola, qué tal?"
>>> texto.upper()
'HOLA, QUÉ TAL?'
>>> texto.lower()
'hola, qué tal?'
>>> texto.title()
'Hola, Qué Tal?'
>>> texto.capitalize()
'Hola, qué tal?'
```

Métodos de `str`

Eliminar espacios en blanco

```
>>> texto = "    algún texto    "
```

Métodos de `str`

Eliminar espacios en blanco

```
>>> texto = "    algún texto    "  
>>> texto.rstrip()  
'    algún texto'
```

Métodos de `str`

Eliminar espacios en blanco

```
>>> texto = "    algún texto    "  
>>> texto.rstrip()  
'    algún texto'  
>>> texto.lstrip()  
'algún texto    '
```

Métodos de `str`

Eliminar espacios en blanco

```
>>> texto = "    algún texto    "  
>>> texto.rstrip()  
'    algún texto'  
>>> texto.lstrip()  
'algún texto    '  
>>> texto.strip()  
'algún texto'
```

Métodos de `str`

Eliminar espacios en blanco

```
>>> texto = "    algún texto    "
>>> texto.rstrip()
'    algún texto'
>>> texto.lstrip()
'algún texto    '
>>> texto.strip()
'algún texto'
>>> texto.strip().capitalize() # Encadenamiento de métodos
'Algún texto'
```

f-strings

Las *f-strings* permiten **interpolar** cadenas de texto.

f-strings

Las *f-strings* permiten **interpolar** cadenas de texto.

Sin *f-strings*:

```
>>> nombre = "Tomás"
>>> edad = 30
>>> print("Soy " + nombre + " y tengo " + str(edad) + " años.")
Soy Tomás y tengo 30 años.
```

f-strings

Las *f-strings* permiten **interpolar** cadenas de texto.

Sin *f-strings*:

```
>>> nombre = "Tomás"
>>> edad = 30
>>> print("Soy " + nombre + " y tengo " + str(edad) + " años.")
Soy Tomás y tengo 30 años.
```

Con *f-strings*:

```
>>> nombre = "Tomás"
>>> edad = 30
>>> print(f"Soy {nombre} y tengo {edad} años.")
Soy Tomás y tengo 30 años.
```

Funciones

¿Por qué usamos funciones?

- Reutilización de código

¿Por qué usamos funciones?

- Reutilización de código
- Organización

¿Por qué usamos funciones?

- Reutilización de código
- Organización
- Modularidad

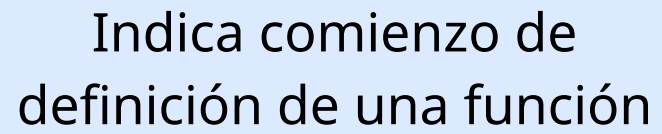
¿Por qué usamos funciones?

- Reutilización de código
- Organización
- Modularidad

Nada que ustedes no supieran ya 😊

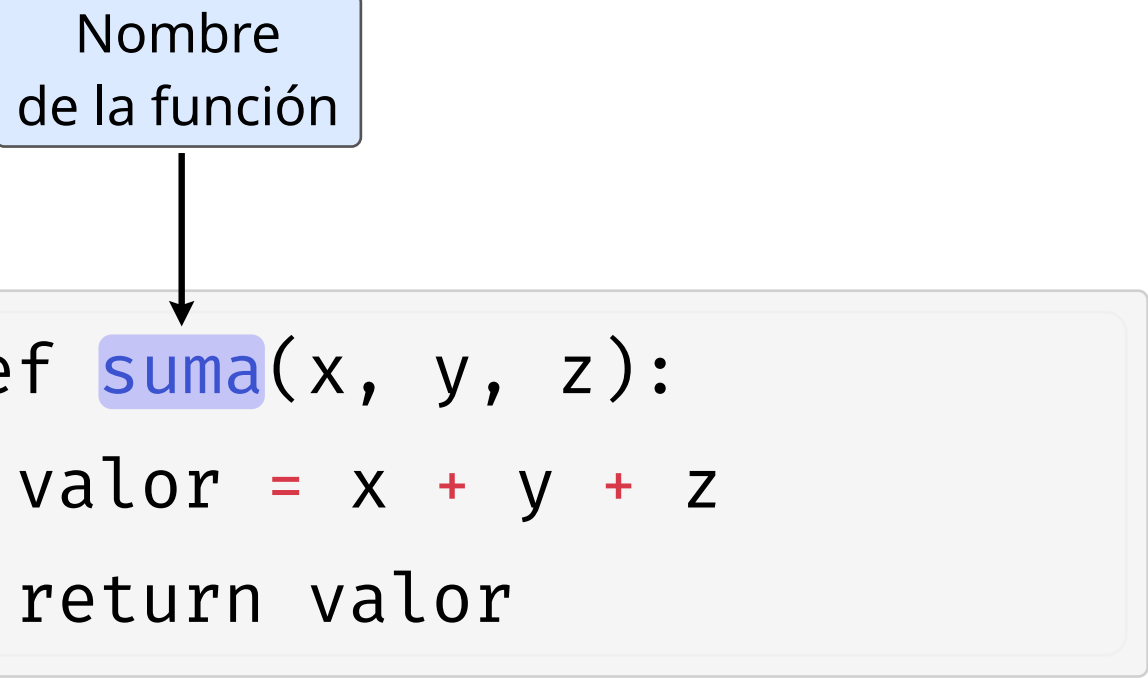
```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

Indica comienzo de
definición de una función



```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

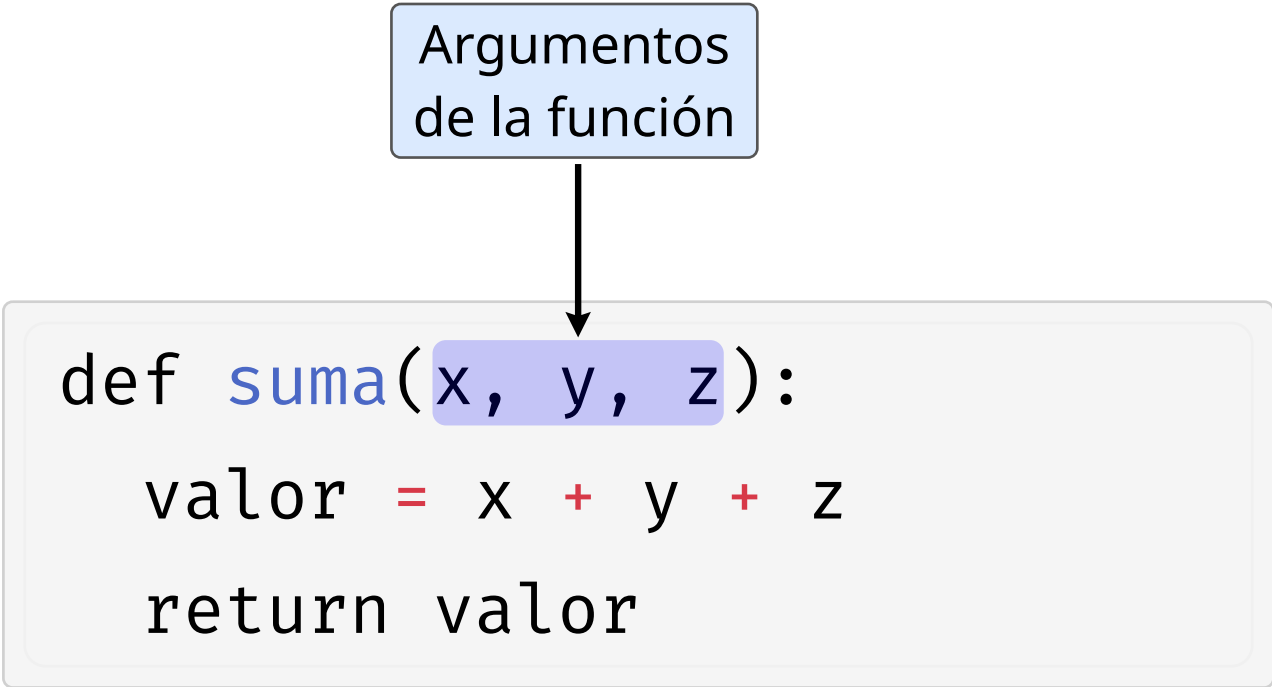
Nombre
de la función



```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

The diagram illustrates the relationship between a function name and its definition. A light blue box at the top contains the text "Nombre de la función". A black arrow points from this box down to a larger, light gray box containing Python code. In the code, the word "suma" is highlighted with a light blue background, indicating its role as the function name. The code defines a function named "suma" that takes three arguments (x, y, z), calculates their sum, and returns the result.

Argumentos
de la función



The diagram consists of a light blue box at the top containing the text 'Argumentos de la función'. A black arrow points downwards from this box to a light gray box containing Python code. In the code, the parameters 'x, y, z' in the function signature 'def suma(x, y, z):' are highlighted with a light purple background. The code also includes 'valor = x + y + z' and 'return valor'.

```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

Comienza un
bloque de código

```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

Código que se ejecuta
cuando se llama a la función

```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

Indica que se devuelve un valor
y el fin de la ejecución
de la función

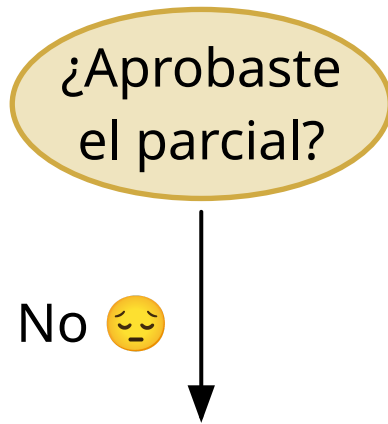
```
def suma(x, y, z):  
    valor = x + y + z  
    return valor
```

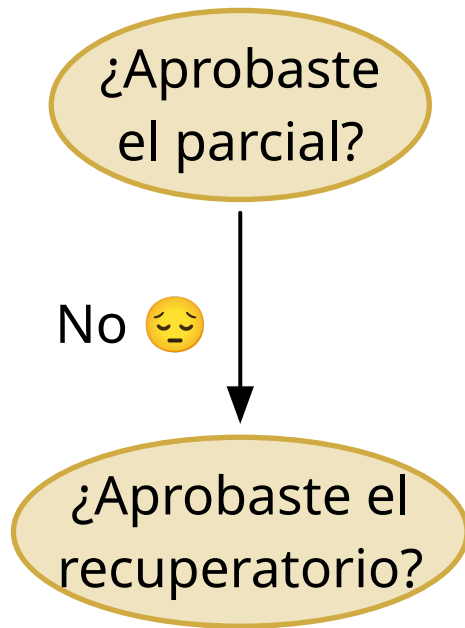
Valor que devuelve la función

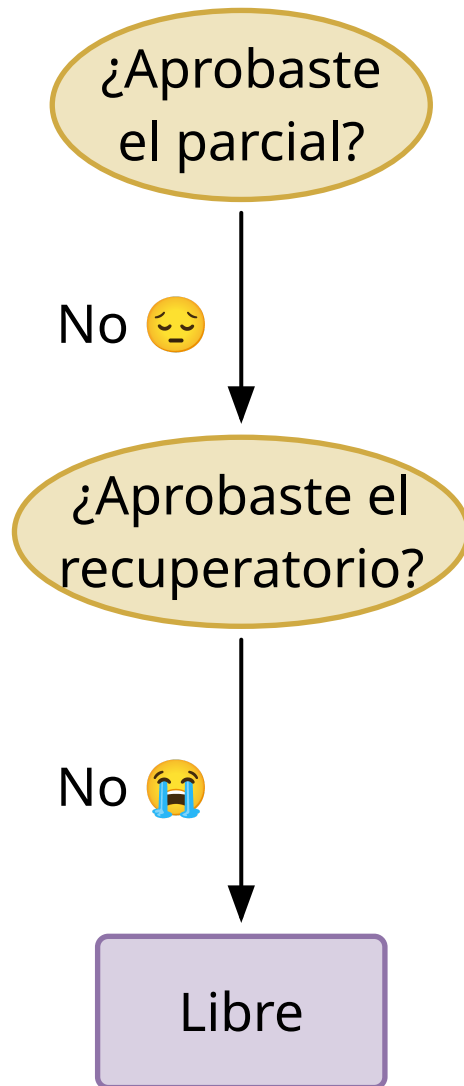
A laburar

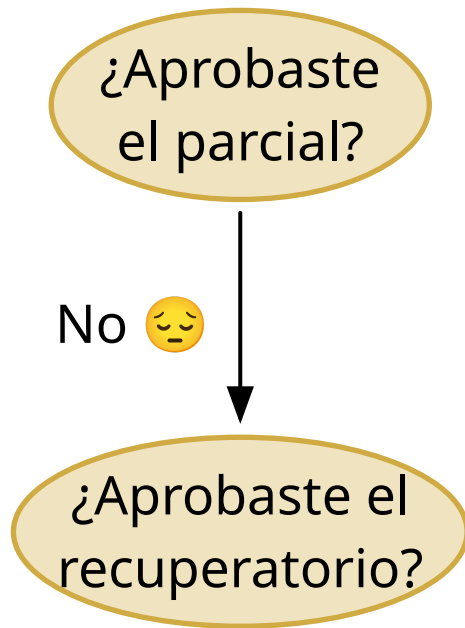
Ejecución condicional

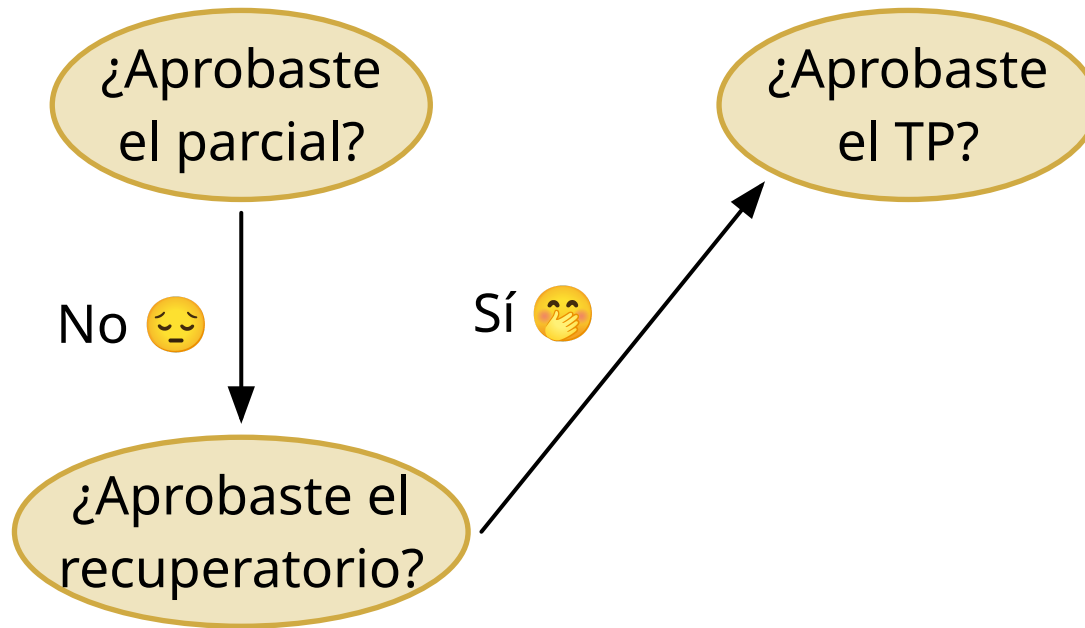
¿Aprobaste
el parcial?

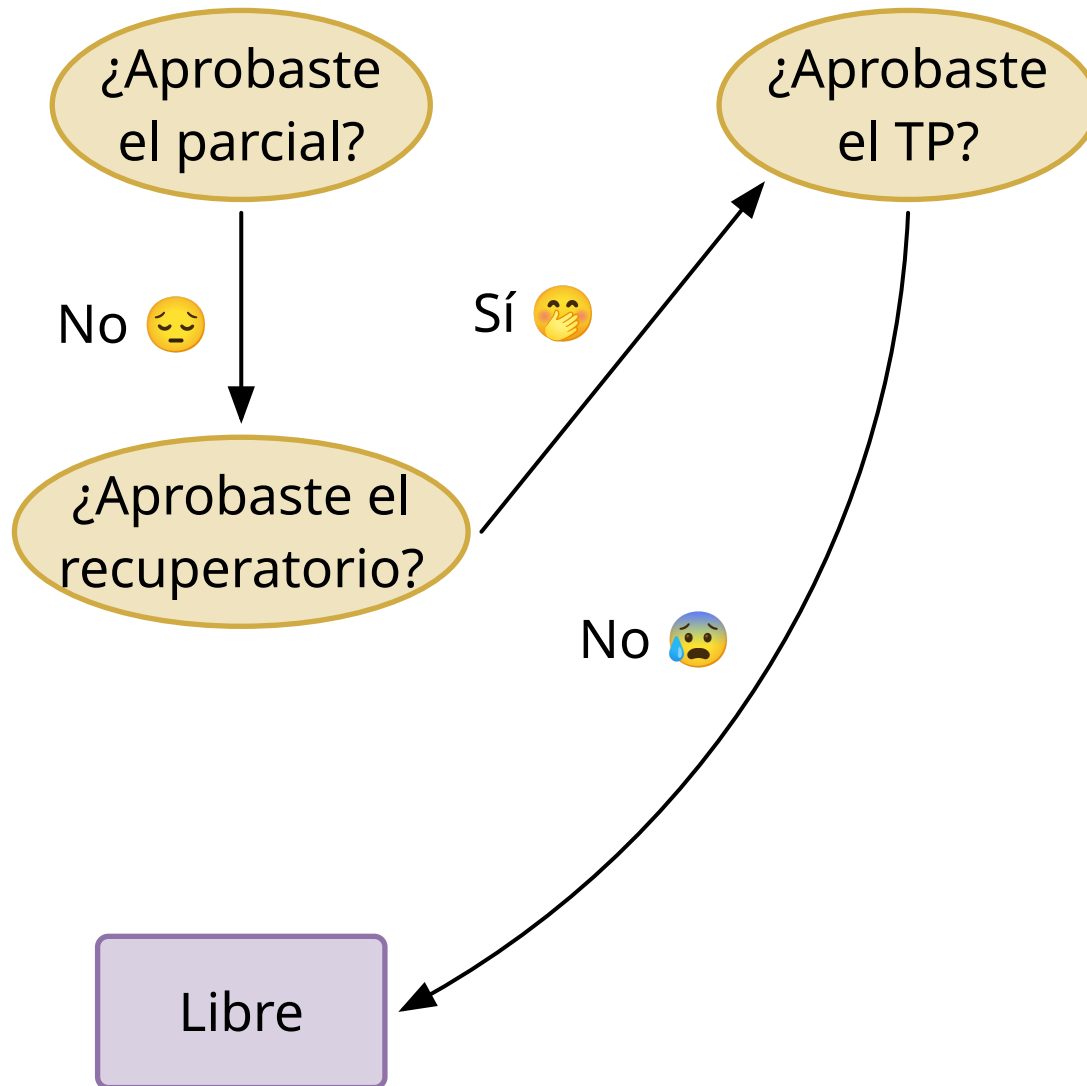


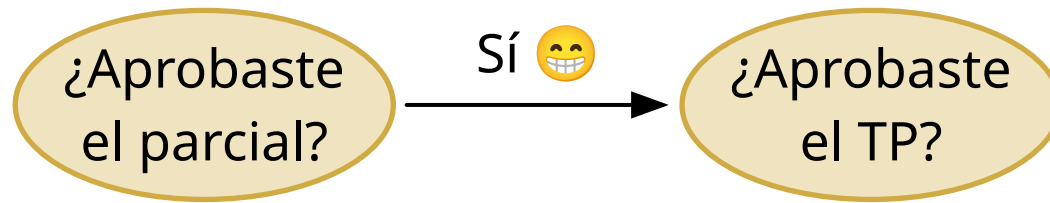


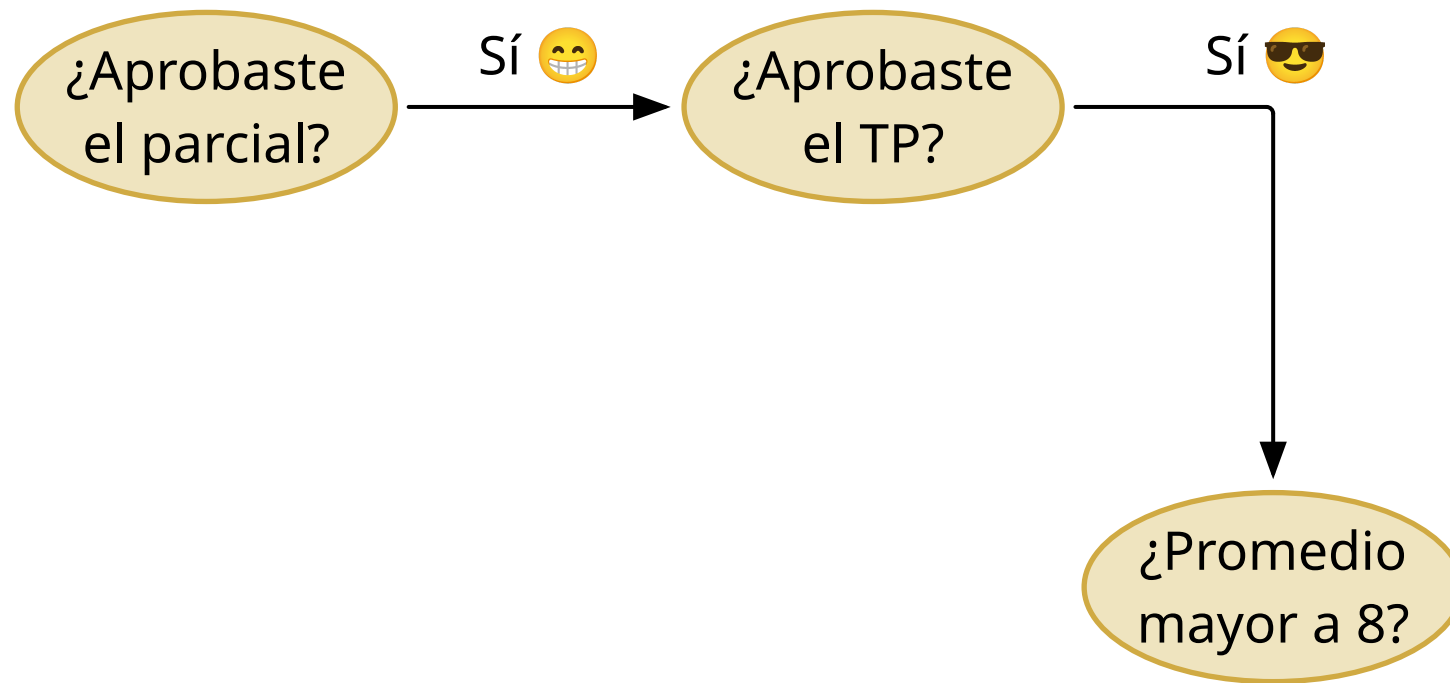


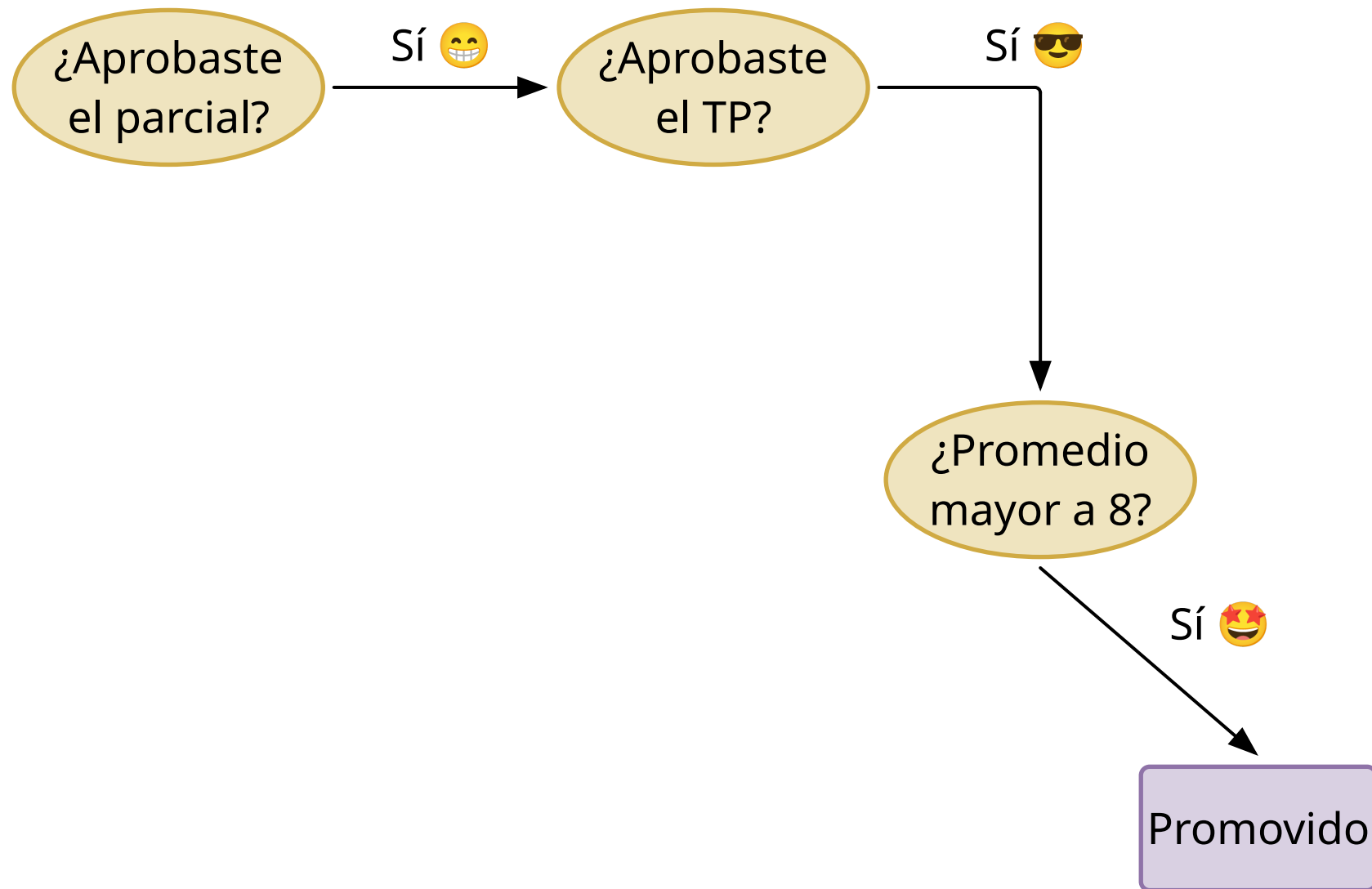


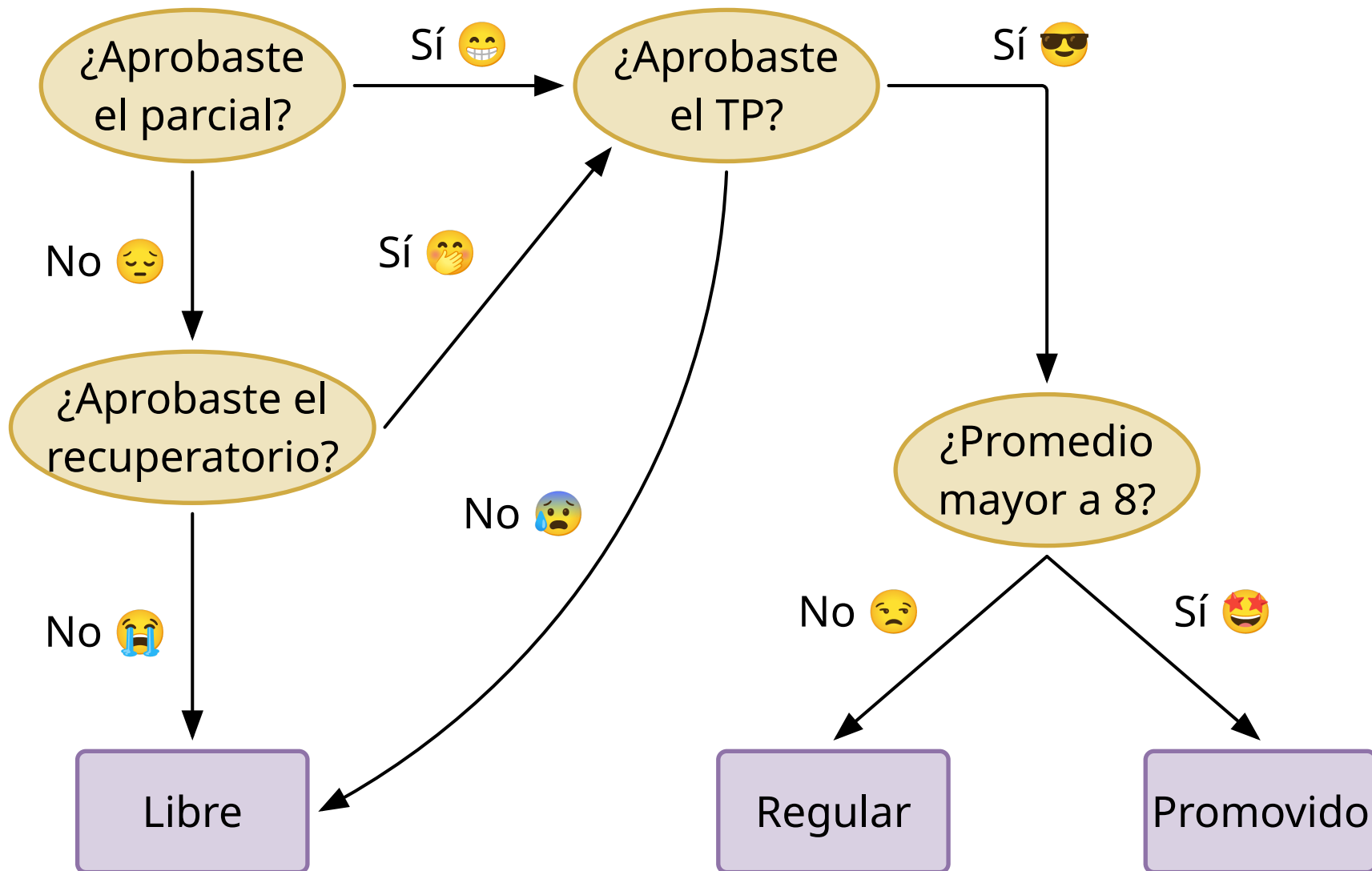












Ejecución condicional

- Realizar diferentes acciones según qué se responde a cada pregunta

Ejecución condicional

- Realizar diferentes acciones según qué se responde a cada pregunta
- Preguntas → Condiciones

Ejecución condicional


- Realizar diferentes acciones según qué se responde a cada pregunta
- Preguntas → Condiciones
- Posible gracias a las siguientes **estructuras de control**:
 - `if`
 - `if-else`
 - `if-elif-else`

Partes de estructura if

```
if nota >= 6:  
    print("Aprobado")
```

Partes de estructura if

Palabra clave if




```
if nota >= 6:  
    print("Aprobado")
```

The diagram illustrates the structure of an if statement in Python. A light blue box at the top contains the text "Palabra clave if". A black arrow points from this box down to the word "if" in a code snippet. The code snippet is displayed in a light gray rounded rectangle and consists of two lines: "if nota >= 6:" followed by an indented line "print("Aprobado")". The word "if" is highlighted with a light blue background, and the string "Aprobado" is highlighted in green.

Partes de estructura if

Expresión, llamada condición,
que al evaluarse resulta en True o False



```
if nota >= 6:  
    print("Aprobado")
```

Partes de estructura if

Comienza un
bloque de código

```
if nota >= 6:  
    print("Aprobado")
```

Partes de estructura if

```
if nota >= 6:  
    print("Aprobado")
```

Bloque que se ejecuta si
la condición es verdadera

Partes de estructura if-else

```
if nota >= 6:  
    print("Aprobado")  
else:  
    print("Desaprobado")
```

Partes de estructura if-else

Declara bloque que se ejecuta cuando la condición del if no es verdadera.
Palabra clave es else.

```
if nota >= 6:  
    print("Aprobado")  
else:  
    print("Desaprobado")
```

Partes de estructura if-else

```
if nota >= 6:  
    print("Aprobado")  
else:  
    print("Desaprobado")
```

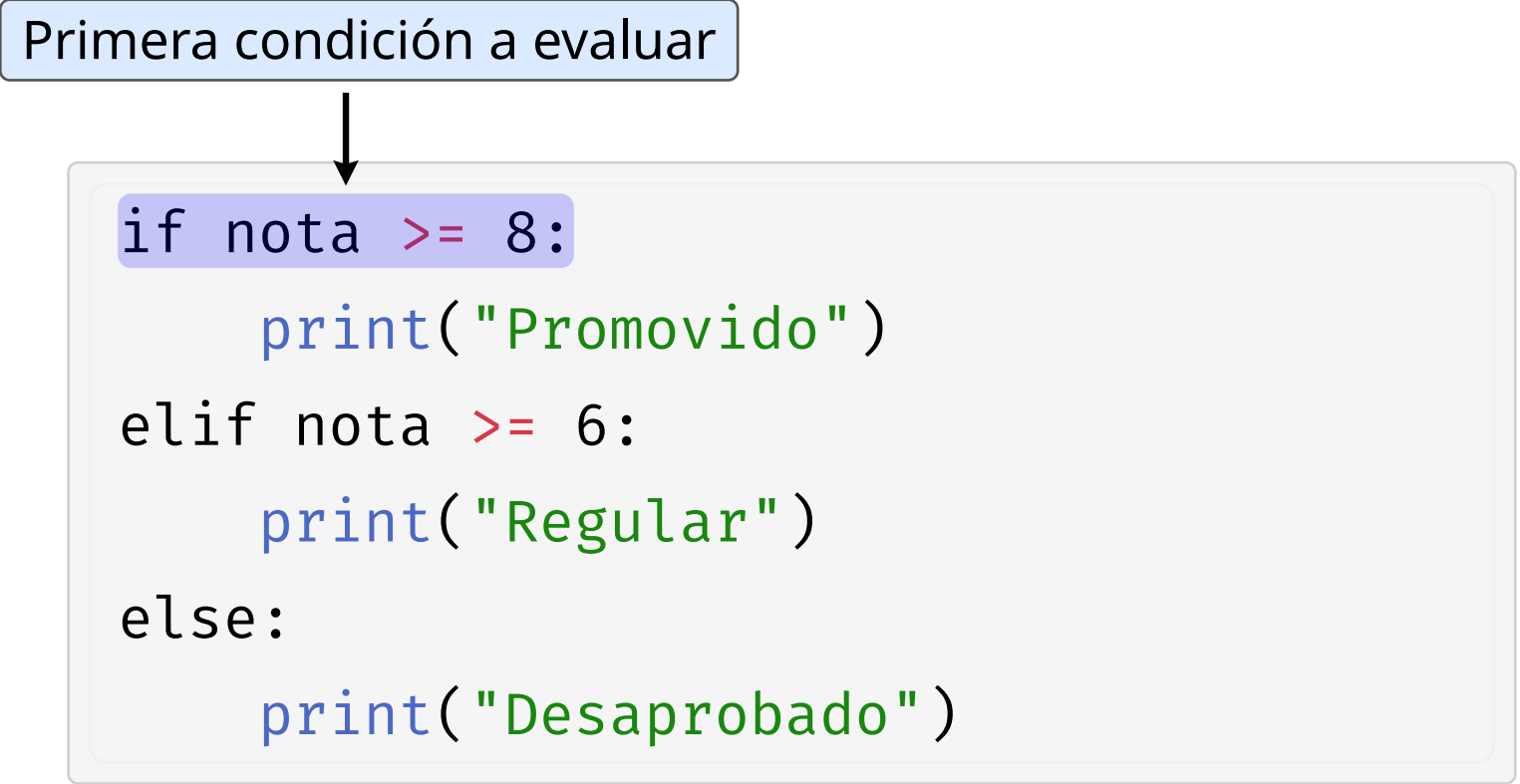
Bloque que se ejecuta
solo cuando
la condición del if es False

Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    print("Regular")  
else:  
    print("Desaprobado")
```

Partes de estructura if-elif-else

Primera condición a evaluar



The diagram illustrates the execution flow of an if-elif-else structure. A light blue box at the top contains the text "Primera condición a evaluar". A black arrow points downwards from this box to the first line of code in a larger, light gray box. The code is as follows:

```
if nota >= 8:
    print("Promovido")
elif nota >= 6:
    print("Regular")
else:
    print("Desaprobado")
```

Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    Bloque que se ejecuta si nota >= 8  
else:  
    print("Desaprobado")
```

Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    print("Regular")  
else:  
    print("Desaprobado")
```

Si la nota no es mayor o igual al 8, se evalúa si es mayor o igual a 6. La palabra clave es elif.

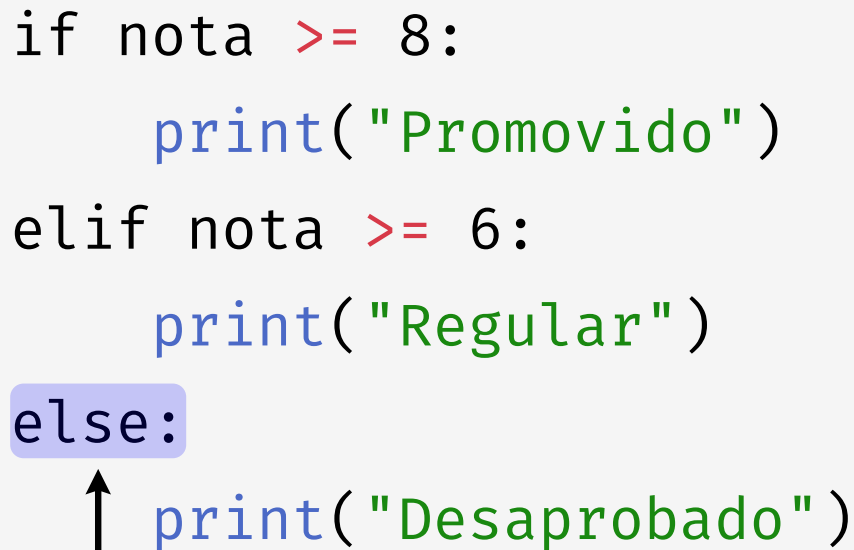
Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    print("Regular")  
else:  
    print("Desaprobado")
```

Bloque que se ejecuta si nota >= 6

Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    print("Regular")  
else:  
    print("Desaprobado")
```



Declara bloque alternativo.
Se ejecuta cuando nota no es mayor o igual a 8,
ni mayor o igual a 6.

Partes de estructura if-elif-else

```
if nota >= 8:  
    print("Promovido")  
elif nota >= 6:  
    print("Regular")  
else:  
    print("Desaprobado")
```

Código del bloque alternativo



A laburar

Bucles

Ejecución repetitiva

- Ejecutar la misma acción para cada valor en una secuencia.

Ejecución repetitiva

- Ejecutar la misma acción para cada valor en una secuencia.
- Ejecutar una acción hasta que se cumpla (o se deje de cumplir) una condición.

Ejecución repetitiva

- Ejecutar la misma acción para cada valor en una secuencia.
- Ejecutar una acción hasta que se cumpla (o se deje de cumplir) una condición.
- No hace falta copiar y pegar el mismo código todo el tiempo...

Ejecución repetitiva


- Ejecutar la misma acción para cada valor en una secuencia.
- Ejecutar una acción hasta que se cumpla (o se deje de cumplir) una condición.
- No hace falta copiar y pegar el mismo código todo el tiempo...
- gracias a las siguientes **estructuras de control**:
 - `for`
 - `while`

Partes de estructura for

```
for i in range(3):  
    print(f"El numero es {i}")
```

Partes de estructura for

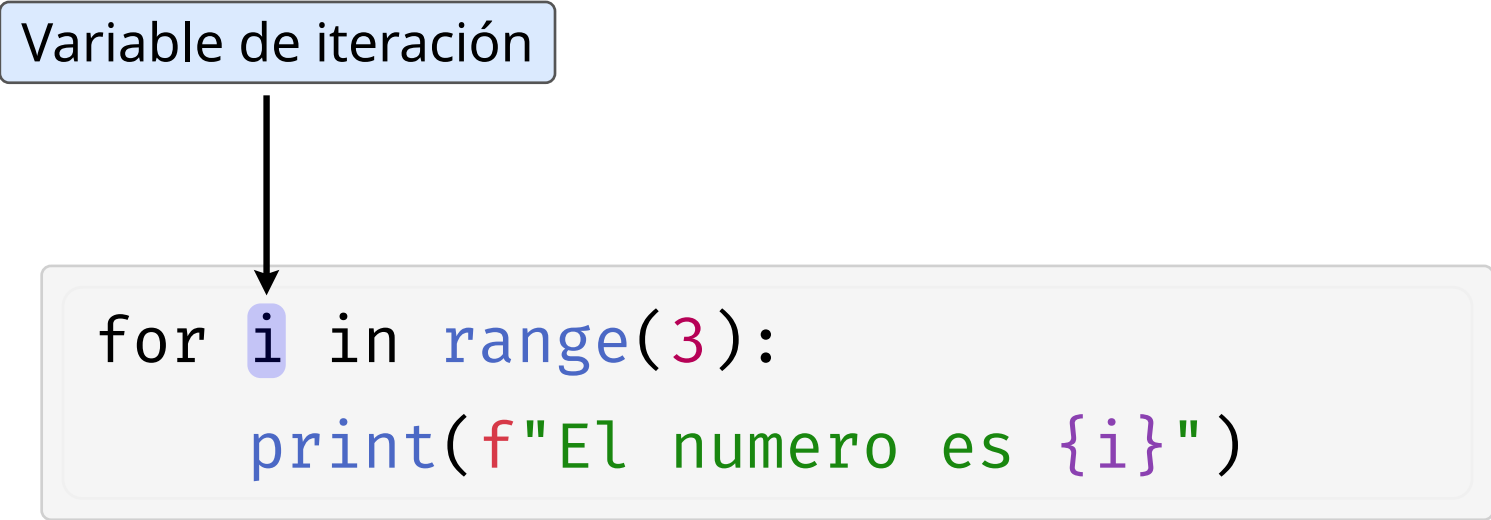
Palabra clave for

A diagram illustrating the structure of a Python 'for' loop. A light blue rounded rectangle at the top contains the text 'Palabra clave for'. A black arrow points from this box down to the first word 'for' in a code snippet. The code snippet is displayed in a light gray rounded rectangle and consists of two lines: 'for i in range(3):' and 'print(f"El numero es {i}")'. The word 'for' is highlighted with a light blue background, and the number '3' is highlighted with a light pink background.

```
for i in range(3):  
    print(f"El numero es {i}")
```

Partes de estructura for

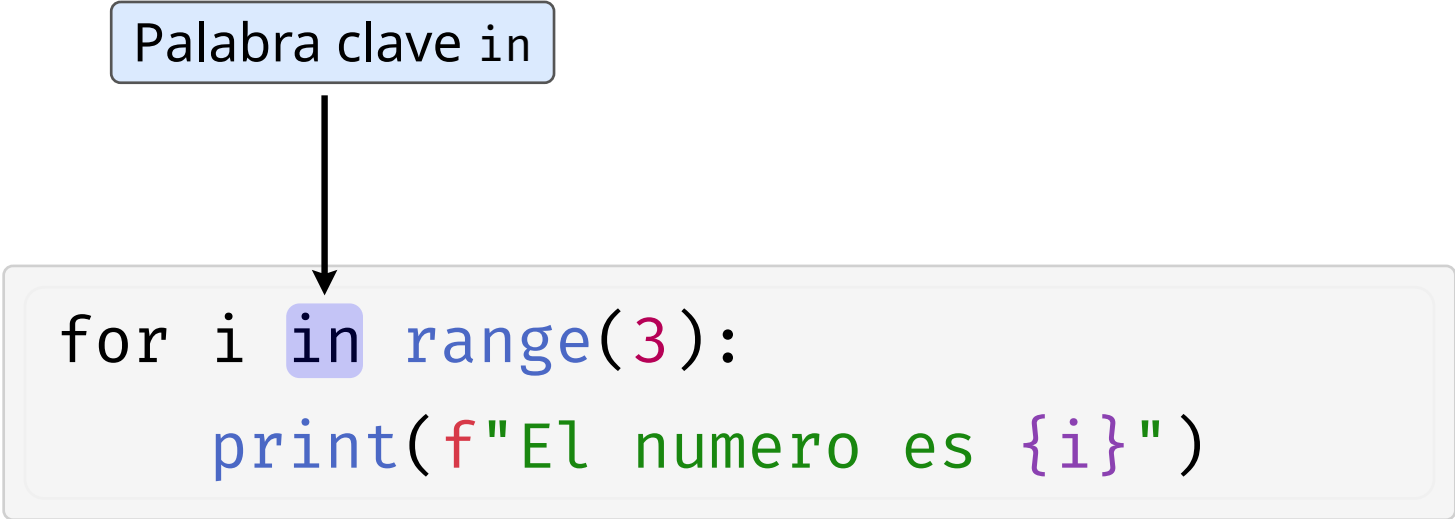
Variable de iteración



```
for i in range(3):  
    print(f"El numero es {i}")
```

Partes de estructura for

Palabra clave in




```
for i in range(3):  
    print(f"El numero es {i}")
```

The diagram illustrates the 'in' keyword in a Python for loop. A light blue box at the top contains the text 'Palabra clave in'. A black arrow points from this box down to the 'in' keyword in the code snippet below. The code snippet is enclosed in a light gray rounded rectangle and shows a for loop: 'for i in range(3):' followed by an indented 'print(f"El numero es {i}")'. The 'in' keyword is highlighted with a light blue background, matching the box above it.

Partes de estructura for

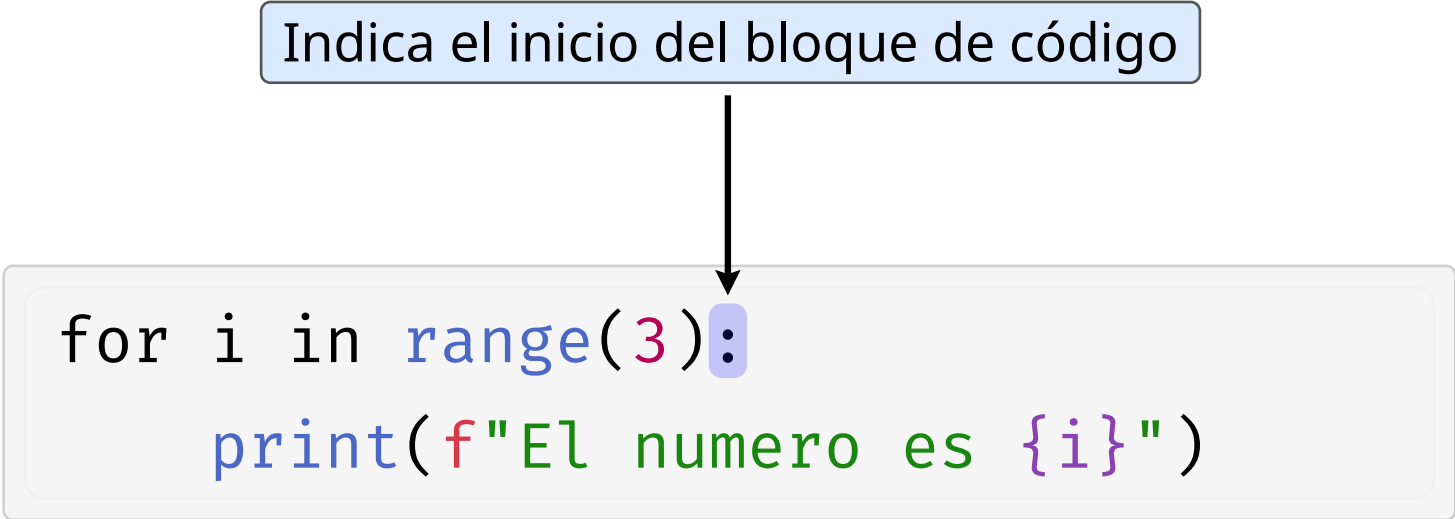
Objeto sobre el que se itera



```
for i in range(3):  
    print(f"El numero es {i}")
```

Partes de estructura for

Indica el inicio del bloque de código



```
for i in range(3):  
    print(f"El numero es {i}")
```

The diagram illustrates the structure of a Python for loop. A callout box at the top contains the text "Indica el inicio del bloque de código" (Indicates the start of the code block). A vertical arrow points from this box to the colon character at the end of the first line of the code snippet below. The code snippet is: `for i in range(3):` followed by an indented line `print(f"El numero es {i}")`. The colon in the first line is highlighted with a blue background, indicating its role as the start of the loop's body.

Partes de estructura for

```
for i in range(3):  
    print(f"El numero es {i}")
```

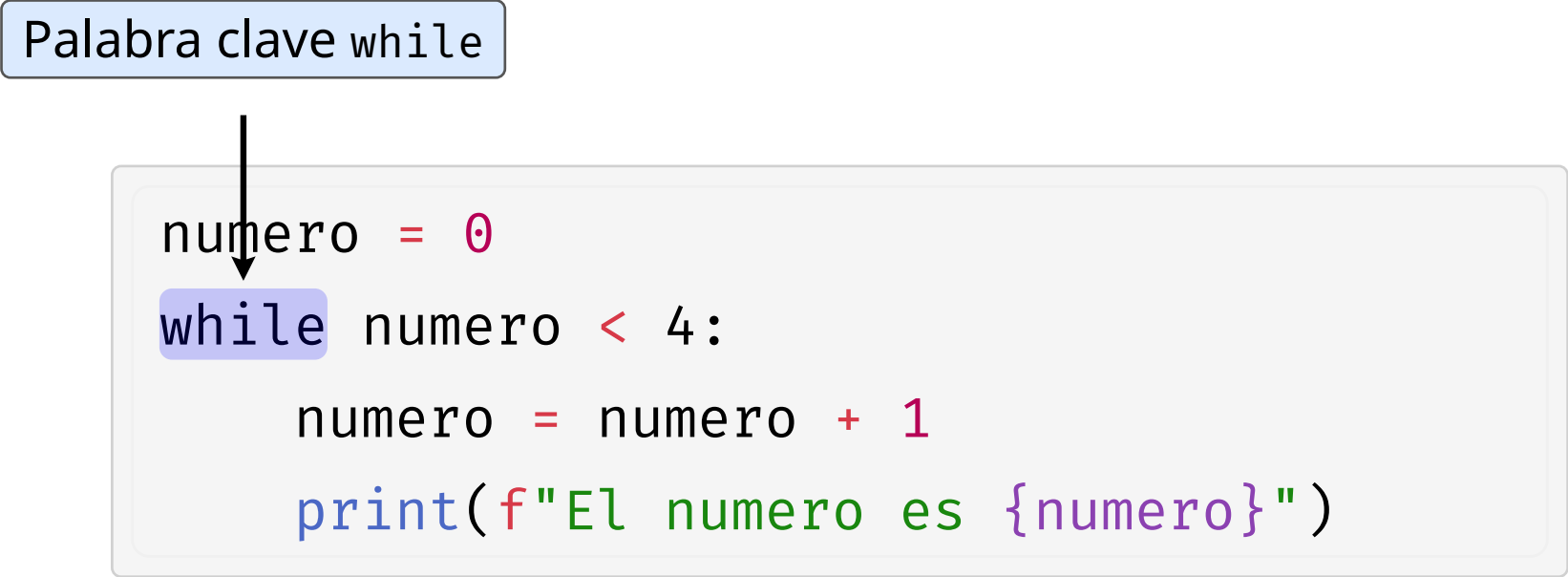
Bloque de código que se ejecuta en cada iteración

Partes de estructura while

```
numero = 0
while numero < 4:
    numero = numero + 1
    print(f"El numero es {numero}")
```

Partes de estructura while

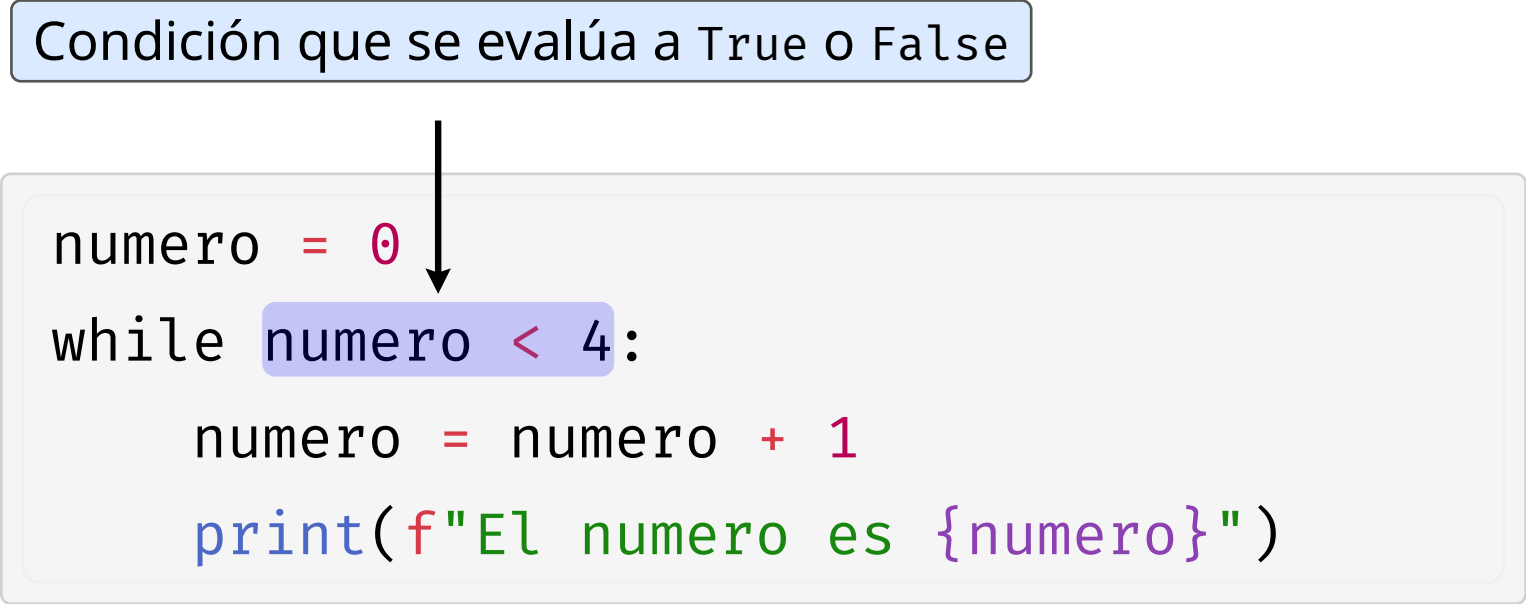
Palabra clave while



```
numero = 0
while numero < 4:
    numero = numero + 1
    print(f"El numero es {numero}")
```

Partes de estructura while

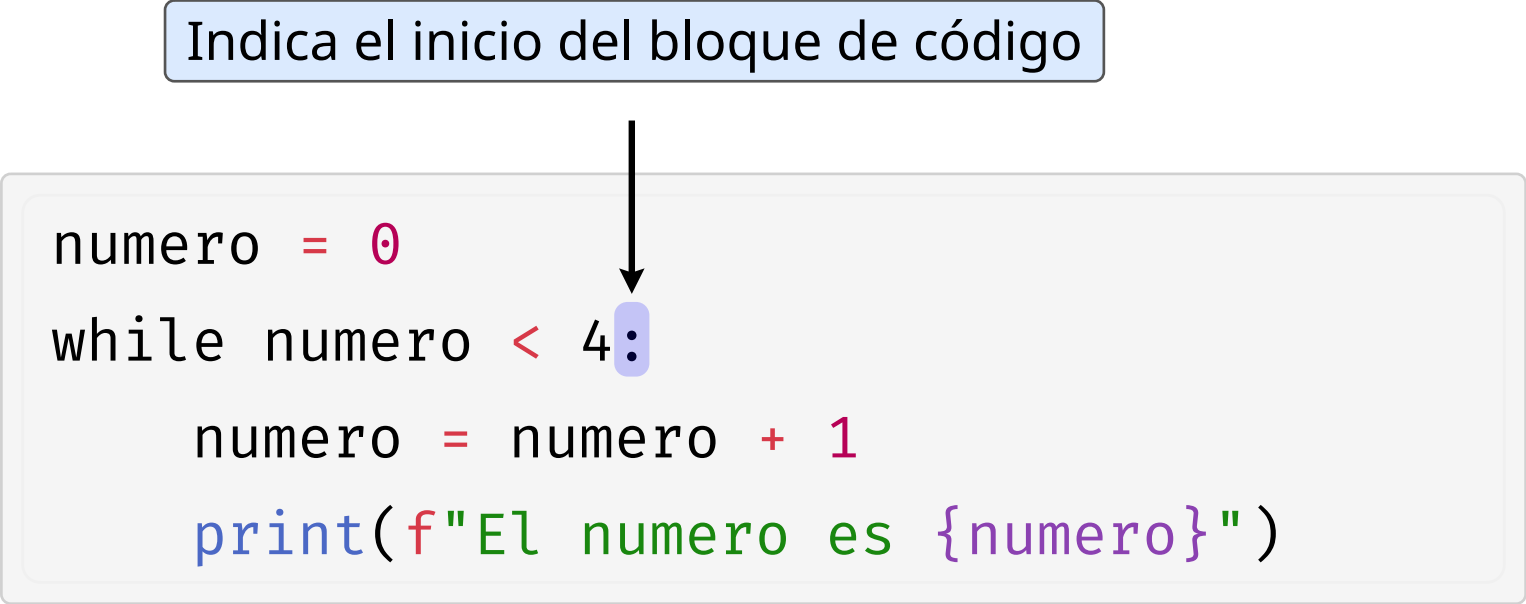
Condición que se evalúa a True o False



```
numero = 0
while numero < 4:
    numero = numero + 1
    print(f"El numero es {numero}")
```

Partes de estructura while

Indica el inicio del bloque de código



```
numero = 0
while numero < 4:
    numero = numero + 1
    print(f"El numero es {numero}")
```

Partes de estructura while

```
numero = 0
while numero < 4:
    numero = numero + 1
    print(f"El numero es {numero}")
```

Bloque de código que se ejecuta en cada iteración
mientras la condición sea verdadera

A laburar

Fin