

Bucle for

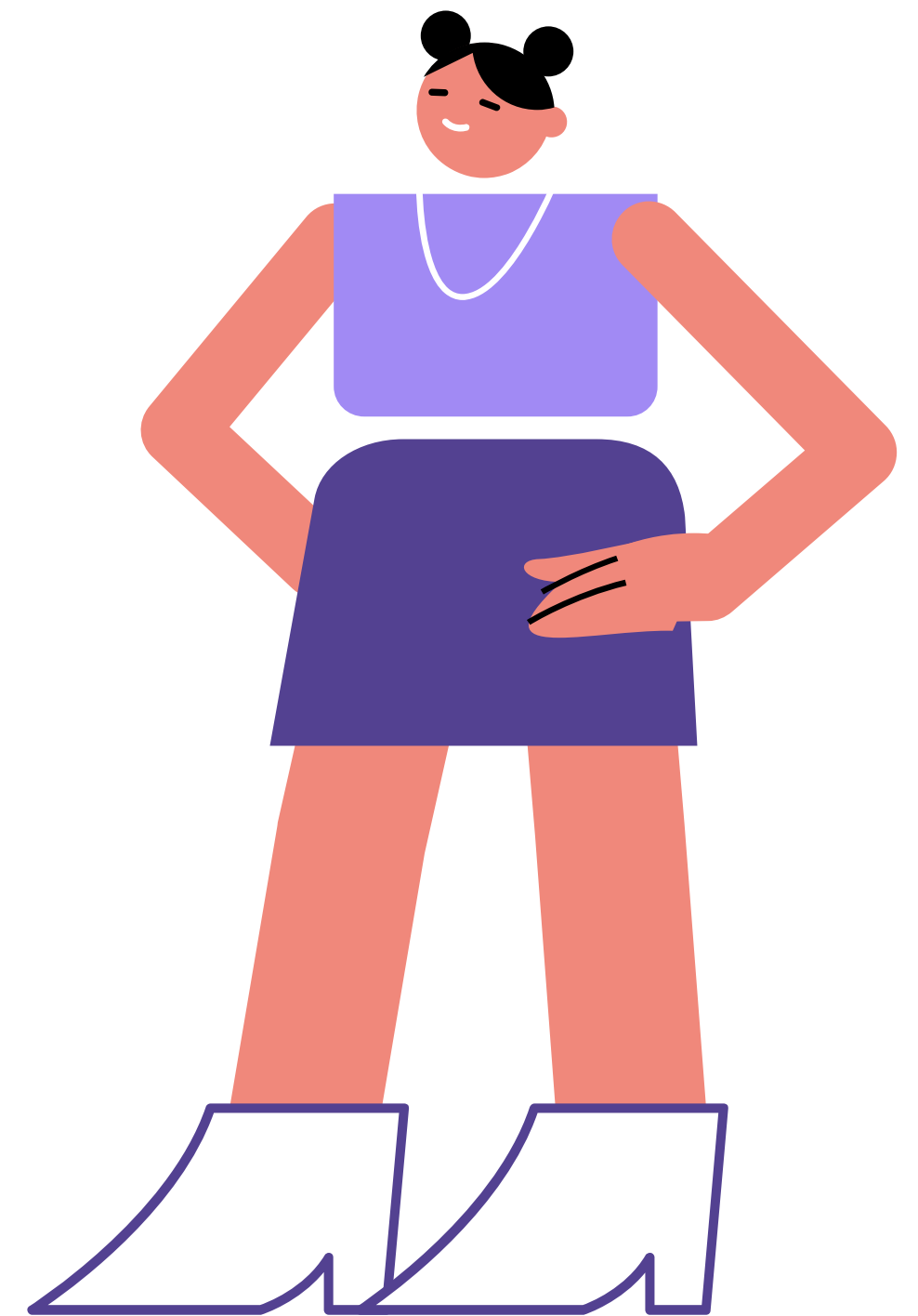
¿Para qué sirve el bucle for?

El bucle **for** nos permite ejecutar un bloque de código repetidamente. Por lo general, se utiliza para iterar sobre una secuencia de datos (slice, array, map o string).

Para crear bucles en Go solo existe la palabra reservada **for**, así podemos formar cuatro tipos de iteraciones:

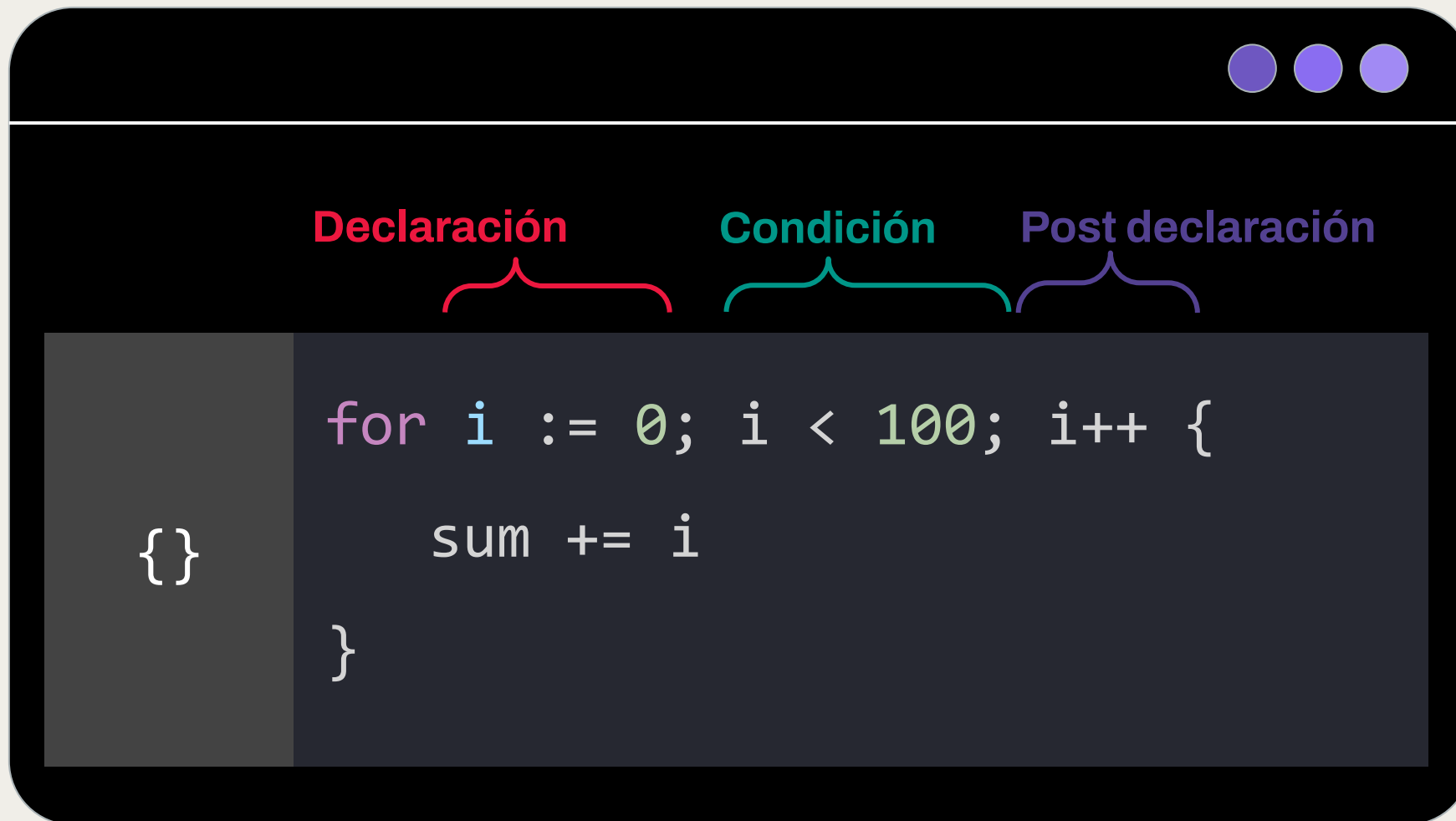
- 1 Standard for
- 2 Bucle while
- 3 Bucle infinito
- 4 Bucle range

Además, tenemos dos palabras reservadas más: **break** y **continue**.



1 Standard for

Esta es la estructura más común de un bucle **for**. Go tiene una sintaxis estándar compuesta por tres componentes y estos son: **declaración**, **condición** y **post declaración**.



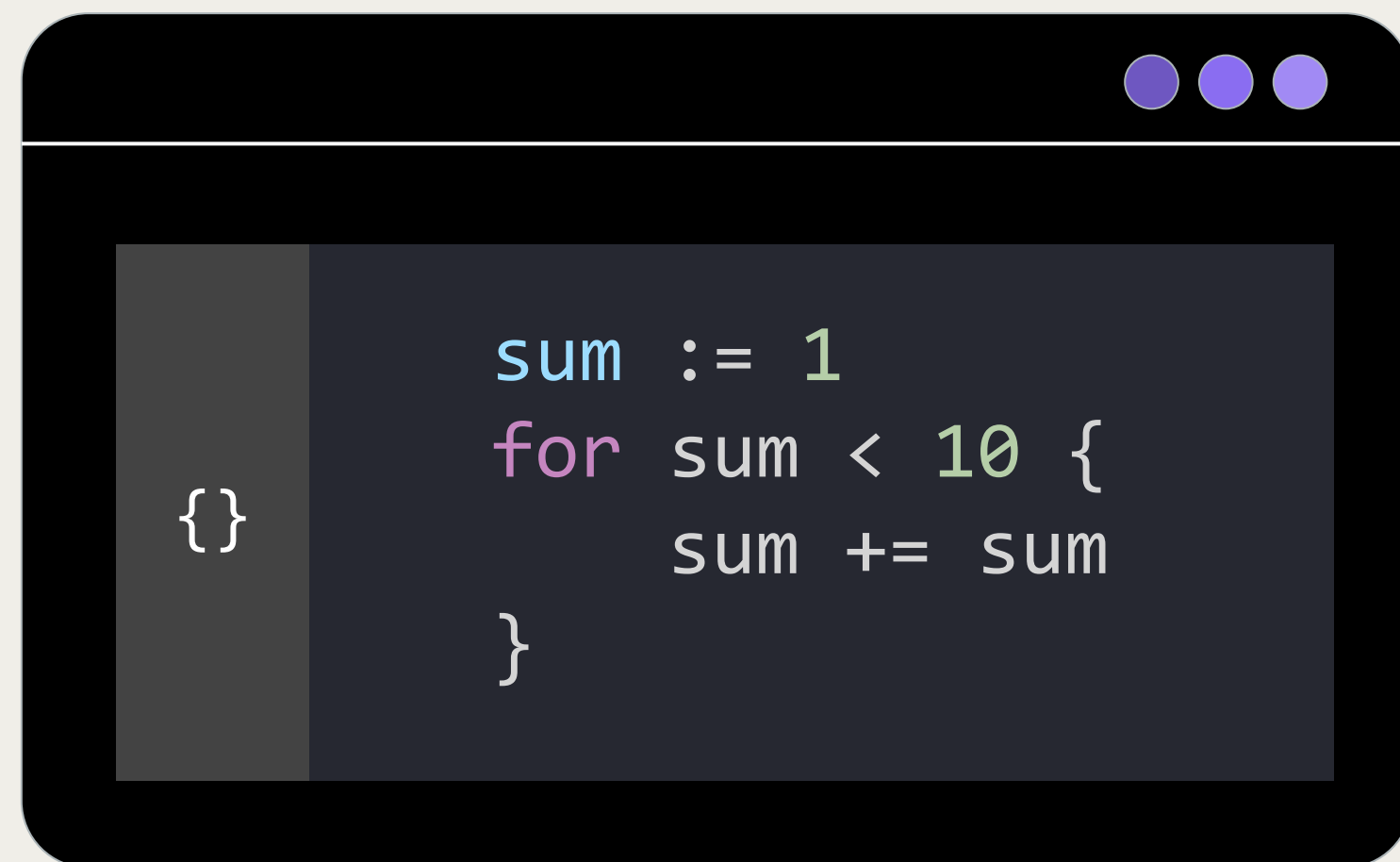
```
for i := 0; i < 100; i++ {  
    sum += i  
}
```

The diagram shows a Go `for` loop with three components labeled above it: **Declaración** (red), **Condición** (teal), and **Post declaración** (purple). The code is: `for i := 0; i < 100; i++ {` followed by an indented block `sum += i` and a closing brace `}`. A large curly brace `{ }` is shown to the left of the code block.

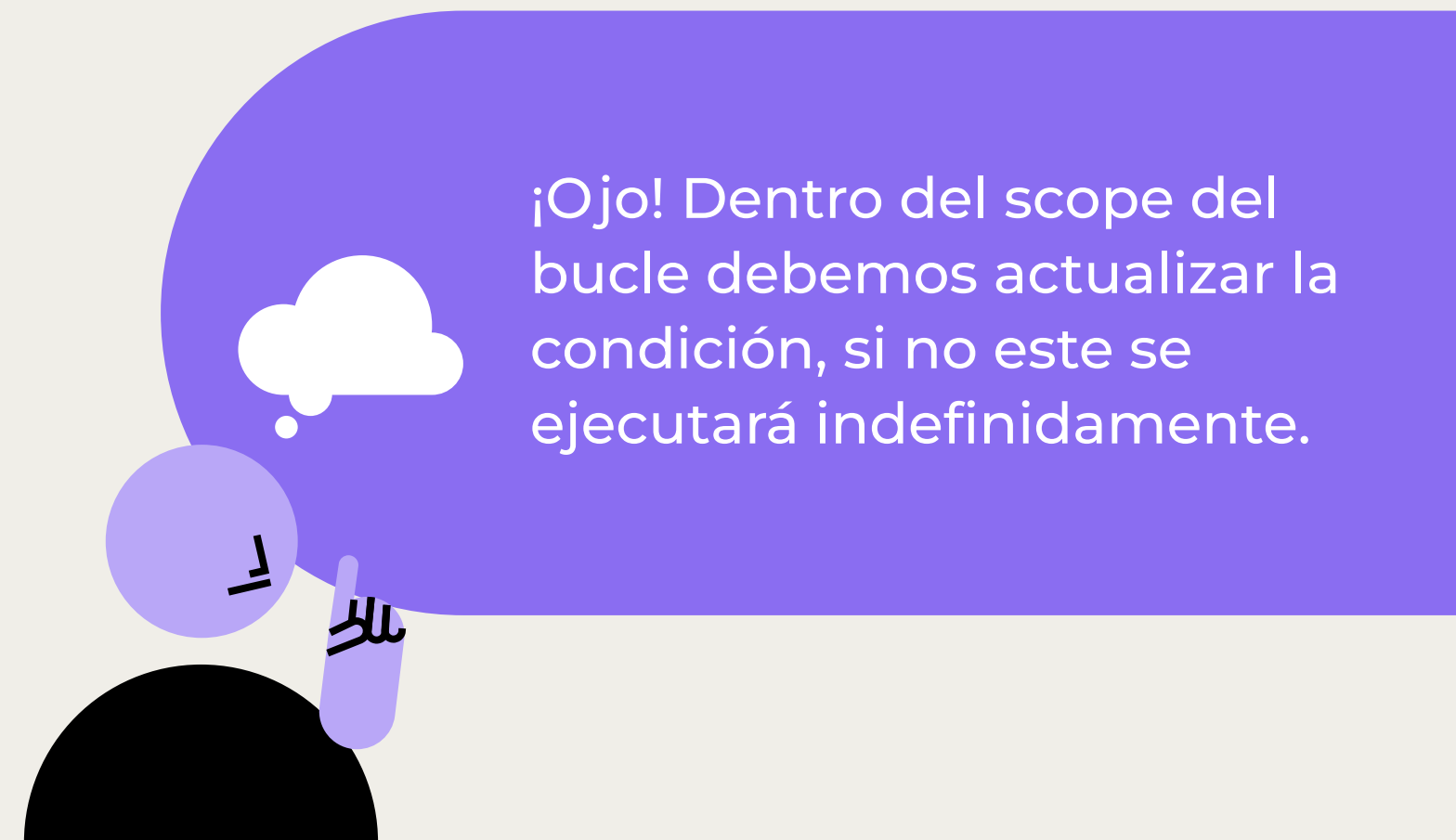
- **Declaración:** declara una variable y la expone dentro del scope del bucle.
- **Condición:** si la condición se cumple, ejecuta el código dentro del bucle; de lo contrario, termina.
- **Post declaración:** se ejecuta la post declaración, comúnmente se utiliza para modificar el valor de la variable declarada.

2 Bucle while

El bucle **while** nos permite ejecutar un bloque de código **mientras una condición se cumpla**. A diferencia del **standard for** de tres componentes, en este caso, solo tenemos la **condición**.



```
sum := 1
for sum < 10 {
    sum += sum
}
```



3 Bucle infinito

Para crear un **bucle infinito** basta con definir un **bucle while** con una condición que sea siempre verdadera. También, podemos obtener el mismo resultado si **no colocamos una condición**.

```
{}  
sum := 0  
for {  
    sum++  
}
```

=

```
{}  
sum := 0  
for true {  
    sum++  
}
```

4 Bucle range

La palabra reservada **range** itera por los elementos de estructuras de datos, retornando siempre **dos** variables. Tenemos:

- **Array or slice**: primero el **índice**, segundo el **elemento** de la lista.
- **String**: primero el **índice**, segundo es la representación del carácter en rune int.
- **Map**: primero la **key**, segundo el **value** del par clave-valor.
- **Channel**: primero el **elemento** del canal, segundo está vacío.

Veamos un ejemplo con un array:

```
{}  
frutas :=  
[]string{"manzana",  
"banana", "pera"}  
for i, fruta := range  
frutas {  
    fmt.Println(i, fruta)  
}
```

Saltar a la siguiente iteración

Puede ser útil pasar a la siguiente iteración de un bucle antes de que termine de correr todo el código. Esto se hace con la palabra reservada **continue**.

El siguiente ejemplo solo imprime los números impares. Cuando el resto de la división por 2 es 0, se trata de un número par. Por lo que entra en la condición y saltea la iteración.



```
for i := 0; i < 10; i++{  
    if i % 2 == 0 {  
        continue  
    }  
    fmt.Println(i, "es impar")  
}
```

Romper un bucle

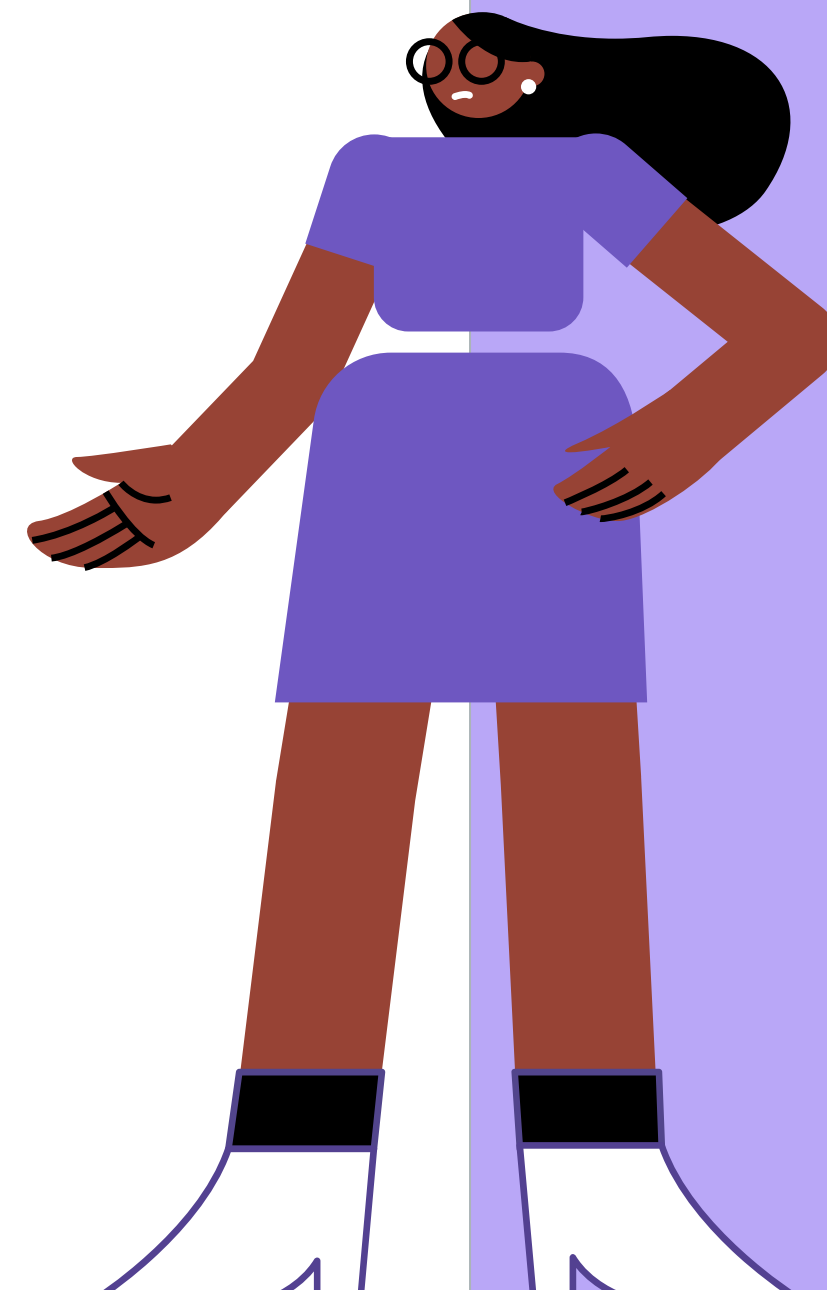
Romper un bucle antes de que termine puede ser útil, especialmente en un bucle infinito. La palabra reservada **break** nos permite terminar con la ejecución del bucle.



```
sum := 0
for {
    sum++
    if sum >= 1000 {
        break
    }
}
fmt.Println(sum)
//output: 1000
```


Conclusiones

La estructura de control **for** nos permite iterar ciertas instrucciones en nuestro programa. De esta forma, podemos recorrer distintas estructuras de datos y trabajar sobre ellas.



¡Muchas gracias!