

Loop for

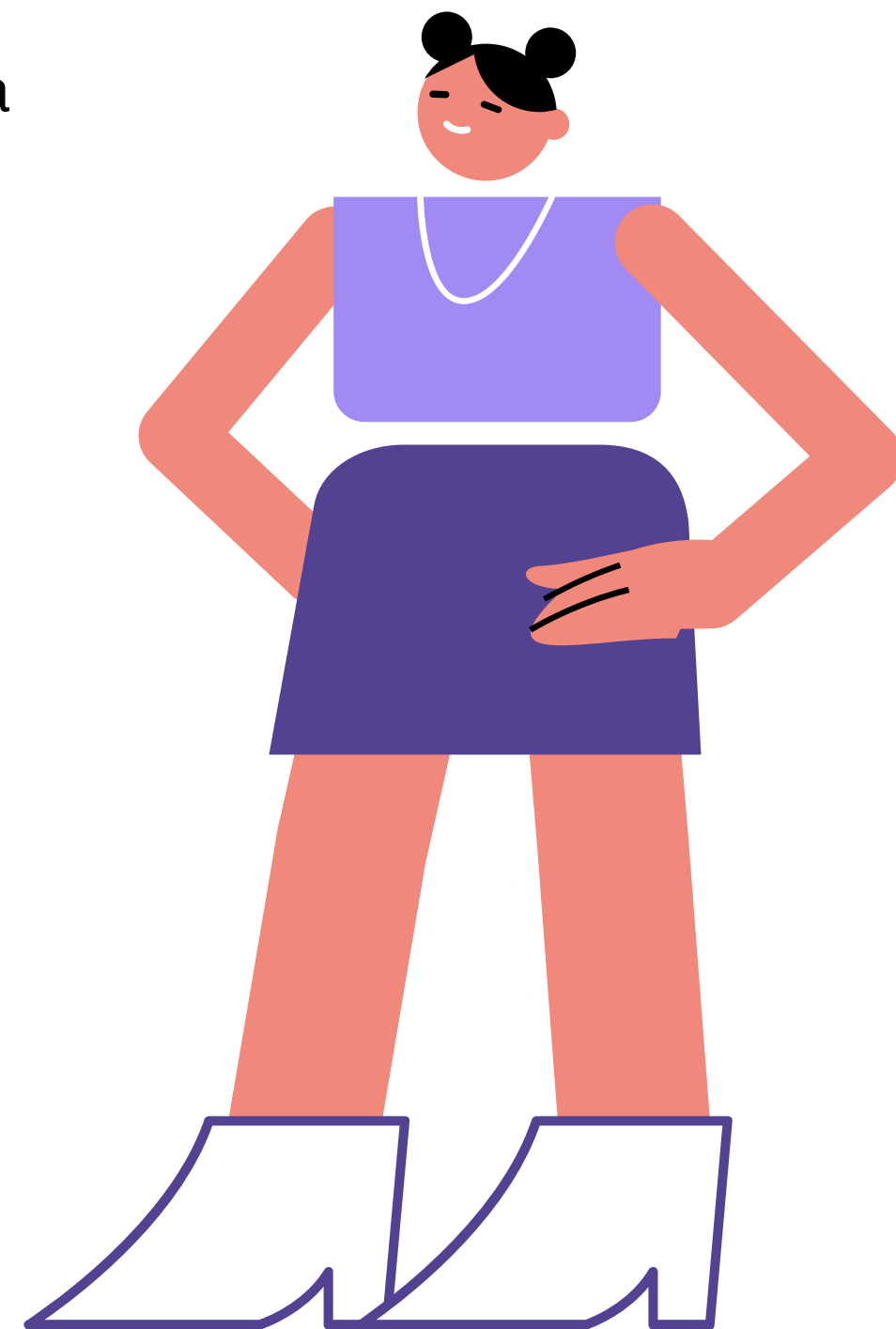
Para que serve o loop for?

O loop **for** nos permite executar um bloco de código repetidamente. Normalmente, é usado para iterar sobre uma sequência de dados (slice, array, map ou string).

Para criar loops em Go, existe somente a palavra reservada **for**. Assim, podemos formar quatro tipos de interações:

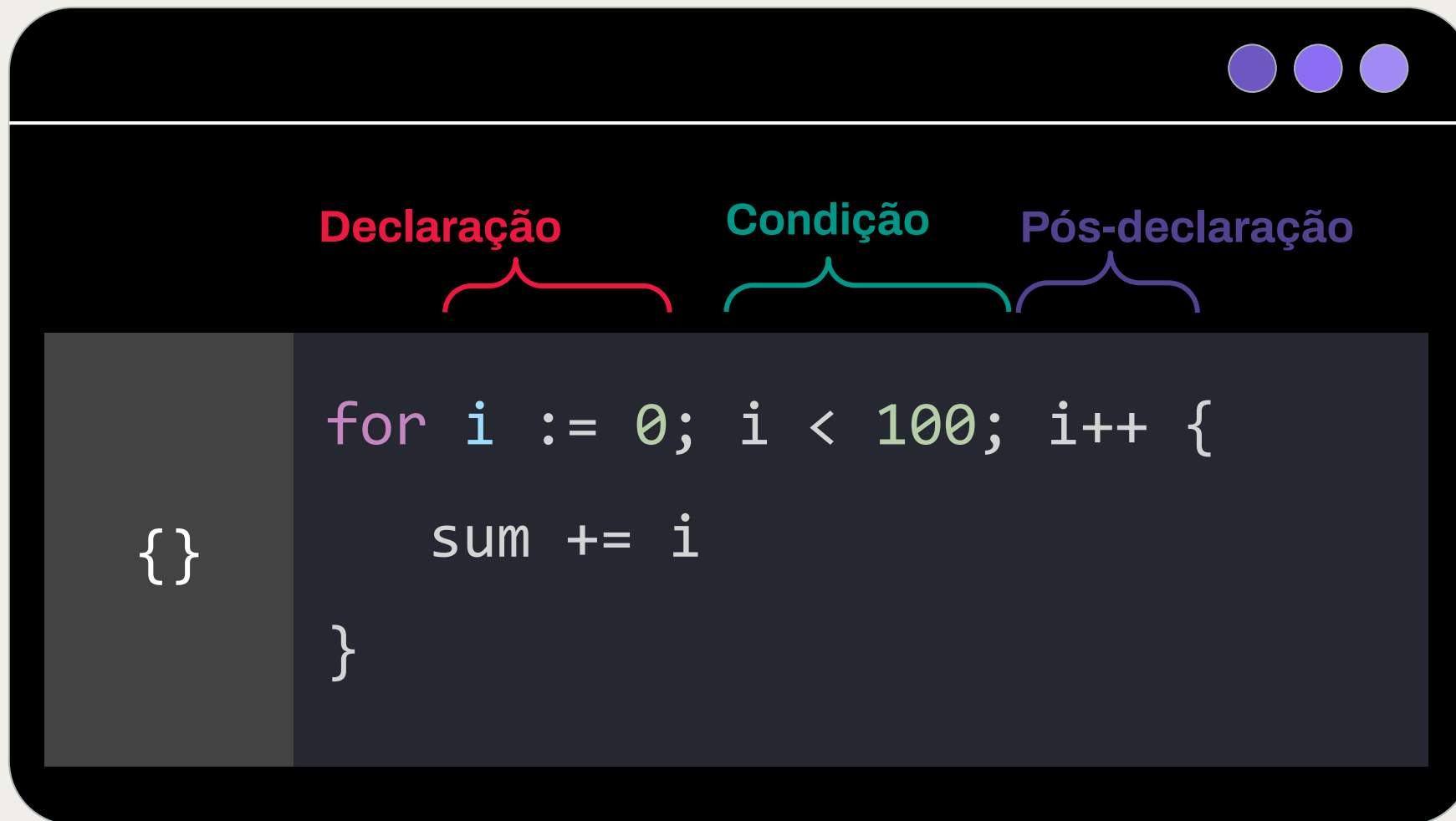
- 1 Standard for
- 2 Loop while
- 3 Infinity Loop
- 4 Loop range

Além disso, temos mais duas palavras reservadas: **break** e **continue**.



1 Standard for

É a estrutura mais comum de um loop **for**. Go tem uma sintaxe padrão formada por três componentes, que são: **declaração**, **condição** e **pós-declaração**.

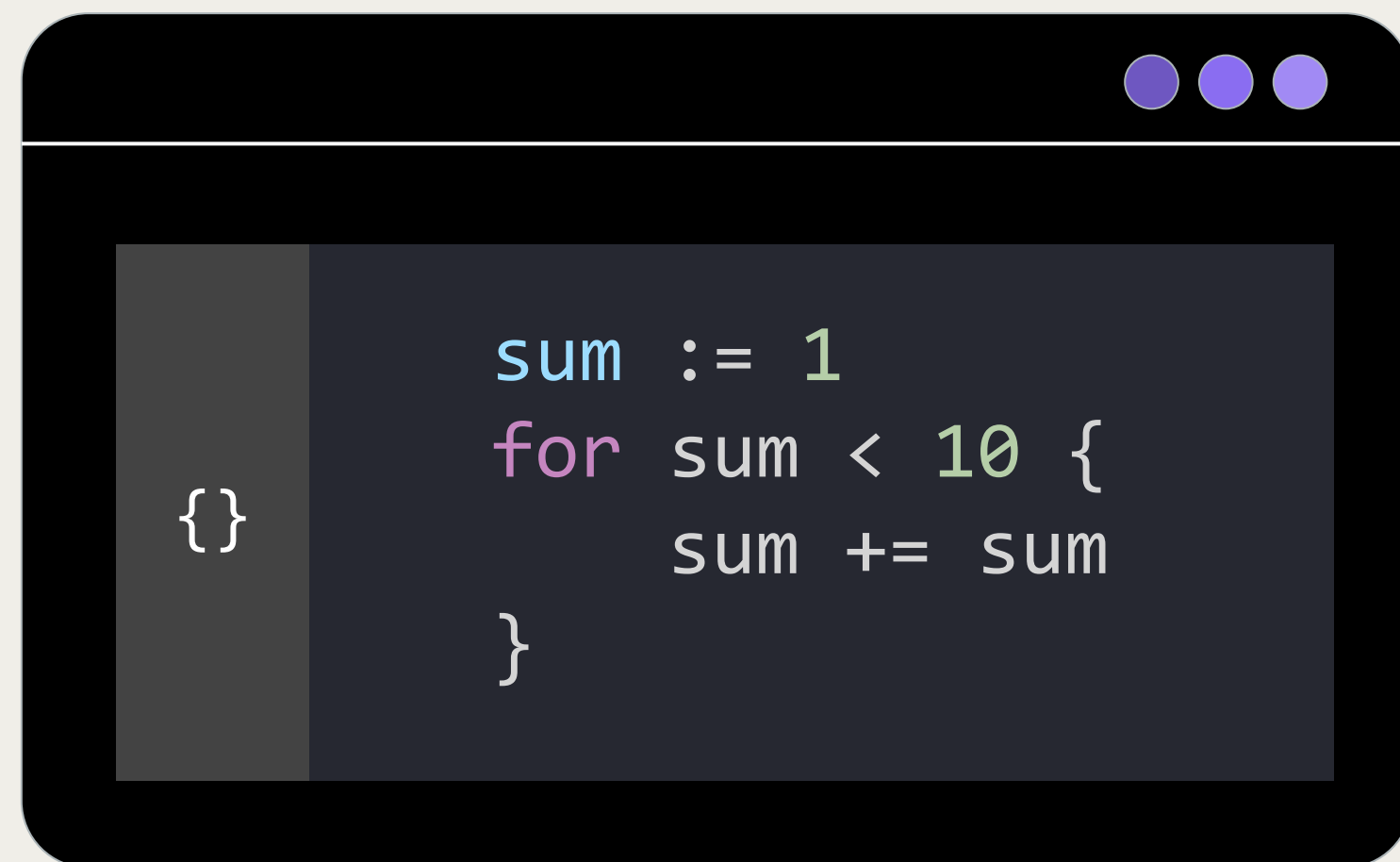


```
for i := 0; i < 100; i++ {  
    sum += i  
}
```

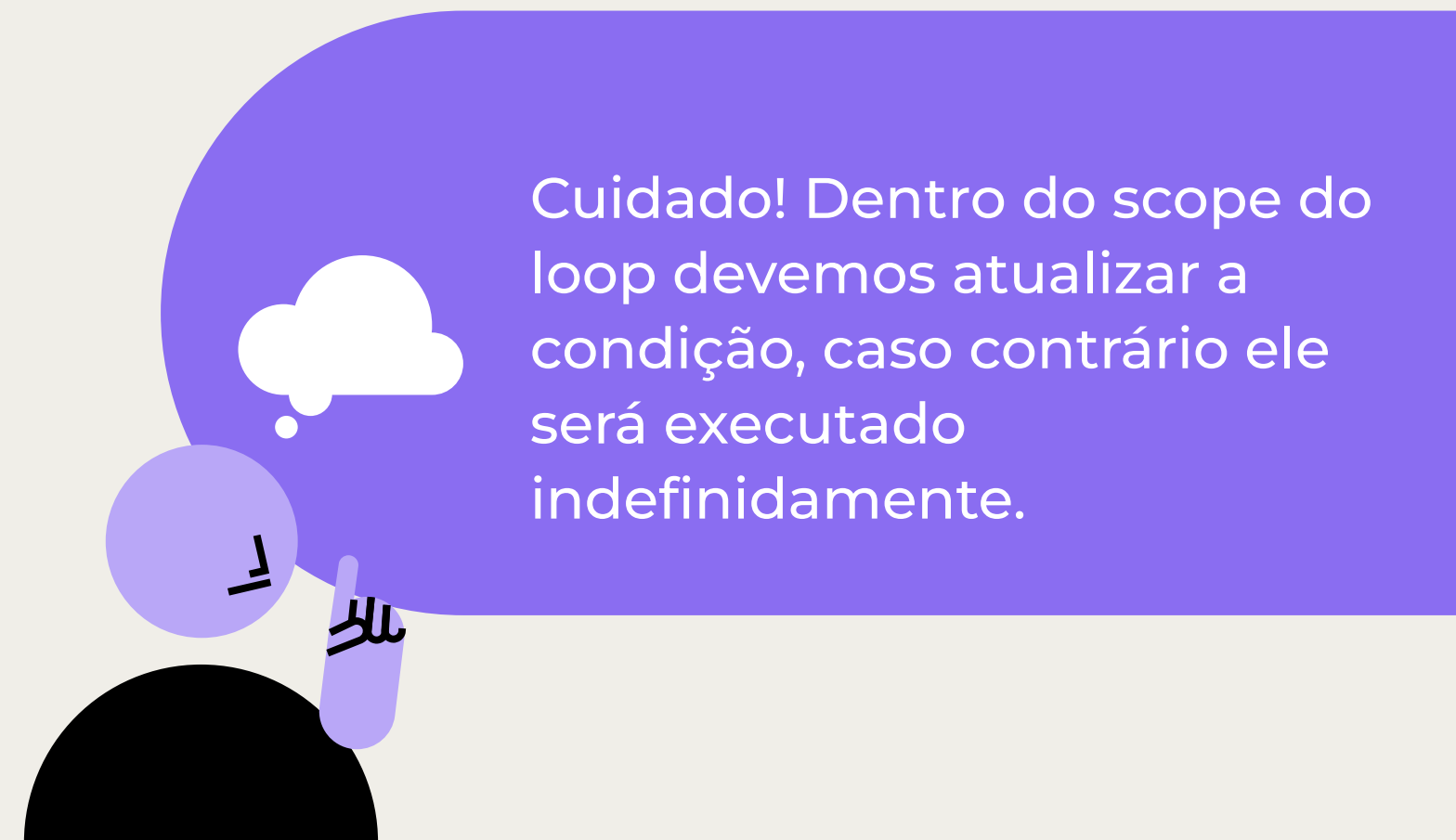
- **Declaração:** declara uma variável e expõe dentro do scope do loop.
- **Condição:** se a condição for cumprida, executa o código dentro do loop; caso contrário, finaliza.
- **Pós-declaração:** a pós-declaração é executada. Ela é normalmente utilizada para alterar o valor da variável declarada.

2 Loop while

O loop **while** nos permite executar um bloco de código desde que uma condição seja cumprida. Diferentemente do **standard for** de três componentes, neste caso, temos somente a **condição**.



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



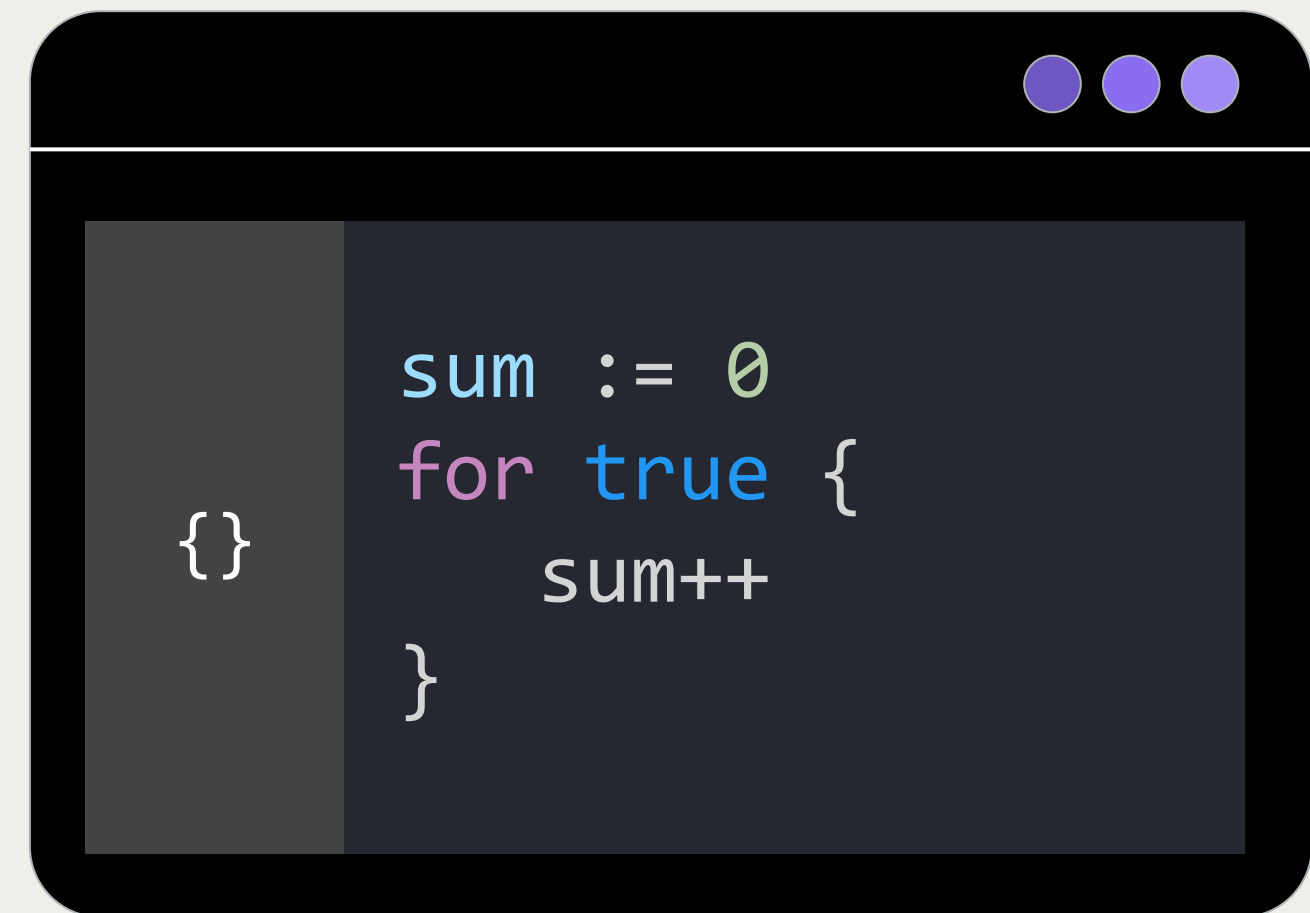
3 Loop infinito

Para criar **loop infinito** basta definir um **loop while** com uma condição que seja sempre verdadeira. Podemos obter o mesmo resultado **mesmo não colocando uma condição**.



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

=




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

4 Loop range

A palavra reservada **range** itera entre os elementos de estruturas de dados, retornando sempre **duas** variáveis.

- **Array or slice**: primeiro o **índice**, segundo o **elemento** da lista.
- **String**: primeiro o **índice**, segundo a representação do carácter em rune int.
- **Map**: primeiro a **key**, segundo o **value** do par chave-valor.
- **Channel**: primeiro o **elemento** do canal, segundo vazio.

Vamos ver um exemplo com um array:



```
{}  
fruits := []string{"maça",  
"banana", "pera"}  
for i, fruit := range fruits  
{  
    fmt.Println(i, fruit)  
}
```

Pular para a próxima iteração

Pode ser útil passar para a seguinte iteração de um loop antes do código terminar de rodar. Isto é feito com a palavra reservada **continue**.

O exemplo a seguir imprime apenas os números ímpares. Quando o resto da divisão por 2 for 0, se trata de um número par e assim a iteração atual será pulada.



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

Parar um loop

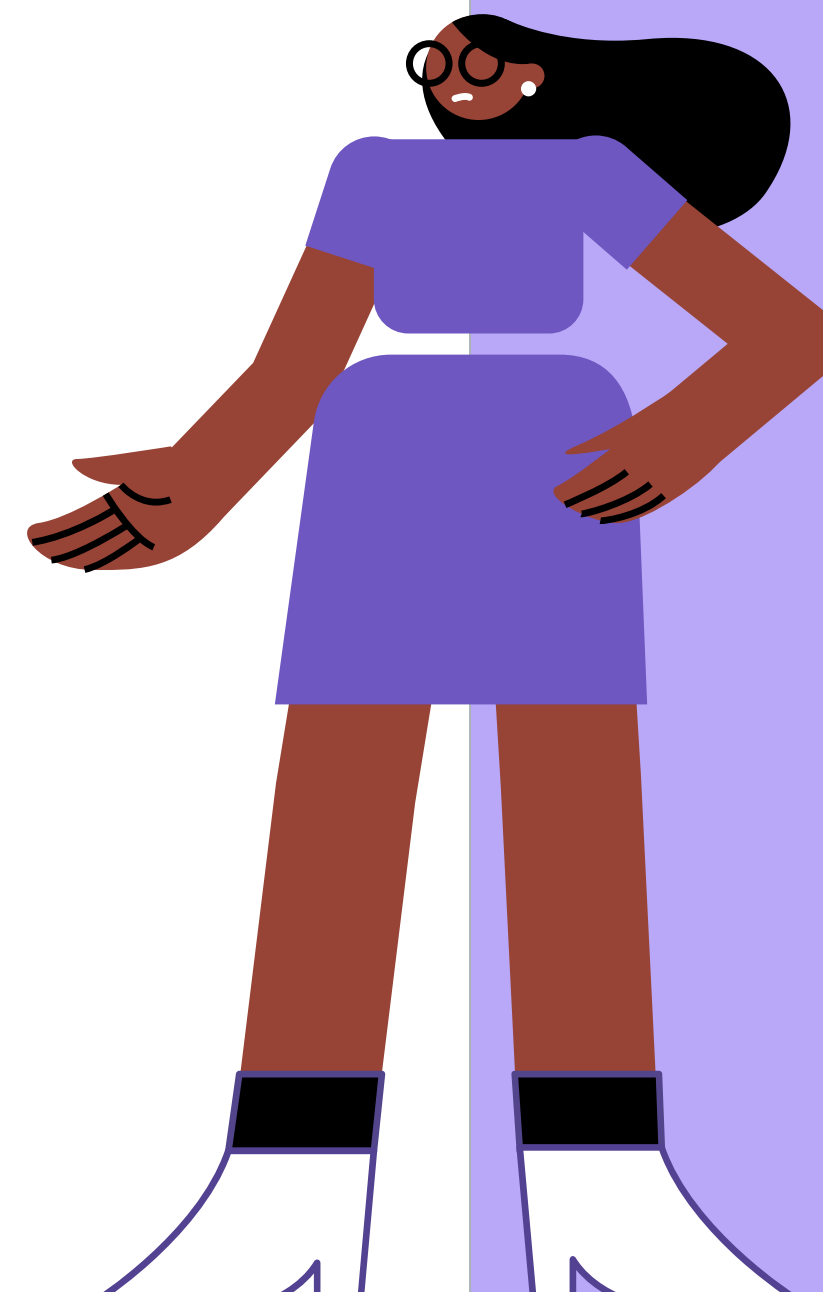
Parar um loop antes dele finalizar pode ser útil, sobretudo em um loop infinito. A palavra reservada **break** nos permite finalizar a execução do loop.



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


Conclusões

A estrutura de controle nos permite iterar determinadas instruções no nosso programa. Assim, podemos percorrer diferentes estruturas de dados e trabalhar sobre elas.



Muito obrigado!