

Loops

01

Basic Loop

02

While Loop

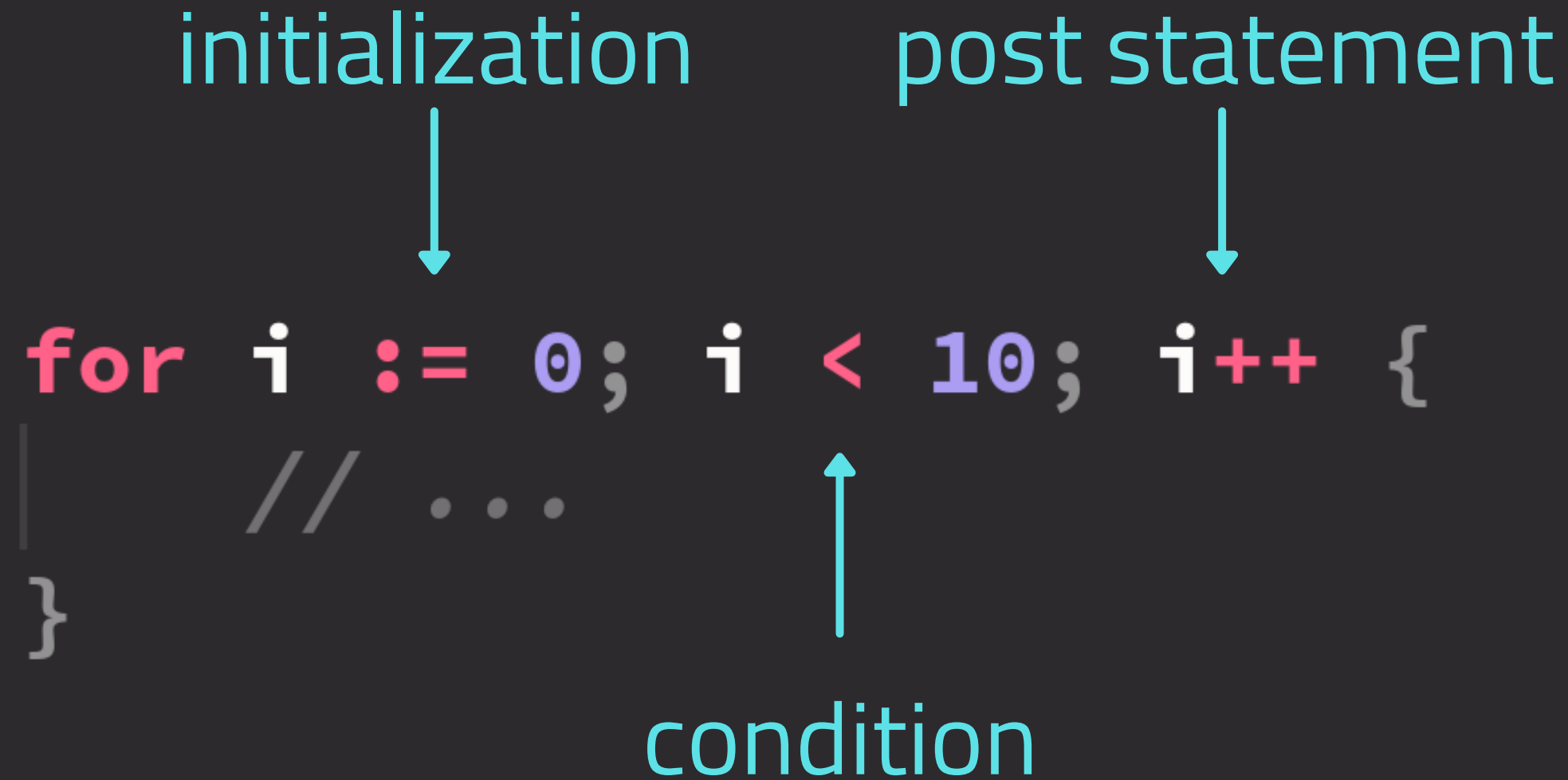
03

Infinite Loop

Looping

- | It is often required to repeat actions in code more than once or to iterate over the items in a collection
 - | These are accomplished using **loops**
- | Go uses the **for** keyword for repetition

for: Basic



The diagram illustrates the components of a basic for loop. It shows the code: `for i := 0; i < 10; i++ {`
|
// ...
}

Annotations with arrows point to the following parts of the code:

- initialization** points to `i := 0`.
- post statement** points to `i++`.
- condition** points to `i < 10`.

The code is color-coded: `for` is red, `i` is white, `:=` is red, `0` is purple, `;` is white, `i` is white, `<` is red, `10` is purple, `;` is white, `i++` is red, and `{` is white.

- | Post statement is executed on each loop iteration
- | Execution continues as long as the condition is true

for: While

```
for i < 10 {  
    // ..  
    i++  
}
```

- | Using loops in this manner require explicitly updating the condition within the loop
 - | Failure to do so results in an infinite loop

for: Infinite

- | Infinite loops are usually used for servers
 - | Use the **break** keyword to exit (break out of) the loop

```
for {  
    // ..  
}
```

```
for {  
    if somethingHappened {  
        break  
    }  
}
```

Continue

- | Use the `continue` keyword to skip the current loop

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

Recap

- | The **for** keyword creates a loop
- | Use the **break** keyword to exit the loop on a specific condition
- | The initialization variable can be used only within the loop block
- | The post statement executes every iteration

```
for i := 0; i < 10; i++ {  
    // ...  
}
```

```
for i < 10 {  
    // ..  
    i++  
}
```

```
for {  
    // ..  
}
```