Level 3–1

Following the Trail

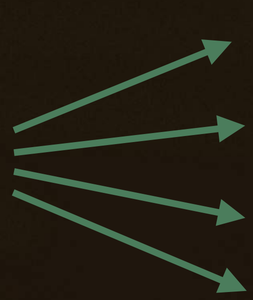
The *for* Loop

ON TRACK with GOLANG

The Only Looping Construct in Go

Unlike other popular languages, the for loop is **the only looping construct** in Go.

Other Languages

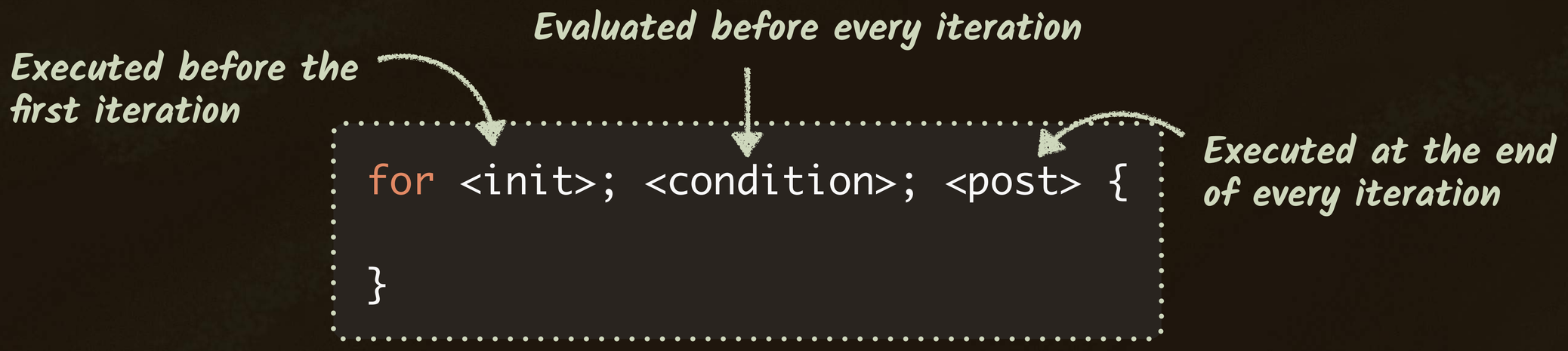


for
while
do/while
each, map, filter, find, indexOf, etc.

Go ⟶ ***for***

The for Loop

There are **no parentheses** in for loops and three different components we can use to control the loop: the **init** statement, a **condition** expression, and a **post** statement.



A Complete for Loop

We can use the := symbol on the **init** statement to create new variables using type inference.

src/hello/main.go

```
package main

import "fmt"

func main() {
  for i := 0; i < 5; i++ {
    fmt.Println(i)
  }
}
```

Create new variable using type inference.

Increments variable by 1 after each run.

Run this loop five times.

```
$ go run main.go
```

```
0
1
2
3
4
```

Print numbers from 0 to 4.

A for Loop With a Single Condition

The for loop components are **optional**. We can create loops with variables declared previously in the code and a **single condition expression.**

```
for <condition> {

}
```

Leave out init and post components.

src/hello/main.go

```
...

func main() {
  i := 0
  isLessThanFive := true
  for isLessThanFive {


    fmt.Println(i)
    i++
  }
}
```

Declare variables and assign initial values.

As long as condition expression is true, the loop will continue to run.

Increment counter i at the end of every run of the loop.

Breaking With a Condition

In order to break from a for loop with no **post** statement, we can change the variable used in the condition expression from inside the body of the loop.

src/hello/main.go

```
...

func main() {
  i := 0
  isLessThanFive := true
  for isLessThanFive {
    if i >= 5 {
      isLessThanFive = false
    }
    fmt.Println(i)
    i++
  }
}
```

```
$ go run main.go
```

```
0
1
2
3
4
5
```

Print numbers from 0 to 5.

Change the condition expression and stops the loop before the next run.

Writing for Loops With No Components

It's also common to write for loops with **no components at all.** To break out of these loops, we can use the break keyword.

src/hello/main.go

```
...

func main() {
  i := 0

  for {
    if i >= 5 {
      break
    }
    fmt.Println(i)
    i++
  }
}
```

The loop stops immediately.

Leave out ALL components.

```
for {
  ...
  break
  ...
}
```

Exit from the loop.

```
$ go run main.go
```

```
0
1
2
3
4
```

Does NOT print number 5

Writing Infinite Loops

Infinite loops are widely used in networking programs. They are useful for **setting up listeners** and **responding to connections.**

src/hello/main.go

```
...

func main() {

  for {
    someListeningFunction()
  }
}
```

Some function listening for connections from other programs

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

Runs indefinitely

```
$ go run main.go
```

→ *(...)*

Does NOT exit the process