Day 21 - Go (Golang) Basics - Loops

A loop is a sequence of instructions that is continually repeated until a certain condition is met.

Go supports only one loop structure, the `for` loop. It can be used in several forms.

Standard For Loop

This form includes an initializer, a condition, and a post statement.

Syntax:

```
for initialization; condition; post {
    // statements
}
```

Example:

```go
package main

import "fmt"

func main() {
    for i := 1; i < 5; i++ {
        fmt.Println(i)
    }
}
```

While-Style For Loop

You can omit the initialization and post statement and use a `for` loop like a `while` loop.

Example:

```go
package main

import "fmt"

func main() {
    i := 1
    for i <= 5 {
        fmt.Println(i)
        i++
    }
}
```

Infinite Loop

A `for` loop with no condition runs forever unless broken manually.

Example:

```go
package main

import "fmt"
```

```go

func main() {
    for {
        fmt.Println("This will run forever")
    }
}
```

Use `break` to exit it when needed.

Break Statement

The `break` statement is used to exit a loop immediately.

Example:

```go
package main
import "fmt"

func main() {
    i := 1
    for i <= 5 {
        if i == 3 {
            break
        }
        fmt.Println(i)
        i++
    }
}


```

```
Output:
1
2
```

Continue Statement

The `continue` statement skips the current iteration and continues with the next.

Example:

```go
package main

import "fmt"

func main() {
    for i := 1; i <= 5; i++ {
        if i == 3 {
            continue
        }
        fmt.Println(i)
    }
}
```

```
Output:
1
```

```
2
4
5
```

Note:

- The `continue` does not stop the loop, it only skips the current iteration.
- `break` exits the loop completely.