Controlling Program Flow

Overview

if, switches

for

Controlling Program Flow

IF, SWITCH

If Statement

`If` statements are used to specify whether a block of code should be executed or not depending on a given condition.

Following is the syntax of `if` statements in Golang -

```
if(condition) {
    // Code to be executed if the condition is true.
}
```

```go
package main
import "fmt"

func main() {
    var x = 25
    if(x % 5 == 0) {
        fmt.Printf("%d is a multiple of 5\n", x)
    }
}
```

```
# Output
25 is a multiple of 5
```

Note that, You can omit the parentheses `()` from an `if` statement in Golang, but the curly braces `{}` are mandatory -

```go
var y = -1
if y < 0 {
    fmt.Printf("%d is negative\n", y)
}
```

You can combine multiple conditions using short circuit operators `&&` and `||` like so -

```go
var age = 21
if age >= 17 && age <= 30 {
    fmt.Println("My Age is between 17 and 30")
}
```

If-Else Statement

An `if` statement can be combined with an `else` block. The `else` block is executed if the condition specified in the `if` statement is false -

```
if condition {
    // code to be executed if the condition is true
} else {
    // code to be executed if the condition is false
}
```

```
package main
import "fmt"

func main() {
    var age = 18
    if age >= 18 {
        fmt.Println("You're eligible to vote!")
    } else {
        fmt.Println("You're not eligible to vote!")
    }
}
```

```
# Output
You're eligible to vote!
```

If-Else-If Chain

`if` statements can also have multiple `else if` parts making a chain of conditions like this -

```
package main
import "fmt"

func main() {
    var BMI = 21.0
    if BMI < 18.5 {
        fmt.Println("You are underweight");
    } else if BMI >= 18.5 && BMI < 25.0 {
        fmt.Println("Your weight is normal");
    } else if BMI >= 25.0 && BMI < 30.0 {
        fmt.Println("You're overweight")
    } else {
        fmt.Println("You're obese")
    }
}
```

```
# Output

Your weight is normal
```

If with a short statement

An `if` statement in Golang can also contain a short declaration statement preceding the conditional expression -

```
if n := 10; n%2 == 0 {
    fmt.Printf("%d is even\n", n)
}
```

Switch Statement

A Switch statement takes an expression and matches it against a list of possible cases. Once a match is found, it executes the block of code specified in the matched case.

```go
package main
import "fmt"

func main() {
    var dayOfWeek = 6
    switch dayOfWeek {
        case 1: fmt.Println("Monday")
        case 2: fmt.Println("Tuesday")
        case 3: fmt.Println("Wednesday")
        case 4: fmt.Println("Thursday")
        case 5: fmt.Println("Friday")
        case 6: {
            fmt.Println("Saturday")
            fmt.Println("Weekend. Yaay!")
        }
        case 7: {
            fmt.Println("Sunday")
            fmt.Println("Weekend. Yaay!")
        }
        default: fmt.Println("Invalid day")
    }
}
```

```
# Output
Saturday
Weekend. Yaay!
```

Switch with a short statement

Just like `if`, `switch` can also contain a short declaration statement preceding the conditional expression. So you could also write the previous switch example like this -

```
switch dayOfWeek := 6; dayOfWeek {
    case 1: fmt.Println("Monday")
    case 2: fmt.Println("Tuesday")
    case 3: fmt.Println("Wednesday")
    case 4: fmt.Println("Thursday")
    case 5: fmt.Println("Friday")
    case 6: {
        fmt.Println("Saturday")
        fmt.Println("Weekend. Yaay!")
    }
    case 7: {
        fmt.Println("Sunday")
        fmt.Println("Weekend. Yaay!")
    }
    default: fmt.Println("Invalid day")
}
```

Combining multiple Switch cases

You can combine multiple `switch` cases into one like so -

```
package main
import "fmt"

func main() {
    switch dayOfWeek := 5; dayOfWeek {
        case 1, 2, 3, 4, 5:
            fmt.Println("Weekday")
        case 6, 7:
            fmt.Println("Weekend")
        default:
            fmt.Println("Invalid Day")
    }
}
```

```
# Output
Weekday
```

Switch with no expression

In Golang, the expression that we specify in the `switch` statement is optional. A `switch` statement without an expression is same as `switch true`. It evaluates all the cases one by one, and runs the first case that evaluates to true -

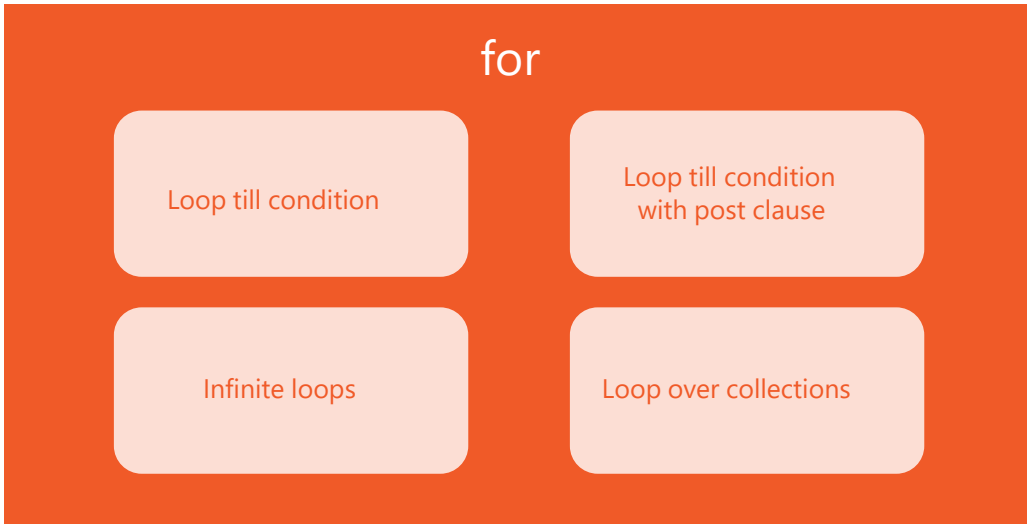
```
package main
import "fmt"

func main() {
    var BMI = 21.0
    switch {
        case BMI < 18.5:
            fmt.Println("You're underweight")
        case BMI >= 18.5 && BMI < 25.0:
            fmt.Println("Your weight is normal")
        case BMI >= 25.0 && BMI < 30.0:
            fmt.Println("You're overweight")
        default:
            fmt.Println("You're obese")
    }
}
```

Switch without an expression is simply a concise way of writing `if-else-if` chains.

Controlling Program Flow

FOR

Looping



For Loop

A loop is used to run a block of code repeatedly. Golang has only one looping statement - the `for` loop.

Following is the generic syntax of `for` loop in Go -

```
for initialization; condition; increment {
    // loop body
}
```

The **initialization** statement is executed exactly once before the first iteration of the loop. In each iteration, the **condition** is checked. If the condition evaluates to `true` then the body of the loop is executed, otherwise, the loop terminates. The **increment** statement is executed at the end of every iteration.

Here is a simple example of a for loop -

```
package main
import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        fmt.Printf("%d ", i)
    }
}
```

```
# Output
0 1 2 3 4 5 6 7 8 9
```

Unlike other languages like C, C++, and Java, Go's for loop doesn't contain parentheses, and the curly braces are mandatory.

Note that, both initialization and increment statements in the `for` loop are optional and can be omitted

- Omitting the initialization statement

```
package main
import "fmt"

func main() {
    i := 2
    for ;i <= 10; i += 2 {
        fmt.Printf("%d ", i)
    }
}
```

```
# Output

2 4 6 8 10
```

- Omitting the increment statement

```
package main
import "fmt"

func main() {
    i := 2
    for ;i <= 20; {
        fmt.Printf("%d ", i)
        i *= 2
    }
}
```

```
# Output

2 4 8 16
```

Note that, you can also omit the semicolons from the `for` loop in the above example and write it like this -

```
package main
import "fmt"

func main() {
    i := 2
    for i <= 20 {
        fmt.Printf("%d ", i)
        i *= 2
    }
}
```

The above `for` loop is similar to a `while` loop in other languages. Go doesn't have a `while` loop because we can easily represent a `while` loop using `for`.

Finally, You can also omit the condition from the `for` loop in Golang. This will give you an infinite loop -

```
package main

func main() {
    // Infinite Loop
    for {
    }
}
```

break statement

You can use `break` statement to break out of a loop before its normal termination.

```
package main
import "fmt"

func main() {
    for num := 1; num <= 100; num++ {
        if num%3 == 0 && num%5 == 0 {
            fmt.Printf("First positive number divisible by both 3 and 5 is %d\n", num)
            break
        }
    }
}
```

```
# Output
First positive number divisible by both 3 and 5 is 15
```

continue statement

The `continue` statement is used to stop running the loop body midway and continue to the next iteration of the loop.

```go
package main
import "fmt"

func main() {
    for num := 1; num <= 10; num++ {
        if num%2 == 0 {
            continue;
        }
        fmt.Printf("%d ", num)
    }
}
```

```
# Output

1 3 5 7 9
```

Overview

if, switches

for