Working with Conditionals

Nigel Poulton
Author & Trainer

@nigelpoulton   nigelpoulton.com

```
if userAge >= 18 {
   <code A>
} else {
   <code B>
}
```

Agenda

"If" syntax

"If" in practice

Simple initialization

Switch syntax

Switch in practice

Breaking and fallthrough

"If" and error handling

Recap

“If” Syntax

Evaluate conditions

Based on Boolean true/false logic

Branching

Execute code based on evaluation

```
if userAge >= 18 {
  <code>
} else {
  <code>
}
```

Evaluate conditions

Based on Boolean true/false logic

Branching

Execute code based on evaluation

Evaluate conditions

Based on Boolean true/false logic

Branching

Execute code based on evaluation

Evaluate conditions

Based on Boolean true/false logic

Branching

Execute code based on evaluation

```
if
```

Start with `if` keyword

```
if userAge >= 18 {
   <code A>
}
```

Start with `if` keyword

Evaluate a Boolean expression (logical true/false)

Curly placement is vital

```
if userAge >= 18 {
  <code A>
} else if {
  <code B>
} else {
  <code C>
}
```

Start with `if` keyword

Evaluate a Boolean expression (logical true/false)

Curly placement is vital

Multiple `else if` statements and a single `else`

Switch and Case Syntax

```
switch
```

Similar to “If”

```
switch <simple-statement>; <expression>
```

Similar to “If”

Variables declared here are scoped to the switch block

```
switch <simple-statement>; <expression> {
}
```

Similar to “If”

Variables declared here are scoped to the switch block

Curly placement is vital

```
switch <simple-statement>; <expression> {
case <value>: <code>
case <value>: <code>
case <value>: <code>
default: <code>
}
```

Similar to “If”

Variables declared here are scoped to the switch block

Curly placement is vital

`default` block runs if no `case` statements evaluate to true

```
switch "Kubernetes Deep Dive" {
case "Kubernetes Deep Dive": code-A
case "K8s Deep Dive": code-B
case "Docker Networking": code-C
default: <code>
}
```

Similar to "If"

Variables declared here are scoped to the switch block

Curly placement is vital

`default` block runs if no `case` statements evaluate to true

Breaking and Fallthrough

“If” and Error Handling

```go
func testConn(target string) (rspTime float64, err error)
```

Idiomatic to return an error as the last return from functions and methods

```go
func testConn(target string) (rspTime float64, err error)
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

```go
func testConn(target string) (rspTime float64, err error)
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

nil/zero code indicates success

```go
func testConn(target string) (rspTime float64, err error)
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

nil/zero code indicates success

Non-zero code indicates error

Recap

```
if userAge >= 18 {
  <code A>
}
```

Start with `if` keyword

Evaluate a Boolean expression (logical true/false)

```
if userAge >= 18 {
  <code A>
} else if {
  <code B>
}
```

Start with `if` keyword

Evaluate a Boolean expression (logical true/false)

Multiple `else if` statements

```
if userAge >= 18 {
   <code A>
} else if {
   <code B>
} else {
   <code C>
}
```

Start with `if` keyword

Evaluate a Boolean expression (logical true/false)

Multiple `else if` statements

Single `else`

```go
func Open(name string) (*File, error) {

}
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

```go
func Open(name string) (*File, error) {

    _, err := os.Open("./test.txt")

}
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

```go
func Open(name string) (*File, error) {

    _, err := os.Open("./test.txt")

    if err != nil {
        fmt.Println("This is the error code:", err)
    }
}
```

Idiomatic to return an error as the last return from functions and methods

Error is a type in Go

nil/zero code indicates success

Non-zero code indicates error

```
switch <expression> {
case <value>: <code> <implicit break>
case <value>: <code> <implicit break>
case <value>: <code> <implicit break>
case <value>: <code> <implicit break>
default: <code>
}
```

Similar to “If”

```
switch <expression> {
case <value>: <code> fallthrough
case <value>: <code> <implicit break>
case <value>: <code> <implicit break>
case <value>: <code> <implicit break>
default: <code>
}
```

Similar to “If”

```
switch <expression> {
case <value>: <code> fallthrough
case <value>: <code> fallthrough
case <value>: <code> fallthrough
case <value>: <code> <implicit break>
default: <code>
}
```

Similar to “If”

Up Next:
Working with Loops