PART III

Fundamentals

Control Flow

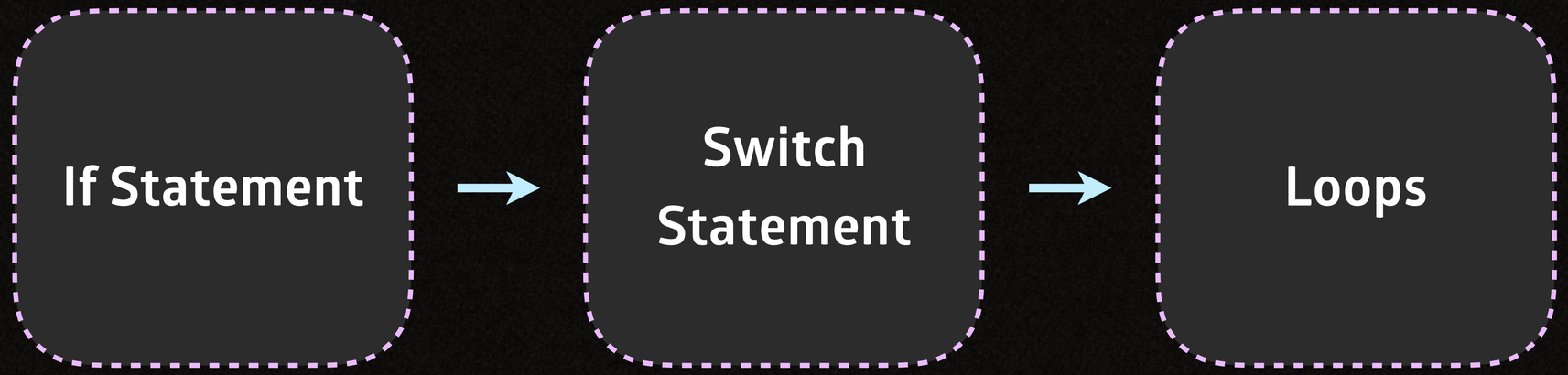
```go
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

```go
package main
import "fmt"

func main() {
    fmt.Println("Hello!")

    if 5 > 1 {
        fmt.Println("bigger")
    }
}
```

PART III

Fundamentals

Control Flow



IF STATEMENT

*Time to learn how to **control** the **execution** flow*

Boolean Operators → If Statement → Error Handling → Simple Statement

Comparison and,
Logical Operators

Only for
—beginners—

Refactor:
Feet to Meters

Project:
PassMe

nil value
error value

Refactor:
Feet to Meters

Famous
Shadowing
Gotcha

BOOLEAN OPERATORS

You're going to learn about boolean operators

Comparison Operators → Logical Operators

OPTIONAL

for experienced programmers

COMPARISON OPERATORS

to be true or not to be true that's the question

EQUALITY OPERATORS

These operators can be used with any comparable type and they yield a bool value

name	operator	example *boolean expressions*	result
equal	==	3 **==** 3	**true**
not equal	!=	"hello" **!=** "hi"	**true**

ORDERING OPERATORS

These operators can be used with any comparable type and they yield a bool value

name	operator	example *boolean expressions*	result
less	<	3 **<** 2.5	**false**
less or equal	<=	4.0 **<=** 4	**true**
greater	>	"abc" **>** "cde"	**false**
greater or equal	>=	'b' **>=** 'a'	**true**

COMPARISON & ASSIGNABILITY

when you can't assign then you can't compare as well

COMPARISON & ASSIGNABILITY

*when you **can't assign** then you **can't compare** as well*

```
speed == "100"
```

```
speed >= force
```

```
func main() {
    var speed int
    speed = "100"

    var running bool
    running = 50

    var force float64
    speed = force
}
```

```
running != 50
```

COMPARISON & ASSIGNABILITY

when you can't assign then you can't compare as well

```
func main() {
  var speed int
  speed =

  var running bool
  running =

  var force float64
  speed =     force
}
```

COMPARISON & ASSIGNABILITY

when you can't assign then you can't compare as well

✓ `speed >= 50`

```
func main() {
  var speed int
  speed = 50

  var running bool
  running = true

  var force float64
  speed = int(force)
}
```

✓ ✓ ✓

`running == true` ✓

`speed == int(force)` ✓

BOOLEAN OPERATORS

Congrats!

Comparison Operators ✓ → Logical Operators

LOGICAL OPERATORS: AND

“Logic will take you from A to B. Imagination will take you everywhere”

- Albert Einstein

a logical operator
combines
bool expressions

and yields
a bool value

a logical operator
can only work with
bool expressions

shouldIStay := true

shouldIGo := false

shouldIStay **OR** shouldIGo

myAge := 25

yourAge := 20

myAge ~~OR~~ yourAge

AND OPERATOR

"Logical AND" operator can only be used with bool values

operand	operator	operand	result
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

LOGICAL OPERATORS: OR

“Right or Wrong, make a decision”

- Anonym

OR OPERATOR

"Logical OR" operator can only be used with bool values

operand	operator	operand	result
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false

LOGICAL OPERATORS: NOT

LOGICAL OPERATORS

operator	expression	result
&&	*true* *true* "hi" == "hi" && 3 > 2	true
||	*false* *true* "hi" != "hi" || 3 > 2	true
!	!("hi" != "hi" || 3 > 2)	false

LOGICAL OPERATORS

NOT Operator Example

```go
func main() {
   var on bool

   on = !on
   fmt.Println(on)

   on = !!on
   fmt.Println(on)
}
```

NOT OPERATOR

!true *becomes* ***false***

!false *becomes* ***true***

!!true *becomes* ***true***

it can only be used with a **bool value**

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

BOOLEAN OPERATORS

Congrats!

Comparison Operators ✓ → Logical Operators ✓

summary

comparison operators

equality operators

ordering operators

equality operators

```
true == true   →   true
true != true   →   false
```

ordering operators

```
1 < 2   →   true
2 > 5   →   false
2 >= 2  →   true
3 <= 2  →   false
```

assignability and comparison

```
i, f := 15, 5.0

i > f

i > int(f)
```

a logical operator
combines
bool expressions

and yields
a bool value

logical operators

Logical AND Operator

```
true && true        → true
```

Logical OR Operator

```
false || false      → false
```

Logical NOT Operator

```
!false              → true
!!false             → false
```

short-circuit

`false && true`	`true || false`
↑	↑
runs	**runs**
↑	↑
skips	**skips**

IF STATEMENT

Congrats!

Boolean Operators ✓ → If Statement → Error Handling → Simple Statement

IF STATEMENT

if ... else if ... else if ... else ...

IF STATEMENT

This if statement will only print "good" when the score is higher than 3

```
if (score > 3) {
    fmt.Println("good")
}
```

IF STATEMENT

if statement doesn't require parentheses

```
if score > 3 {
    fmt.Println("good")
}
```

BLOCK

if statement's block is executed only if its condition expression is true

```
if score > 3 {
    fmt.Println("good")
}
```

Declared names in this **block**
are **only visible inside** the if block
and in the following branches...

CONDITION EXPRESSION

*This **expression** should always yield a **bool value***

```
if score > 3 {
    fmt.Println("good")
}
```

👉 *Check out the Javascript example in the resources!*

CONDITION EXPRESSION

Condition expression** should always yield a **bool value

bool

```
if score > 3 && valid == true {
    fmt.Println("good")
}
```

CONDITION EXPRESSION

You can use bool values directly

```
if score > 3 && valid {
    fmt.Println("good")
}
```

CONDITION EXPRESSION

You can use bool values directly

```
if score > 3 && !valid {
    fmt.Println("good")
}
```

EXAMPLE

```go
func main() {
   score, valid := 5, true

   if score > 3 && valid {
      fmt.Println("good")
   }
}
```

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

EXAMPLE

```go
func main() {
  score, valid := 3, true

  if score > 3 && valid {
    fmt.Println("good")
  }
}
```

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

ELSE AND ELSEIF BRANCHES

if ... else if ... else if ... else ...

ELSE BRANCH

Previously...

```go
func main() {
  score, valid := 3, true

  if score > 3 && valid {
    fmt.Println("good")
  }
}
```

ELSE BRANCH

Else branch will print "low" when if branch is not executed

```go
func main() {
  score, valid := 3, true

  if score > 3 && valid {
    fmt.Println("good")
  } else {
    fmt.Println("low")
  }
}
```

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

ELSE BRANCH

If branch's condition was false

```
func main() {
  score, valid := 3, true

  if score > 3 && valid {
    fmt.Println("good")
  } else {
    fmt.Println("low")
  }
}
```

false

ELSE BRANCH

Else branch is executed if all the other branches are false

```go
func main() {
  score, valid := 3, true

  if score > 3 && valid {
    fmt.Println("good")
  } else {
    fmt.Println("low")
  }
}
```

***executed** if*
all the branches
*are **false***

ELSE IF BRANCH

Else if branch** is like an **if branch

```go
func main() {
    score := 3

    if score > 3 {
        fmt.Println("good")
    } else if score == 3 {
        fmt.Println("on the edge")
    } else {
        fmt.Println("low")
    }
}
```

false
skip

*one of the previous branches were **true***
skip

score is 3
execute

```
$ go run main.go
on the edge
```

```go
func main() {
  score := 2

  if score > 3 {
    fmt.Println("good")
  } else if score == 3 {
    fmt.Println("on the edge")
  } else if score == 2 {
    fmt.Println("meh...")
  } else {
    fmt.Println("low")
  }
}
```

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

only once
required

```go
func main() {
  score := 2

  if score > 3 {
    fmt.Println("good")
  } else if score == 3 {
    fmt.Println("on the edge")
  } else if score == 2 {
    fmt.Println("meh...")
  } else {
    fmt.Println("low")
  }
}
```

```go
func main() {
  score := 2

  if score > 3 {
    fmt.Println("good")
  } else if score == 3 {
    fmt.Println("on the edge")
  } else if score == 2 {
    fmt.Println("meh...")
  } else {
    fmt.Println("low")
  }
}
```

sky is the limit...

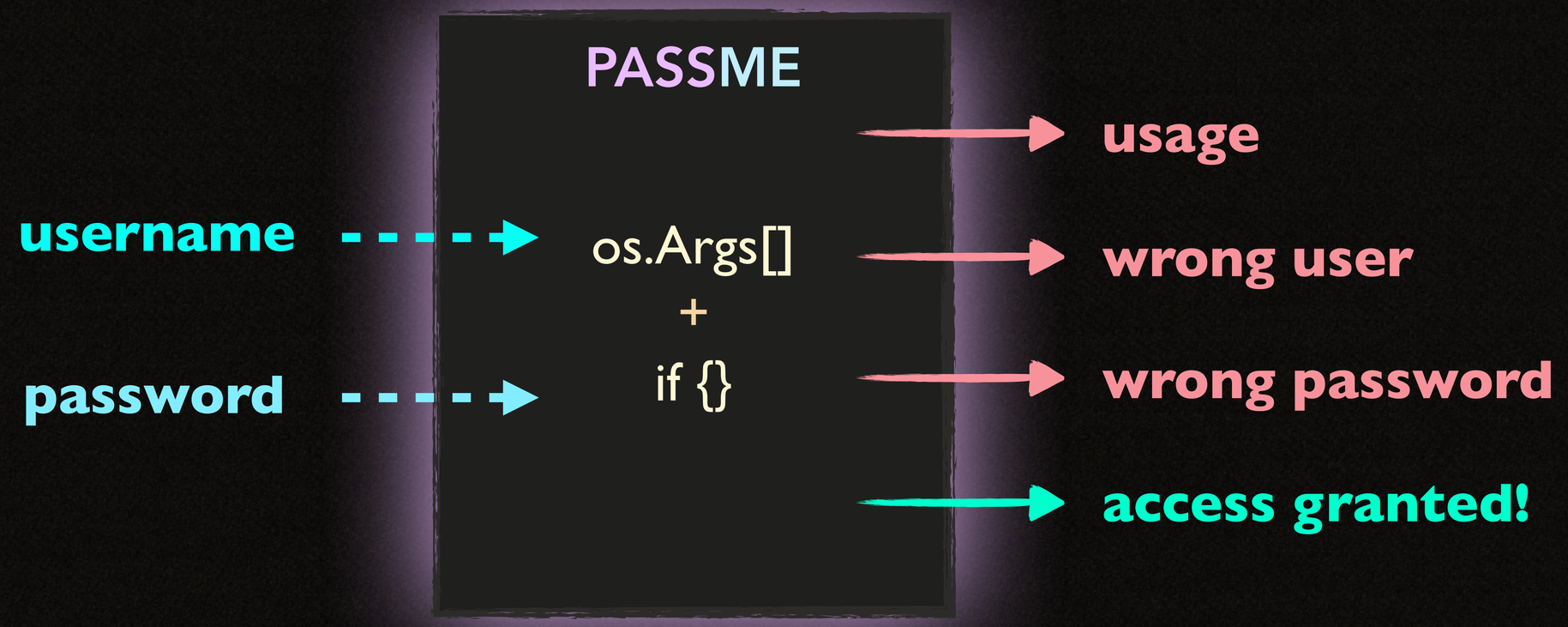
optional

```go
func main() {
  score := 2

  if score > 3 {
    fmt.Println("good")
  } else if score == 3 {
    fmt.Println("on the edge")
  } else if score == 2 {
    fmt.Println("meh...")
  } else {
    fmt.Println("low")
  }
}
```

only once
optional

PASSME

User Password Protection is ON!



It should work only one username and password for now...

PASSME

User Password Protection is ON!

PASSME

CHALLENGE!
Add one more user with its own password!

go run main.go inanc 1879 ✓

go run main.go jack 1888 ✓

PASSME

SOLUTION

Add one more user with its own password!

go run main.go inanc 1879 ✓

go run main.go jack 1888 ✓

summary

if statement doesn't require parentheses

```
if (true) {...}  ✗
        ↓
if true   {...}  ✓
```

condition expressions can only be bool

```
if 1 {...}
```

```
if true {...}
```

directly use bools

```
if true == true {...}
```

```
if true {...}
```

if statement

```
if true {
} else if true {
} else if true {
} else if true {
} else {
}
```

IF STATEMENT

Congrats!

Boolean Operators ✓ → If Statement ✓ → Error Handling → Simple Statement

ERROR HANDLING BASICS

You're going to learn how to handle the errors

Why?

Not everything goes according to the plan

Things can fail

- **Network problems**
- **File access problems**
- **User input problems**
- **...**

Error handling is crucial

And it is part of your programming job

nil

nil value means that the value is not initialized yet

javascript	python	java	ruby
null	None	null	nil

nil

zero value for pointer-based types

pointers	slices	maps	interfaces	channels
nil	nil	nil	nil	nil

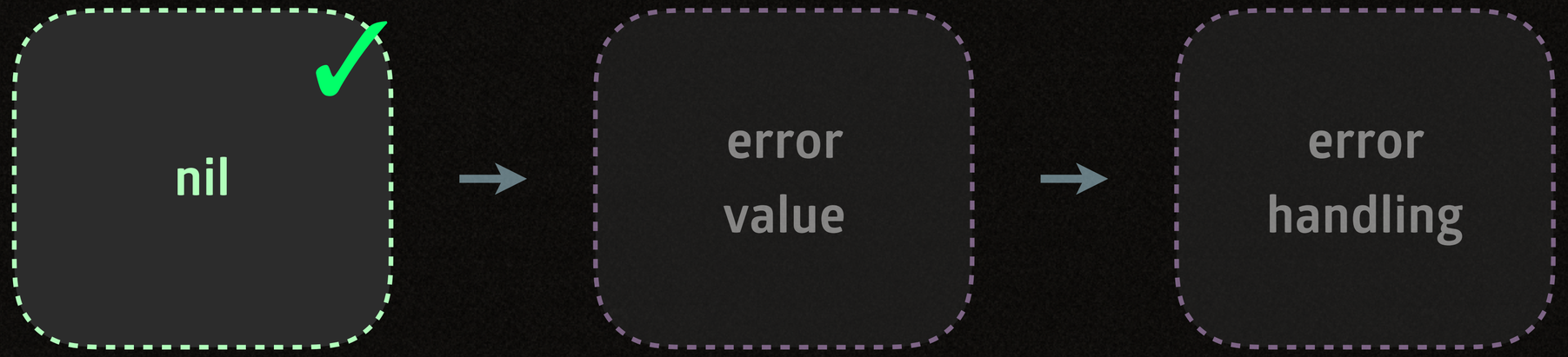
nil

means there is no error

```
err := do()

if err != nil {
  // ❌ error
  // handle it!
  // terminate!
}

// 👍 success
// continue...
```

ERROR HANDLING BASICS

Congrats!



error value

ERROR VALUE

***strconv.Itoa** never fails*

```
func Itoa(i int) string
```

✓ **this function never returns errors**

ERROR VALUE

strconv.Atoi** sometimes **fails**, so you've to handle the **error

```
func Atoi(s string) (int, error)
```

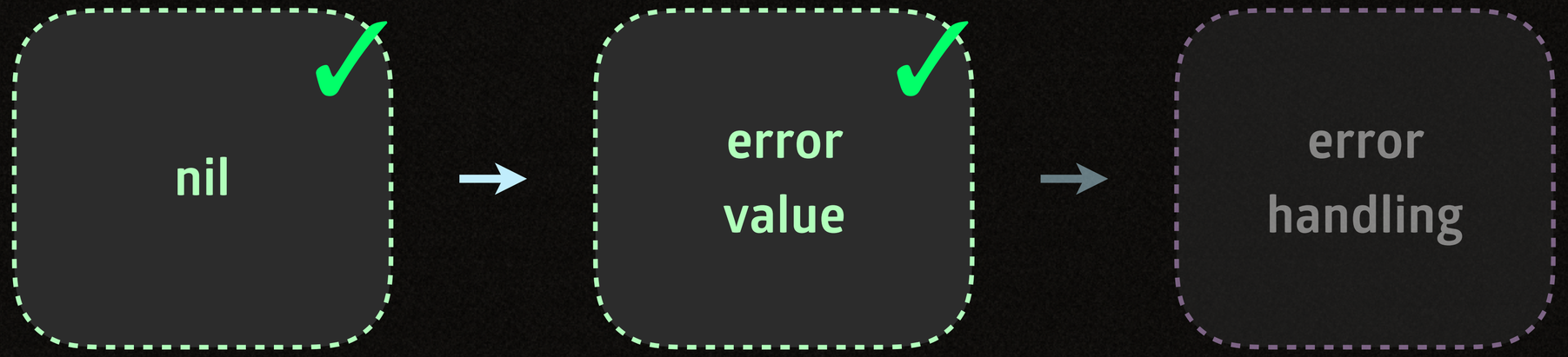
returns a value with an error type

✓ **success** → **nil**
(uninitialized error value)

✗ **error** → non-**nil**
(initialized error value)

ERROR HANDLING BASICS

Congrats!



FEET TO METERS

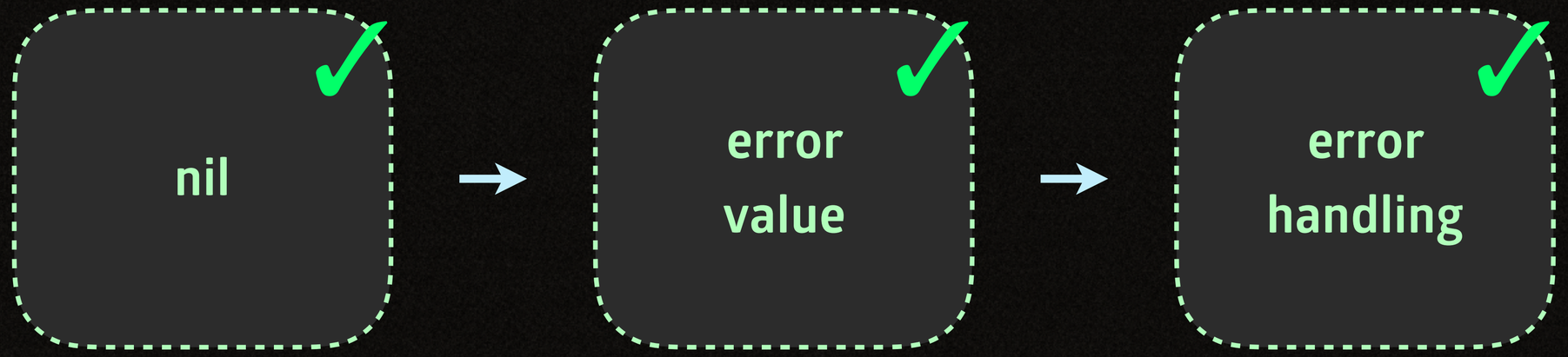
challenge: handle the error!

FEET TO METERS

solution: handle the error!

ERROR HANDLING BASICS

Congrats!



summary

nil

```
if err != nil {
  // ❌ error
  // quit
}
// 👍 success
```

error

```
if err != nil {
  // ❌ error
  // quit
}
// 👍 success
```

immediately handle the errors

```
if err != nil {
  // ❌ error
  // quit
}
// 👍 success
```

there is no throw/catch

```
if err != nil {
  // ❌ error
  // quit
}
// 👍 success
```

IF STATEMENT

Congrats!

Boolean Operators ✓ → If Statement ✓ → Error Handling ✓ → Simple Statement

SHORT IF

You're going to learn how to use Simple Statements with an if statement

Short If → Scopes → Shadowing

SHORT IF

You can use a Simple Statement with an if statement

(Short Statements)

SHORT IF

An example snippet of code which doesn't use a simple statement

```
n, err := strconv.Atoi("42")

if err == nil {
  fmt.Println("There was no error, n is", n)
}
```

SHORT IF

***Simple statements** allow you to execute a statement **inside** another statement*

```
if n, err := strconv.Atoi("42"); err == nil {
    fmt.Println("There was no error, n is", n)
}
```

SIMPLE STATEMENT

***Simple statements** allow you to execute a statement **inside** another statement*

```
if n, err := strconv.Atoi("42"); err == nil {
    fmt.Println("There was no error, n is", n)
}
```

Simple statements are completely optional

SEPARATOR

; separator separates the simple statement and the condition expression

separator

```
if n, err := strconv.Atoi("42"); err == nil {
    fmt.Println("There was no error, n is", n)
}
```

CONDITION EXPRESSION

***Condition expression** can use the **declared variables** inside the simple statement*

condition expression

```
if n, err := strconv.Atoi("42"); err == nil {
    fmt.Println("There was no error, n is", n)
}
```

SCOPE

Declared variables are only visible inside the if statement (and its branches)

```
if n, err := strconv.Atoi("42"); err == nil {
    fmt.Println("There was no error, n is", n)
}
```

SHORT IF

Well done!

Short If ✓ → Scopes → Shadowing

SCOPES

Example

SHORT IF

Well done!

Short If ✓ → Scopes ✓ → Shadowing

SHADOWING

Famous Gotcha

SHORT IF

Well done!

Short If ✓ → Scopes ✓ → Shadowing ✓

summary

simple statement

(as known as the short statement)

```
err := do()
if err != nil {}
```

↓

```
if err := do(); err != nil {}
```

scopes

```
if p := 1; ... {
} else if q := 2; ... {
} else {
}
```

shadowing gotcha

```
err := do()

if err := run(); err != nil {}
```

IF STATEMENT

Congrats!

Boolean Operators ✓ → If Statement ✓ → Error Handling ✓ → Simple Statement ✓

PART III

Fundamentals

Control Flow

If Statement ✓ → Switch Statement → Loops