Conditionals

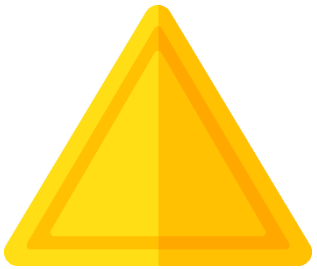
Nigel Poulton

@nigelpoulton | www.nigelpoulton.com

Module Objectives

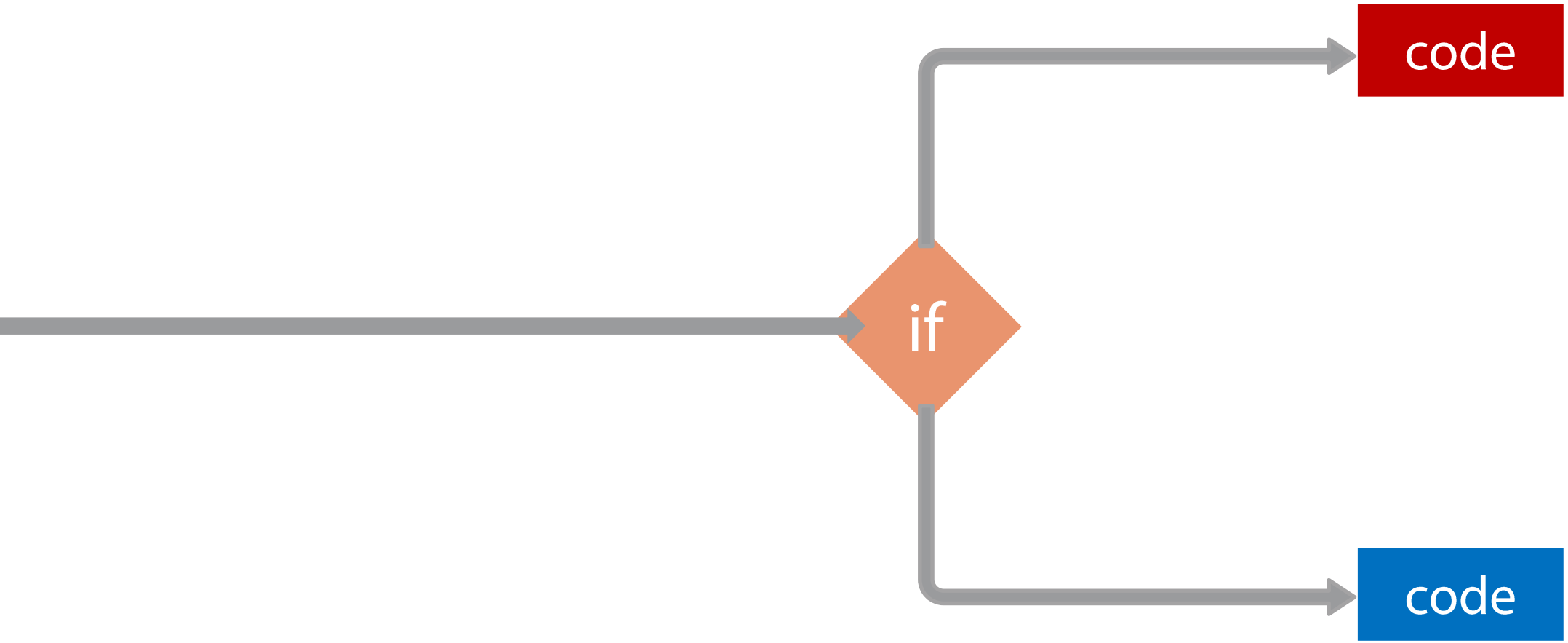
`if`

`switch`



Highlighting things specific to Go

The `if` Statement


code
if
code

```
if <Boolean expression> {
{  <code block>
} else if <Boolean expression> {
   <code block>
} else {
   <code block>
}
```

Boolean Comparison Operators

==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
&&	AND
||	OR

```
if <Boolean expression> {
   <code block>
}
```

`if`

- Accepts Boolean expressions
- Placement of {} are vital
- Variables declared as part of the **`if`** statement are scoped to it
- Supports multiple **`else if`** statements
- Supports a single **`else`** statement (must be last)
- First match in lexical order gets executed
- Supports nesting

The `switch` Statement

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

```
switch “Double Dependency” { <expression> {
case “Dapper Deep Dive”: <code>
case “Expression Trees”: <code>
default: <code>
}
```

`switch`

Placement of {} are vital

`default` statement *can* go anywhere

`switch` and **`case`** *types* must match

Implicit breaks

Requires explicit fallthrough

`case` statements can have multiple comma-separated values

The Role of `if` in Error Handling

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

if err != nil {
     <error handling code>
}
<code…>
```

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

`nil` is used to indicate success

Idiomatic to **always** check the value of returned errors

Summary

if

```
if a < b {
  <code>
} else if b < a {
  <code>
} else {
  <code>
}
```

- Boolean expressions
- {} placement vital
- Multiple **`else if`**
- Single **`else`**

```
if err != nil {
  <error handling>
}
  <normal code…>
```

- Only first match executed
- Nesting Supported
- Used for error checking

switch

```
switch “value” {
case “test1”:
  <code>
case “test2”:
  <code>
case “test3”:
  <code>
}
```

- Test for matched expressions
- Can be cleaner than `if else-if else` chains
- `switch` and `case` types must match
- Implicit breaks