codecademy

Learn Go: Conditionals

Go If Statement

A Go `if` statement evaluates a condition and executes a statement block enclosed in curly braces `{..}` if the evaluation returns `true`. The condition may be optionally enclosed inside a pair of parentheses `(...)`.

```
if (healthy) {
  fmt.Println("Work.")
}
if sick {
  fmt.Println("Stay home.")
}
```

Go else Statement

A Go `else` statement can succeed an `if` or `if else-if` statement block and its code executed if the conditions in the preceding `if` or `if else-if` statements evaluate to `false`.

```
sick := false
if sick {
  fmt.Println("Call the doctor.")
} else {
  fmt.Println("Enjoy your day.")
}
```

Go Comparison Operators

Go supports the standard comparison operators that compare two values and return a boolean. These include:

- `==` equivalence operator
- `!=` not equal
- `<` less than
- `>` greater than
- `<=` less than or equal
- `>=` greater than or equal

```
same := 3 == 3
// evaluates to true
notsame := "ABC" != "abc"
// evaluates to true
lessthan := 5 <= -5
// evaluates to false
```

Go Logical Operators

In addition to comparison operators, Go also supports logical operators which evaluate boolean values and return a boolean value. For example:

- `&&` is the AND operator that returns `true` if all the booleans are `true`
- `||` is the OR operator that returns `true` if one of the booleans is `true`
- `!` is the NOT operator that returns the opposite of a boolean value

```
answer := true && false
// returns false
answer = true || false
// returns true
answer = !false
// returns true
```

Go Else If Statement

The Go `else if` statement provides an additional condition to evaluate besides the first `if` conditional. It can only appear after the `if` statement and before an `else` statement if it exists. For example:

```
if (temperature < 60) {
  fmt.Println("Put on a jacket.")
} else if (temperature >= 60 &&
temperature < 75) {
  fmt.Println("Put on a light
sweater.")
} else {
  fmt.Println("Wear summer clothes.")
}
```

Multiple `else if` statements can exist alongside the `if` statement. The `if` `else if` statements are scanned from top to bottom and only the code block associated with a true condition is executed. If none of the conditions are satisfied, the `else` code block is executed if it exists.

Go Short Variable Declaration

A short variable declaration can be made within the scope of an `if` or `switch` statement before the condition is specified but after the `if` or `switch` keyword. A semicolon, `;` , is appended to the declaration to separate it from the condition.

```
if age := 55; age >= 55 {
  fmt.Println("You are retiring!")
}
switch season := "spring"; season {
  case "spring":
    fmt.Println("Plant some bulbs.")
  case "summer":
    ...
}
```

Go Switch Statement

The Go `switch` statement can be used as an alternative to a set of `if` followed by `else if` statements. The `switch` statement compares the expression inside a condition with a set of values encapsulated in `case` s. The code inside a matched `case` value is executed and the `switch` block terminates. A `default` case without a value can be appended to the last `case` and its code executed if no prior match is found.

```
day := "Tuesday"
switch day {
  case "Monday":
    fmt.Println("Monday is
magnificent.")
  case "Tuesday":
    fmt.Println("Tuesday is
terrific.")
  case "Wednesday":
    fmt.Println("Wednesday is wacky.")
  default:
    fmt.Println("We survived.")
}
```

Go Seed Value

A seed value in Go is used for generating random numbers. By default, the seed value is `1` and this leads to a predictable number instead of random. To make the seed value unique, call the seed function, `rand.Seed()` , with the argument `time.Now().UnixNano()` to return the difference in time (in Nanoseconds) since Janurary 1st 1970.

```
rand.Seed(time.Now().UnixNano())
```

Go Random Number Generator

Go provides a function, `math.rand.Intn()`, in the `math.rand` package to generate a random number. To generate such a number between `0` to `99`, pass `100` as the function argument.

```
number := math.rand.Intn(100)
```