## Control Flow

Control Flow in Go is similar to most C family languages but has some novel
differences. Classic structures like `if` and `switch` are present but their
syntax is a little different from what you might be used to.

There are no ternary operators.

## Notes

* `if` statements require an exact boolean expression. There is no "truthy" or "falsey".
* `if` statements may have a pre-expression similar to the first part of a `for` statement.
* `switch` statements do not require `break`s and will not fall through by default.
* `switch` statements can have multiple values per case.
* Use the `||` "or" and `&&` "and" operators for complex conditions.
* Use shorter variable names that still provide context for the value they represent.

## Code Review

[if statements](example1/example1.go)
[switch statements](example2/example2.go)
[variable names](example3/example3.go)
## Functions

Functions are at the core of the language. They provide a mechanism to group and organize our code to separat
e and distinct pieces of functionality. They can be used to provide an API to the packages we write and are a
 core component to concurrency.

## Notes

* Functions can return multiple values and most return an error value.
* The error value should always be checked as part of the programming logic.
* The blank identifier can be used to ignore return values.


Use ‘_’ to keep unused imports
println
## Grouping
Don't group types by a common DNA but by a common behavior.
    [Grouping By State](grouping/example1/example1.go)
    [Grouping By Behavior](grouping/example2/example2.go)

## Assertion
[Interface Conversions](assertions/example1/example1.go)
[Runtime Type Assertions](assertions/example2/example2.go)
[Behavior Changes](assertions/example3/example3.go)
## packaging allows a developer to identify where a package belongs inside a Go project and the design guidel
ines the package must respect. It defines what a Go project is and how a Go project is structured. Finally, i
t improves communication between team members and promotes clean package design and project architecture that
 is discussable.
## Error Handling