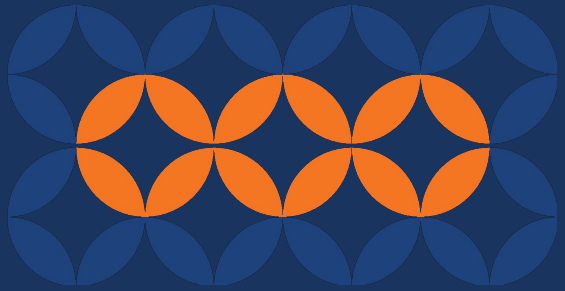
alterra
academy



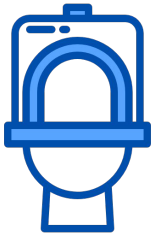
Basic Programming of Golang

alterra academy

OUR RULES

Silent Mode

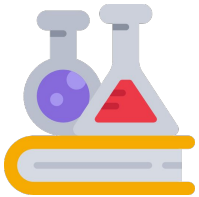
Ask Question



Go Toilet

TIME ALLOCATION

Explanation



Challenge

Review

OUTLINE

alterra academy

- Introduction Golang
- Fmt Package
- Go Module
- Package "fmt"
- Variables, types, const, zero value
- Arithmetic Operators
- Control Structures (Branching & Looping)

alterra
academy

Let's get started!

alterra
academy

Introduction Golang

alterra
academy

WHAT IS GO?

is a new, general-purpose programming language that makes it easy to build, simple, reliable, and efficient software.

alterra academy

SYSTEM PROGRAMMING

is a great language for writing lower-level program that provide service to other systems, called system programming

Application Programs

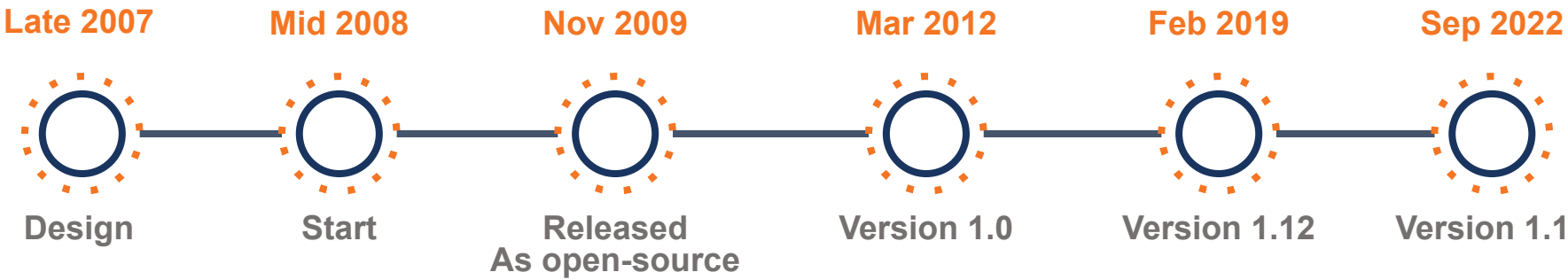
E-commerce
Music Player
Social Media Apps

System Programs

APIs
Game engines
CLI apps

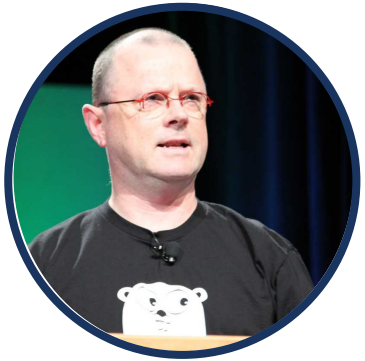
alterra academy

HISTORY OF GOLANG



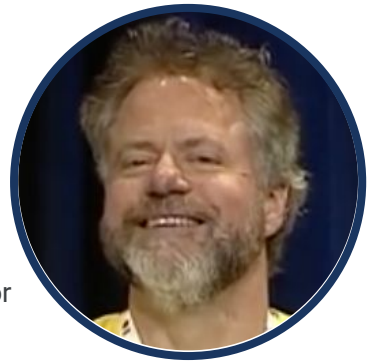
DEVELOPED BY :

Robert Grisemer



Rob Pike

Ken Thompson



Ian Lance Taylor

Russ Cox

alterra academy

WHY (MUST) GO?

- Fun and Make Programming Fun (simple)
- Combine the best of both.
  - Compiled statically-typed languages (like C) and Dynamic languages (like JavaScript).
  - Go has the lighter feel of a scripting language but is compiled.
- Lightweight syntax.
- Built-in concurrency.
  - Large scale, networked computing, such as Google web search
  - Multi-core hardware
- Open Source.
- Used by Big Company

alterra academy

COMPANIES USING GO

Uber

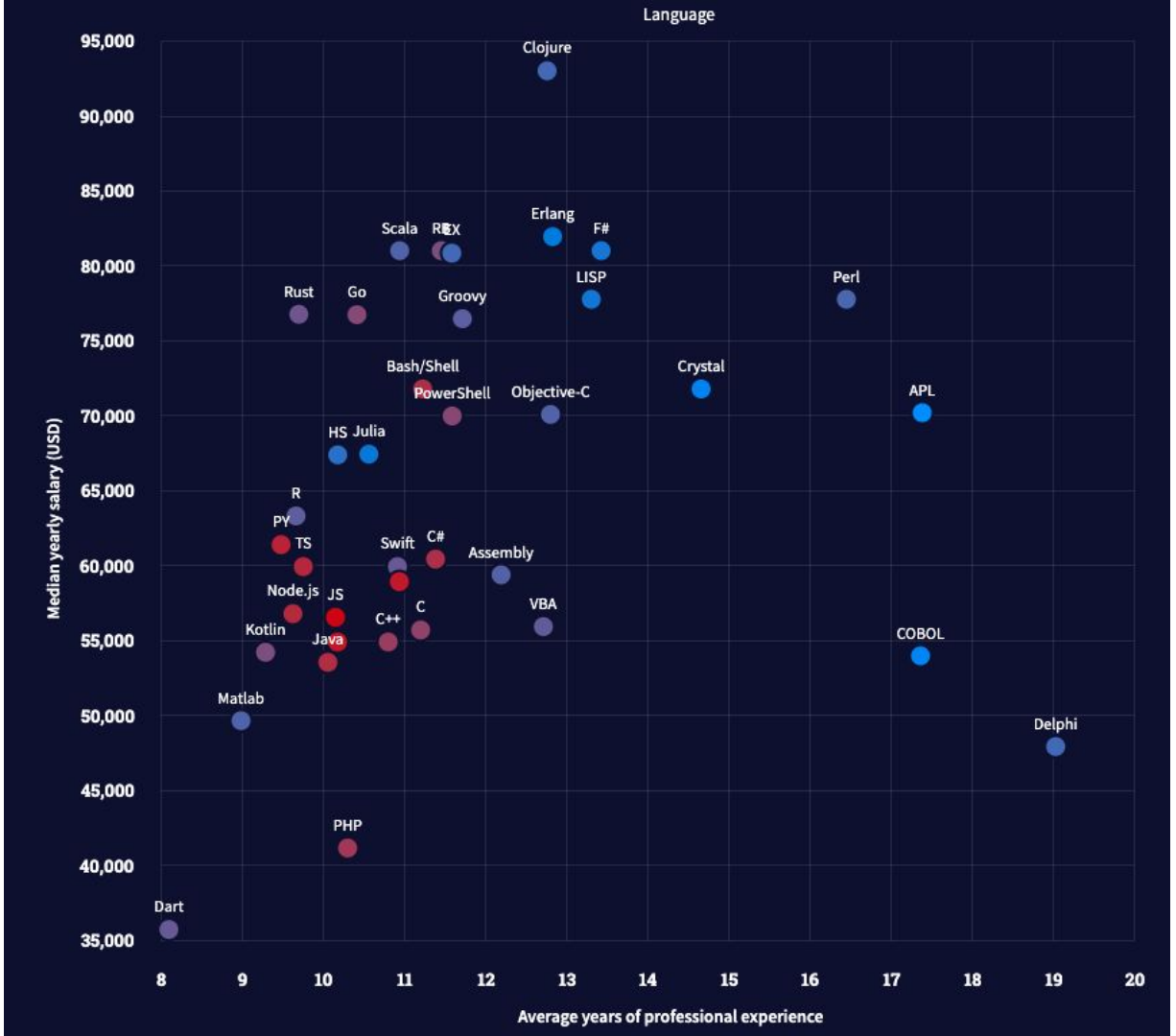
Google

Medium

Grab

tokopedia

Salary By Language



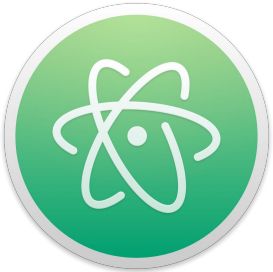
alterra
academy

Installation Golang

EDITOR (IDE)

VS Code

GoLand



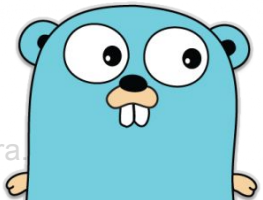
Atom

Vim

INSTALLATION

Via File Archive (Mac, Linux, & Windows)

- Go to https://golang.org/dl/ and download the latest version
- Follow Instruction Installation



alterra academy

Via Terminal ( )

-terminal

```
$ brew update
$ brew install golang
```

Via Terminal ( )

-terminal

```
curl -O
https://storage.googleapis.com/golang/go1.11.2.linux-amd64.tar.gz

tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

GO WORKSPACE

alterra academy

-terminal

```
workspace/
  bin/
  pkg/
  src/
```

bin: *contains the binary executables.*

pkg: *contains Go package archives.*

src: *contains Go source files.*

LIVE

alterra academy

GO MODULES

In Go Version > 1.11

What if we create projects **outside** **GOPATH**?

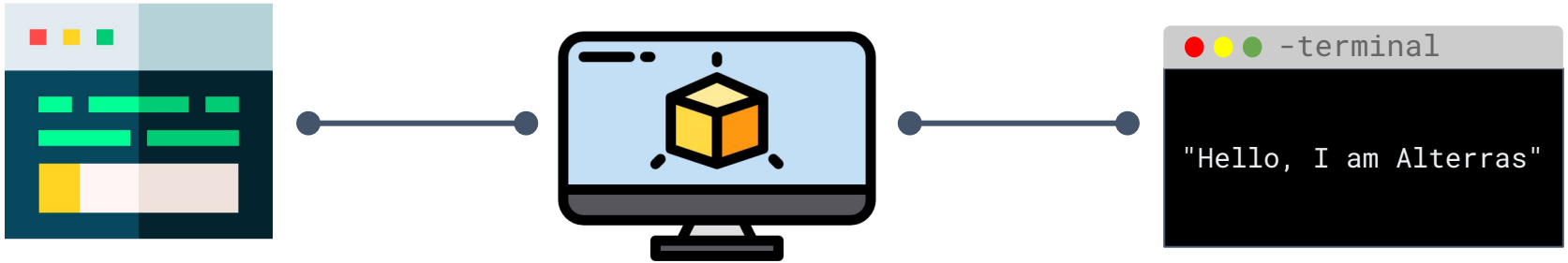
1. Create project outside **GOPATH**.
2. Open Terminal in set directory to this project
3. Run command!

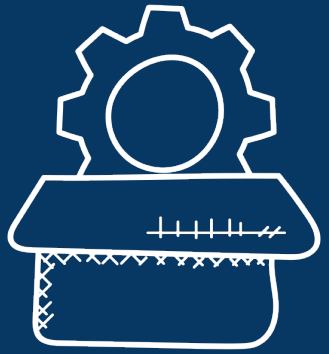
-terminal

```
$ go mod init
```

LIVE

COMPILING PROCESS





COMPILING & RUNNING

alterra academy

-code editor

```
package main

import "fmt"

func main() {
  fmt.Println("Hello, I am Programmer")
}
```

-code editor

```
$ go build main.go
$ ./main
```

OUTPUT:

```
Hello, I am Programmer
```

LIVE

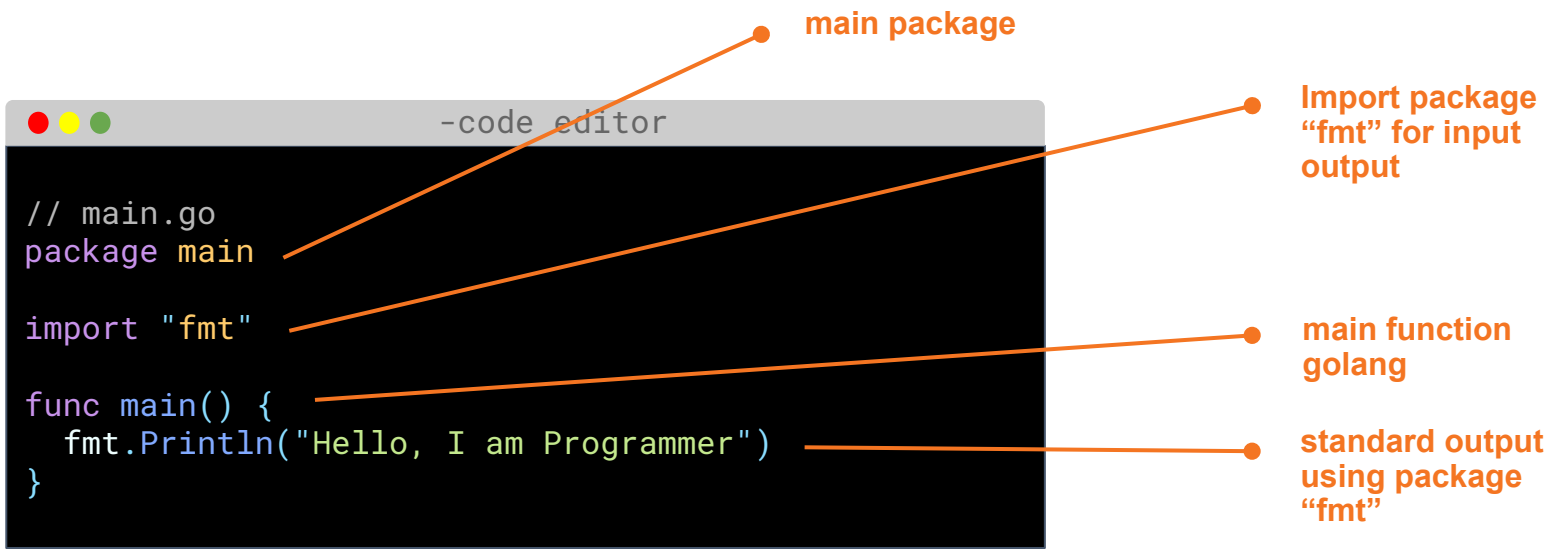
alterra academy

COMMAND GO TERMINAL

Command	Function
go run	Execute the program with extension **.go**
go build	Compile program files.
go install	Like go build and continue the install process.
go test	For testing with suffix **_test.go**
go get	download the Go package.

LIVE

MAIN SYNTAX



LIVE

Package “fmt”

alterra academy

Output

```
fmt.Printf()
fmt.Println()
fmt.Sprintf()
etc
```

Scanning

```
fmt.Scanln()
```

Format Verb

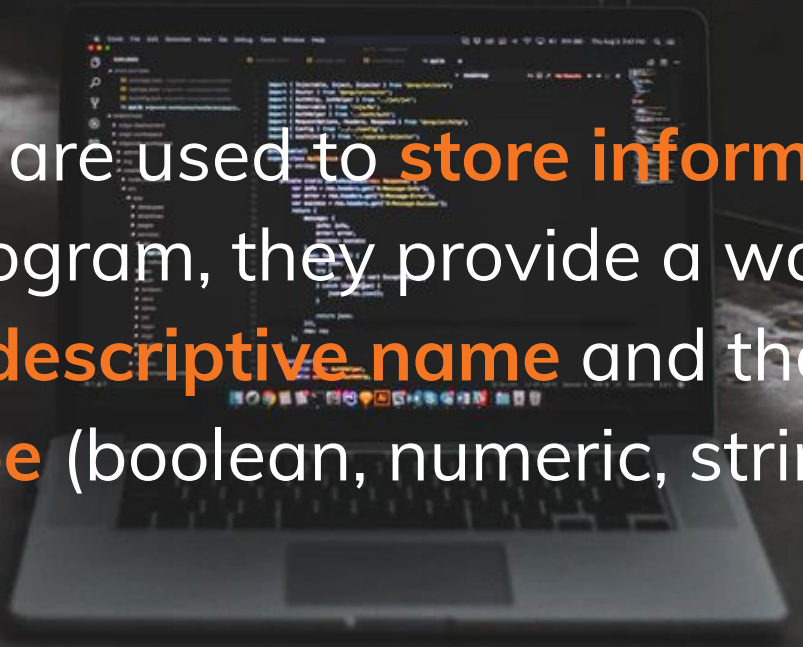
```
%T,%v,%s,%q,etc.
```

alterra
academy

Variables & Types

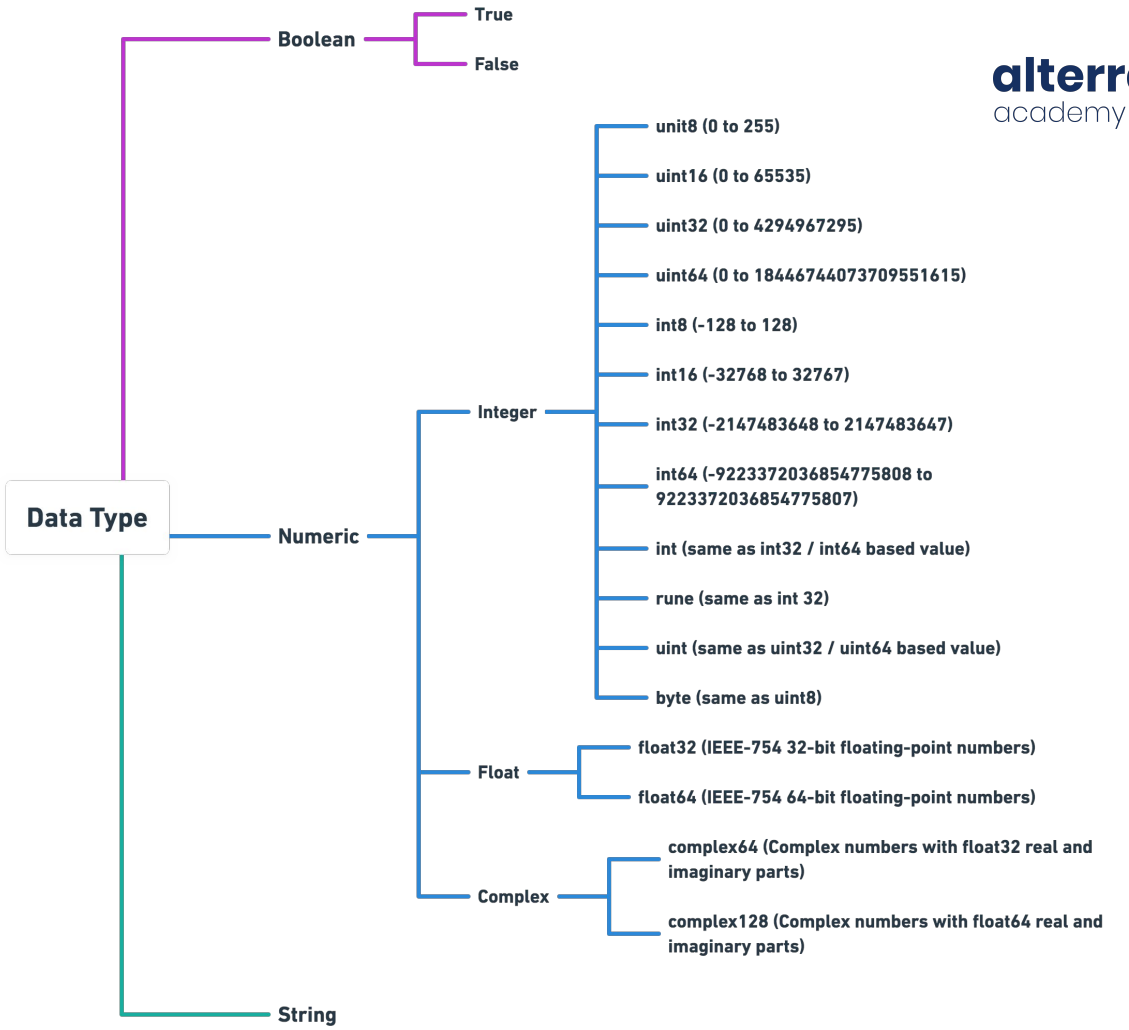
alterra academy

WHAT IS VARIABLES ?

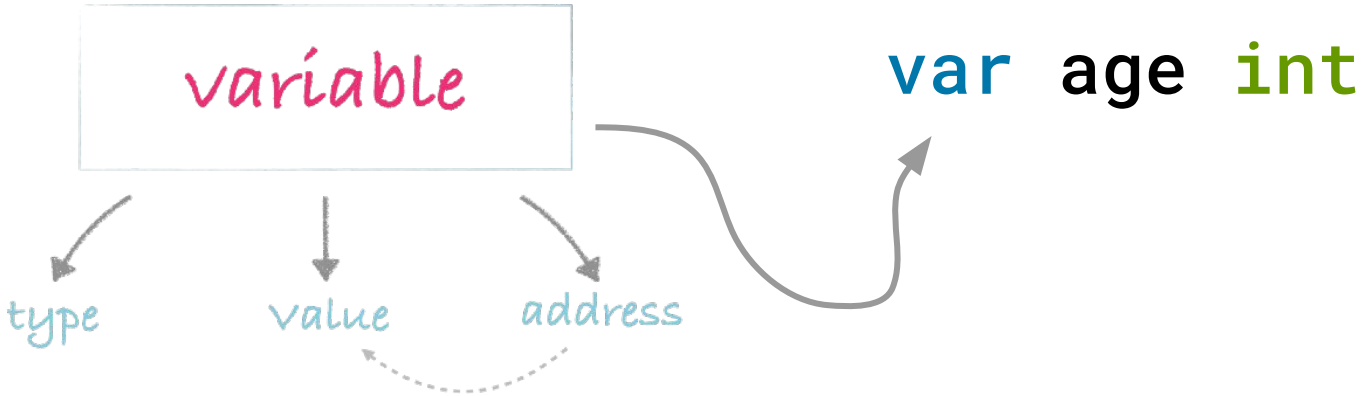


Variables are used to **store information** in a computer program, they provide a way of labeling data with a **descriptive name** and they have **data type** (boolean, numeric, string).

DATA TYPE IN GOLANG



VARIABLE DECLARATION



alterra academy

TYPE DECLARATION

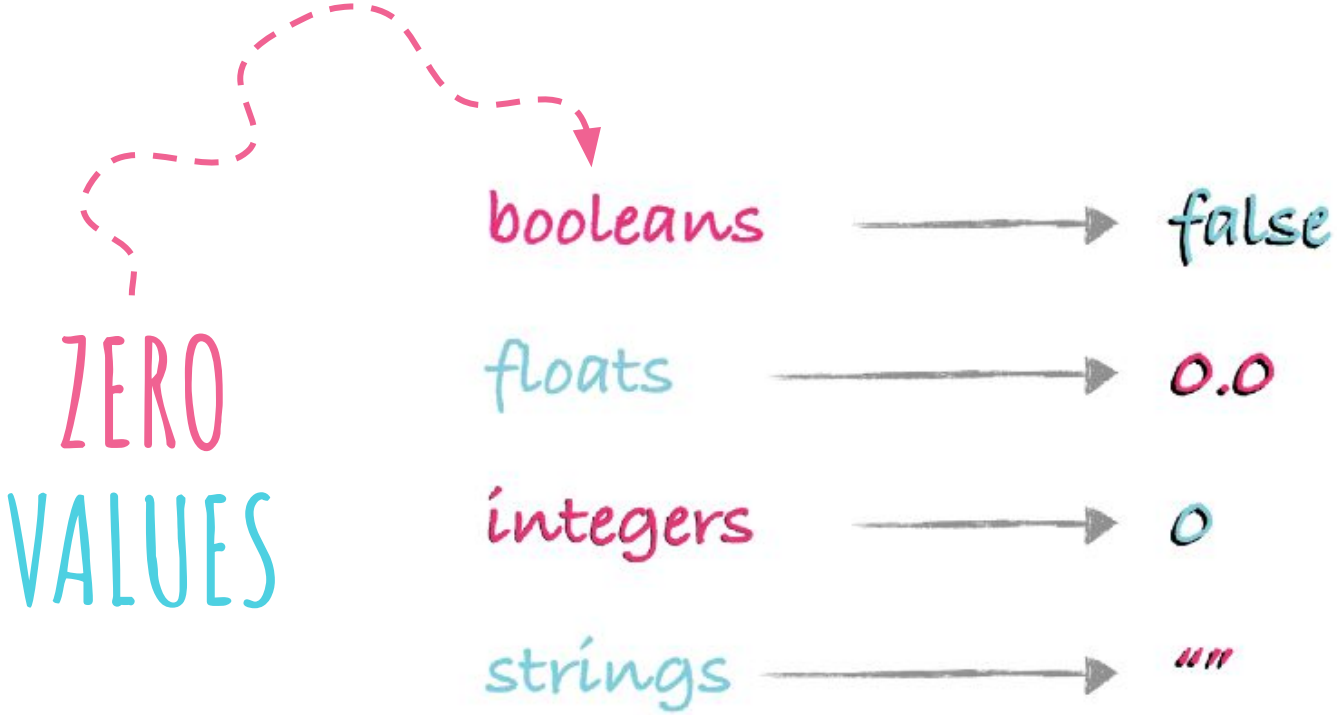
Long

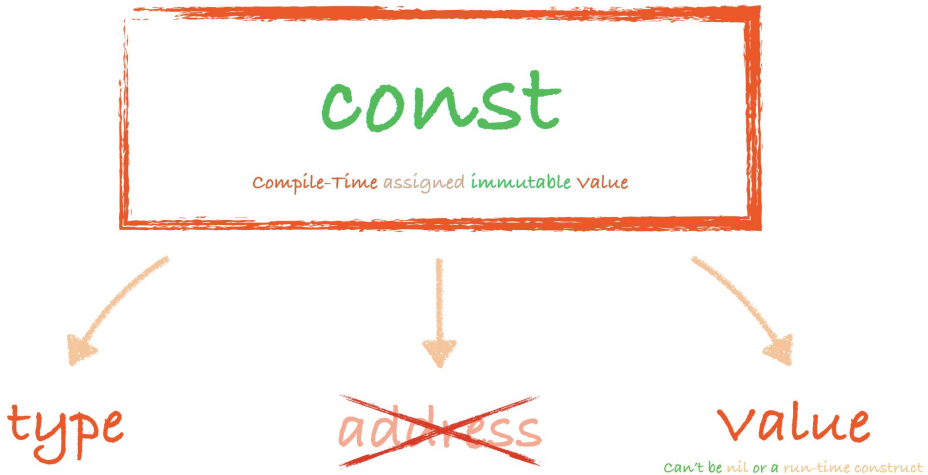
```
var <variable_name> <type_data>
    └── var name string

var <variable_name> <type_data> = <value>
    └── var name string = “moryku”

var <list_variable_name> <type_data>
    └── var name, gender string

var <list_variable_name> <type_data> = <value>
    └── var name, gender string = “moryku”, ”L”
```

Short

```
<variable_name> := <value>
    └── name := “moryku”
```

LIVE


ZERO VALUES
booleans
false
floats
0.0
integers
0
strings
""


const
Compile-Time assigned immutable value
type
address
value
Can't be nil or a run-time construct

alterra academy

Single Constants

```
const Pi float64 = 3.14
```

declare — name — type — value (64 bit precision)

Multiple Constants

```
const (
    Pi float64 = 3.14
    Pi2
    Age = 10
)
```

- `Pi`: declares a float64 constant with a value of 3.14
- `Pi2`: repeats the type; repeats the value; declares the same float64 constant with a value of 3.14
- `Age = 10`: declares an int constant with a value of 10

Example </> Variable declaration

alterra academy

-code editor

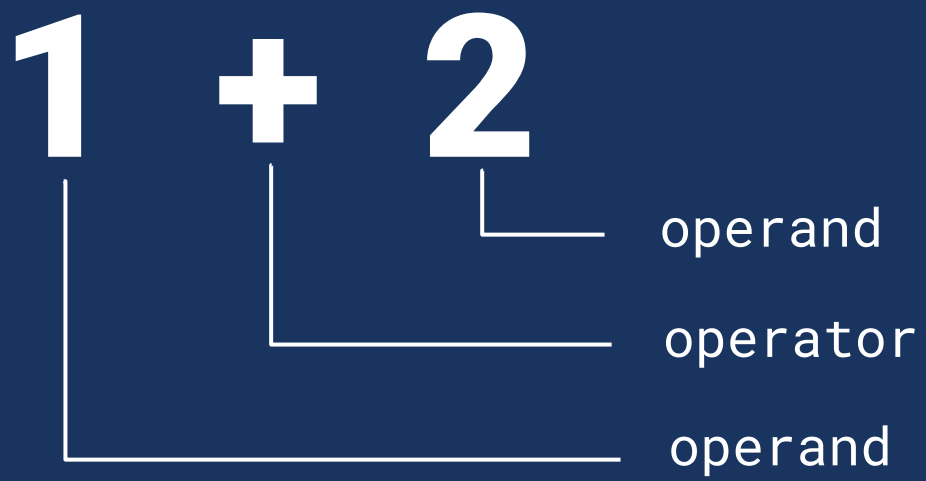
```go
package main

import "fmt"

func main() {
 // declaration boolean with assignment
 var isLogin bool = false
 fmt.Println(isLogin)
 // declaration numeric integer
 var prime int = 7
 fmt.Println(prime)
 // declaratio decimal number
 var decimal float64 = 45.6
 fmt.Println(decimal)
 // declaration string
 var hello string = "Hello World!"
 fmt.Println(hello)
 // declaration const
 const pi = 3.14159265358979323846
 fmt.Println(pi)
}
```

LIVE

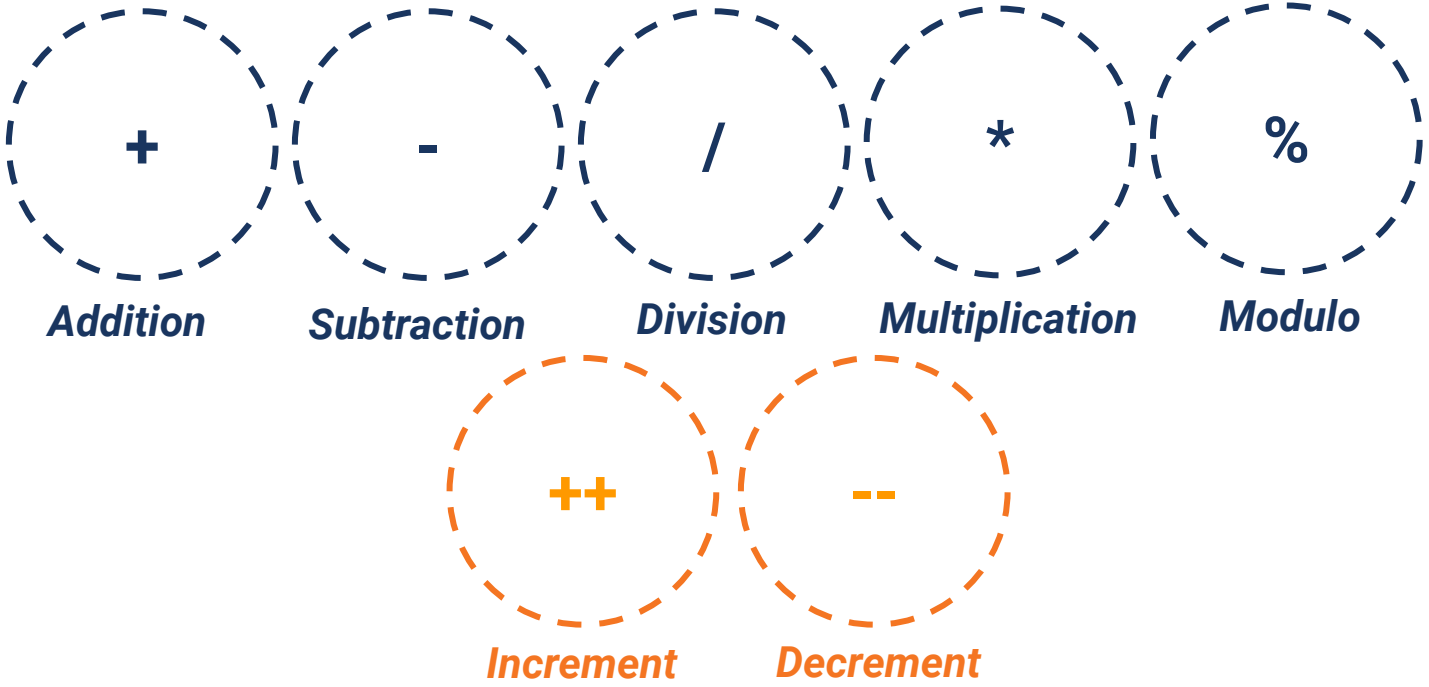
alterra
academy

Operator

alterra
academy



ARITHMETIC OPERATORS



Example </> Using Arithmetic Operator

alterra academy

-code editor

```go
package main

import "fmt"

func main() {
  // number operation
  a := 10
  t := 15
  L := (a * t) / 2
  fmt.Println(L)

  // string operation
  helloworld := "hello" + " " + "world"
  fmt.Println(helloworld)
}
```

OPERATOR IN GOLANG

alterra academy

- Arithmetic
  - ex : `+, -, *, /, %, ++, --`
- Comparison
  - ex : `==, !=, >, <, >=, <=`
- Logical
  - ex : `&&, ||, !`
- Bitwise
  - ex : `&, |, ^, <<, >>`
- Assignment
  - ex : `=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=`
- Miscellaneous
  - ex : `&, * (Pointer)`

Example </>

alterra academy

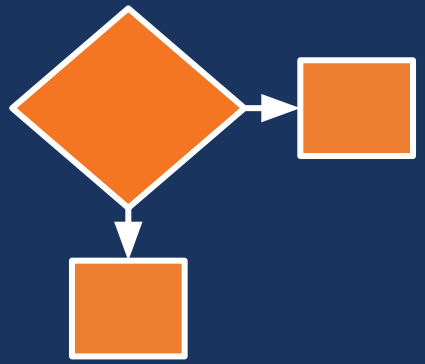
-code editor

```go
package main
import "fmt"

func main() {
  a := 1 * 2  // Arithmetic
  b := a > 0  // Relational & Assignment
  c := a >> 1 // Bitwise
  d := &c     // Miscellaneous
  fmt.Println(a, b, c, d)
}
```

command line

```
2 true 1 0xc000084000
```

alterra
academy



CONTROL STRUCTURES BRANCHING & LOOPING

alterra
academy

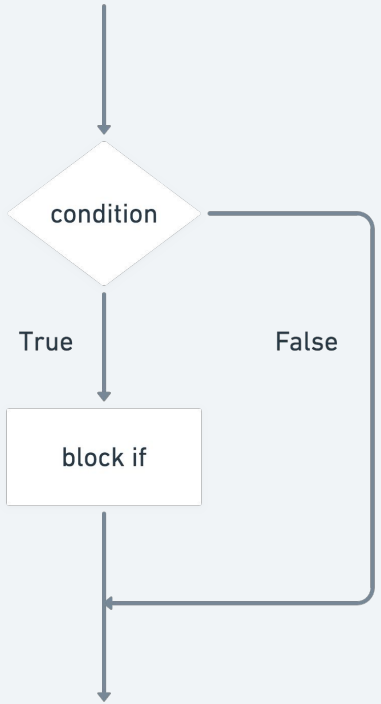
CONTROL STRUCTURES

There are only a few **control structures** in Go.
to branching we use **if** and **switch**,
to write loops we use the **for** keyword.

IF STATEMENT WITH COMPARISON

-code editor

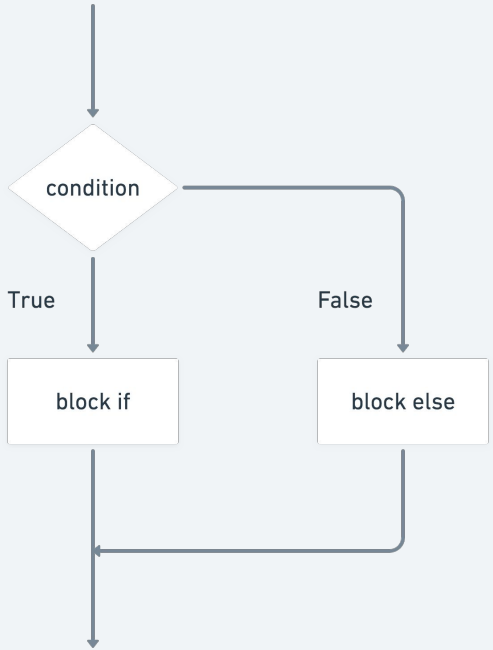
```
var myAge = 17

if myAge == 5 {
  fmt.Println("You too young")
}
if (myAge == 17) {
  fmt.Println("So Sweet")
}
if myAge >= 17 && myAge <= 30 {
  fmt.Println("My Age is between 17 and 30")
}
if dadAge := 9; dadAge < myAge {
  fmt.Println(dadAge)
}
```



IF ELSE STATEMENT

-code editor

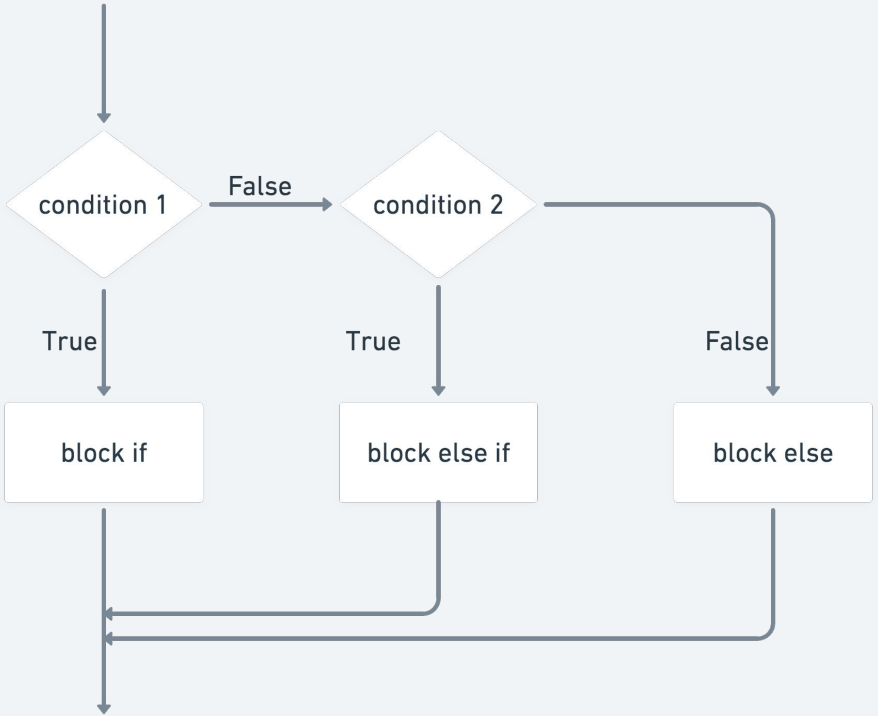
```
number := 4
if number%2 == 0 {
   fmt.Println("Genap")
} else {
   fmt.Println("Ganjil")
}
```



IF, ELSE IF, ELSE

-code editor

```
hour := 15
if hour < 12 {
  fmt.Println("Selamat Pagi")
} else if hour < 18 {
  fmt.Println("Selamat Sore")
} else {
  fmt.Println("Selamat Malam")
}
```



NESTED IF STATEMENT

-code editor

```
var v1 int = 400
var v2 int = 700

if v1 == 400 {
    if v2 == 700 {
      fmt.Printf("Value of v1 is 400 and v2 is 700\n" );
    }
}
```

SWITCH </>

```
switch expression {
    case value1:
        statement(s)
    case value2:
        statement(s)
    default:
        statement(s)
}
```

-code editor

```
var today int = 2
switch today {
    case 1:
        fmt.Printf("Today is Monday")
    case 2:
        fmt.Printf("Today is Tuesday")
    default:
        fmt.Printf("Invalid Date")
}
```

command line

```
Today is Tuesday
```

SWITCH WITHOUT EXPRESSION

alterra academy

```
switch {
    case expression==value1:
        statement(s)
    case expression==value2:
        statement(s)
    default:
        statement(s)
}
```

-code editor

```
var today int = 2
switch {
    case today == 1:
        fmt.Printf("Today is Monday")
    case today == 2:
        fmt.Printf("Today is Tuesday")
    default:
        fmt.Printf("Invalid Date")
}
```

command line

```
Today is Tuesday
```

FALLTHROUGH >_

In Go the control comes out of the switch statement immediately after a case is executed. A fallthrough statement is used to transfer control to the first statement of the case that is present immediately after the case which has been executed.

alterra academy

-code editor

```
value := 42
switch value {
case 100:
  fmt.Println(100)
  fallthrough
case 42:
  fmt.Println(42)
  fallthrough
case 1:
  fmt.Println(1)
  fallthrough
default:
  fmt.Println("default")
}
```

command line

```
42
1
default
```

alterra academy

LOOPS

statement which allows code to be repeatedly executed.

```
for init; condition; post {
  // run commands till condition is true
}
```

-code editor

```go
package main
import "fmt"
func main() {
  for i := 0; i < 5; i++ {
    fmt.Println(i)
  }
}
```

command line

```
0
1
2
3
4
```

WHILE

For is Go's "while"

At that point you can drop the semicolons: C's while is spelled for in Go.

alterra academy

-code editor

```
package main

import "fmt"

func main() {
  sum := 1
  for sum < 10 {
    sum += sum
  }
  fmt.Println(sum)
}
```

command line

```
16
```

Continue & break

alterra academy

-code editor

```go
package main
import "fmt"
func main() {
  for i := 0; i < 5; i++ {
    if i == 1 {
      continue
    }
    if i > 3 {
      break
    }
    fmt.Println(i)
  }
}
```

command line

```
0
2
3
```

Loops Over String

command line

```
character H starts at byte position 0
character e starts at byte position 1
character l starts at byte position 2
character l starts at byte position 3
character o starts at byte position 4
-----
H
e
l
l
o
```

-code editor

```go
package main

import (
  "fmt"
)

func main() {
  sentence := "Hello";

  for i := 0; i < len(sentence); i++ {
    fmt.Println(string(sentence[i]))
  }

  fmt.Println("-----")

  for pos, char := range sentence {
      fmt.Printf("character %c starts at byte position %d\n", char, pos)
  }
}
```

Advance looping (1)

alterra academy

-code editor

```go
package main

import "fmt"

func main() {
 N := 5
 for i := 0; i < N; i++ {
   for j := 0; j < N; j++ {
     fmt.Print("*")
   }
   fmt.Println()
 }
}
```

command line

```
*****
*****
*****
*****
*****
```

Advance looping (2)

alterra academy

-code editor

```go
package main

import "fmt"

func main() {
 N := 5
 for i := 0; i < N; i++ {
   for j := 0 ; j <= i; j++ {
     fmt.Print("*")
   }
   fmt.Println()
 }
}
```

command line

```
*
**
***
****
*****
```

“Talk is cheap. Show me the code.”

~ Linus Torvalds