Variables

```
// declare variable
var a string
var a string = "something"
// type is inferred:
var a = "something"
a := "something"
a, b := "one", "two"
// map
var a map[string]string
a := make(map[string]string)
a := map[string]string{"k": "v"}
// array
var d [2]int = [2]int{1,2}
// slice
var a []string
a := make([]string, 5)
a := make([]string, 5, 10)
// multiple variables:
var (
  a string
  b int
)
// struct
type myStruct struct {
   c string
}
var a myStruct
a := myStruct{c: "a string"}
a := myStruct{"a string"}
a.c = "a string"
```

Types

```
bool
string
byte (alias for uint8)
rune (alias for int32)
float32 float64
int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64
uintptr
complex64 complex128
```

Conditions

```
Comparison: <  >  <=  >=  ==  !=
Logical: ||  && !
if a >= 10 && b >= 10 {
} else if !(a < 5) {
}  else {
}
switch a {
case 5:
default: // optional
}
```

Operators

```
a := 5 + 5
a += 5 // same as a = a + 5
b := a >= 5 // returns boolean
b++ // increase b by one
b--  // decrease b by one
```

Slices

```
a := []int{}
a := []int{1,2,3,4,5,6,7}
fmt.Printf("%v", a[0:2]) // [1 2]
fmt.Printf("%v", a[5:len(a)]) // [6 7]
```

Functions

```
func returnTwo() (bool, bool) { ... }
func myFunc(b int, c int64) bool {
    return true
}
func myFunc(b, c int) { ... }
// assigning function to a variable:
a := func (a int) int { return a + 1 }
fmt.Printf("%v", a(1))
```

Loops

```
for i := 0; i < 5; i++ { ... }
for { ... } // indefinite loop
for {
  if a == 5 { break } // break loop
}
for {
  if a == 5 { continue } // next iter.
}
// range keyword
for k, v := range a {
   // k will be int for array/slice
   // k will be key for map
   // v will be value
}
```

Methods

```
type myStruct struct {
   c string
}
func (m myStruct) method() {
  m.c = "newstr" // will not update
}
func (m *myStruct) method() {
  m.c = "newstr" // will update
}
a := MyStruct{}
b := &MyStruct{}
```

Interfaces

```
type myIface interface {
  method()
  method2(a int) (bool, error)
}
```

Various

```
// constants (fixed value)
const a int = 5
// run hello() in goroutine
go hello()
```

Pointers

```
a := "123"
b := &a // pointer to a
fmt.Println(b) // 0xc00001c030
fmt.Println(*b) // "123"
*b = "456" // change a through pointer b
```

Edward Viaene v2022.11.16