GO

Variables

```
  // Outside a function, every statement begins with a keyword (var, func,
and so on) and so the := construct is not available.
  var a int
  var a, b int
  var i,j int = 1, 2


  var x int = 3
  // Inside a function, the := short assignment statement can be used in
place of a var declaration with implicit type.
  y := 3
```

Types

Types conversion

The expression **T(v)** converts the value **v** to the type **T**. Unlike in C, in Go assignment between items of different type requires an explicit conversion.

```
i := 42
f := float64(i)
u := uint(f)
```

Type assertions

```
t := i.(T)
```

This statement asserts that the interface value **i** holds the concrete type **T** and assigns the underlying **T** value to the variable t. If **i** does not hold a **T**, the statement will trigger a panic. To test whether an interface value holds a specific type, a type assertion can return two values: the underlying value and a boolean value that reports whether the assertion succeeded.

```
t, ok := i.(T)
```

```
if t, ok := i.(T); ok {
...
```

```
}
```

method names

- Method names written from upper-case letter are exported form module
- to write internal methods (like static function in c) start function name from lower case

closures

```
func main() {
  add := func(x, y int) int {
    return x + y
  }
  fmt.Println(add(1,1))
}
```

```
func makeEvenGenerator() func() uint {
  i := uint(0)
  return func() (ret uint) {
    ret = i
    i += 2
    return
  }
}
```

Concatenate slices

```
    moreBytes := []byte{1,2,3,4}
    bytes := []byte{2}
    bytes = append(bytes, 4)
    bytes = append(bytes, moreBytes...)
```