GoLang Cheat Sheet

by tahir24434 via cheatography.com/67694/cs/24941/

Packages

Packages in Go supports modularity, encapsulation, separate compilation, and reuse.

Package declaration at top of every source file

Standalone executables program are in package main

If an entity is declared within a function, it is local to that function.

If declared outside of a function, however, it is visible in all files of the package to which it belongs

The case of the first letter of a name determines its visibility across package boundaries.

  Upper case identifier: Exported i.e visible and accessible outside of its own package.

  Lower case identifier: private (not accessible from other packages)

Pointers

```
var x int = 11
```

```
/*
*int is integerPointer type.
‘p’ will contain the address of an integer variable.
You can also say that p points to an int variable.
*/
var p *int
```

```
// Expression &var (address of var) yields a pointer to a variable.
p = &x // will contain address of x
```

```
// Expression *p points to the variable whose address p contains. *p is an alias for x.
fmt.Println(*p)
```

Arrays

Arrays and Structs are aggregate type
Arrays are homogeneous
Array is fixed length sequence of zero or more elements of particular type.

```
var a[3] int       // Array of 3 integers
```

```
var a[3]int = [3]int{1, 2, 3} // use an array literal to initialize an array with a list of values
```

```
a[len(a)-1]        // Print last element
```

```
q := [...]int{1, 2, 3} // with ellipsis ... , array length is determined by the number of initializer
```

Naming Convention

Declarations

There are four major kinds of declarations: **var, const, type, func**

var

```
var name type = expression
// Either the type or the =expression part may be omitted, but not both
// If the type is omitted, it is determined by the initializer expression.
If the expression is omitted, the initial value is the zero value for the type, which is 0 for numbers, false for booleans.
```

```
var foo int = 42 // declare and init. var name type = expression
```

```
var sep string           // implicit initialize
```

```
s, sep := "", ""         // Short variable declaration. name := expression
```

```
p := new(int) // p, of type *int, points to an unnamed int variable
// new(T) creates unnamed variable of type T, initialize it to the zero value of T and returns its address.
```

const

A constant is an identifier for a fixed value. The value of a variable can vary, but the value of a constant must remain constant.

```
const constant = "This is a constant"
```

```
const a float64 = 3.14
```

Function Declaration

A function declaration has a name, a list of parameters, an optional list of results

```
// function with params
func getFullName(firstName string, lastName string) {}
```

```
// Multiple params of the same type
func getFullName(firstName, lastName string) {}
```

```
// Can return type declaration
func getId() int
```

```
// Can return multiple values at once
func person() (int, string) {
  return 23, "vinay"
}
```

```
// Can return multiple named results
func person() (age int, name string) {
  age = 23 name = "vinay"
  return
}
var age, name = person()
```

```
// Can return function
func person() func() (string,string) {
  area:=func() (string,string) {
    return "street", "city"
  }
return area
}
```

Loops

// a name begins with a letter or an underscore and may have any number of additional letters, digits, and underscores
type playerScore struct // Use CamelCase
const MaxTime int
var fileClosed bool // Use the complete words in larger scopes
var arg []string // Use fewer letters in smaller scopes
var localAPI string // Use All caps for acronym

// There only for, no while, no until
for i := 1; i < len(os.Args); i++ {}   // initialization; condition; post {}
for condition {}   // While loop
// 'range' produces a pair of values: the index and the value of the element at that index. '_' is called blank identifier. for _, arg := range os.Args[1:]