DEVHINTS.IO

Edit

Go cheatsheet

Introduction

A tour of Go
(tour.golang.org)

Go repl
(repl.it)

Golang wiki
(github.com)

Hello world

hello.go

```
package main

import "fmt"

func main() {
  message := greetMe("world")
  fmt.Println(message)
}

func greetMe(name string) string {
  return "Hello, " + name + "!"
```

Constants

```
const Phi = 1.618
const Size int64 = 1024
const x, y = 1, 2
const (
  Pi = 3.14
  E  = 2.718
)
const (
  Sunday = iota
  Monday
  Tuesday
  Wednesday
  Thursday
  Friday
  Saturday
)
```

Constants can be character, string, boolean, or numeric values.

See: Constants

Basic types

Strings

```
str := "Hello"
```

```
str := `Multiline
string`
```

Strings are of type `string`.

Numbers

Typical types

```
num := 3          // int
num := 3.         // float64
num := 3 + 4i     // complex128
num := byte('a')  // byte (alias f
```

Other types

```
var u uint = 7        // uint (uns
var p float32 = 22.7  // 32-bit fl
```

Pointers

```
func main () {

  fmt.Println("Value is", b)
}
```

```
func getPointer () (myPointer *int) {
  a := 234

}
```

```
a := new(int)
```

Pointers point to a memory location of a variable. Go is fully garbage-collected.

See: Pointers

Type conversions

```
i := 2
f := float64(i)
u := uint(i)
```

See: Type conversions

Flow control

Conditional

Statements in if

```
for count := 0; count <= 10; count++ {
  fmt.Println("My counter is at", count)
}
```

See: For loops

See: If

```
entry := []string{"Jack","John","J
for i, val := range entry {
  fmt.Printf("At position %d, the
}
```

See: For-Range loops

See: If with a short statement

Functions

Lambdas

```
  return x > 10000
}
```

Functions are first class objects.

Multiple return types

```
a, b := getMessage()
```

```
func getMessage() (a string, b str

}
```

Packages

Importing

```
import "fmt"
import "math/rand"
```

```
import (
  "fmt"        // gives fmt.Println
  "math/rand"  // gives rand.Intn
)
```

Both are the same.

See: Importing

Aliases

```
r.Intn()
```

Packages

```
package hello
```

Every package file has to start with packa

Concurrency

Goroutines

```
func main() {
  // A "channel"


  // Start concurrent routines




  // Read 3 results
  // (Since our goroutines are concurrent,
  // the order isn't guaranteed!)

}
```

```
func push(name string, ch chan string) {
  msg := "Hey, " + name

}
```

Buffered channels

```
ch <- 1
ch <- 2
ch <- 3
// fatal error:
// all goroutines are asleep - dea
```

Buffered channels limit the amount of me

See: Buffered channels

WaitGroup

```
func main() {


  for _, item := range itemList {
```

Channels are concurrency-safe communication objects, used in g

See: Goroutines, Channels

```
    // Increment WaitGroup Counter
    wg.Add(1)
    go doOperation(&wg, item)
  }
  // Wait for goroutines to finish
  wg.Wait()

}
```

```
func doOperation(wg *sync.WaitGrou

  // do operation on item
  // ...
}
```

A WaitGroup waits for a collection of gor
number of goroutines to wait for. The go

Error control

Defer

```
func main() {

  fmt.Println("Working...")
}
```

Defers running a function until the surrounding function returns. The arguments are evaluated immediately, but the function call is not ran until later.

See: Defer, panic and recover

Deferrin

```
func ma


  fmt.P
}
```

Lambda

```
func ma
  var d


  fmt.P
  d = t
}
```

The defe

Structs

Defining

```
func main() {
  v := Vertex{1, 2}
  v.X = 4
  fmt.Println(v.X, v.Y)
}
```

See: Structs

Literals

```
v := Vertex{X: 1, Y: 2}
```

```
// Field names can be omitted
v := Vertex{1, 2}
```

```
// Y is implicit
v := Vertex{X: 1}
```

You can also put field names.

Methods

Receivers

```
type Vertex struct {
  X, Y float64
}
```

```
  return math.Sqrt(v.X * v.X + v.Y * v.Y)
}
```

```
v := Vertex{1, 2}
v.Abs()
```

There are no classes, but you can define functions with receivers.

Mutatio

```
  v.X =
  v.Y =
}
```

```
v := Ve
v.Scale
// `v`
```

By defin

See: Poi

See: Methods

Interfaces

A basic interface

```
type Shape interface {
  Area() float64
  Perimeter() float64
}
```

Struct

```
type Re
  Lengt
}
```

Struct R

Methods

```
func (r Rectangle) Area() float64 {
  return r.Length * r.Width
}

func (r Rectangle) Perimeter() float64 {
  return 2 * (r.Length + r.Width)
}
```

The methods defined in Shape are implemented in Rectangle.

Interfac

```
func ma
  var r
  fmt.P
}
```

References

Official resources

A tour of Go
(tour.golang.org)

Golang wiki
(github.com)

Effective Go
(golang.org)

Other li

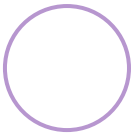
Go by E
(gobyex

Aweso
(aweson

JustFo
(youtub

Style C
(github.

Search 358+ cheatsheets



Over 358 curated cheatsheets, by developers for developers.

Devhints home

Other C-like cheatsheets

- C Preprocessor cheatsheet
- C# 7 cheatsheet

Top cheatsheets

- Elixir cheatsheet
- ES2015+ cheatsheet
- React.js cheatsheet
- Vimdiff cheatsheet
- Vim cheatsheet
- Vim scripting cheatsheet