Golang for Node.js Developers

> *Examples of Golang examples compared to Node.js for learning*

This guide full of examples is intended for people learning Go that are coming from Node.js, although the vice versa can work too. This is not meant to be a complete guide and it is assumed that you've gone through the Tour of Go tutorial. This guide is meant to be barely good enough to help you at a high level understand how to do *X* in *Y* and doing further learning on your own is of course required.

Contents

- switch
  - Node.js
  - Go
- arrays
  - Node.js
  - Go
- uint8 arrays
  - Node.js
  - Go
- array iteration
  - Node.js
  - Go
- array sorting
  - Node.js
  - Go
- buffers
  - Node.js
  - Go
- maps
  - Node.js
  - Go
- objects
  - Node.js
  - Go
- functions
  - Node.js
  - Go
- default values
  - Node.js
  - Go
- destructuring
  - Node.js
  - Go
- spread operator
  - Node.js
  - Go
- rest operator
  - Node.js
  - Go
- swapping
  - Node.js
  - Go
- classes
  - Node.js
  - Go
- generators
  - Node.js

    - Go
- datetime
    - Node.js
    - Go
- timeout
    - Node.js
    - Go
- interval
    - Node.js
    - Go
- IIFE
    - Node.js
    - Go
- files
    - Node.js
    - Go
- json
    - Node.js
    - Go
- big numbers
    - Node.js
    - Go
- promises
    - Node.js
    - Go
- async/await
    - Node.js
    - Go
- streams
    - Node.js
    - Go
- event emitter
    - Node.js
    - Go
- errors
    - Node.js
    - Go
- try/catch
    - Node.js
    - Go
- exceptions
    - Node.js
    - Go
- regex
    - Node.js
    - Go
- exec (sync)

    - Node.js
    - Go
- exec (async)
    - Node.js
    - Go
- tcp server
    - Node.js
    - Go
- udp server
    - Node.js
    - Go
- http server
    - Node.js
    - Go
- url parse
    - Node.js
    - Go
- gzip
    - Node.js
    - Go
- dns
    - Node.js
    - Go
- crypto
    - Node.js
    - Go
- env vars
    - Node.js
    - Go
- cli args
    - Node.js
    - Go
- cli flags
    - Node.js
    - Go
- stdout
    - Node.js
    - Go
- stderr
    - Node.js
    - Go
- stdin
    - Node.js
    - Go
- modules
    - Node.js
    - Go

Examples

All sample code is available in examples/

comments

Node.js

```
// this is a line comment

/*
 this is a block comment
*/
```

Go

```
package main

func main() {
    // this is a line comment

    /*
       this is a block comment
    */
}
```

back to top

printing

Node.js

```
console.log('print to stdout')
console.log('format %s %d', 'example', 1)
console.error('print to stderr')
```

Output

```
print to stdout
format example 1
print to stderr
```

Go

```
package main

import (
    "fmt"
    "os"
)

func main() {
    fmt.Println("print to stdout")
    fmt.Printf("format %s %v\n", "example", 1)
    fmt.Fprintf(os.Stderr, "print to stderr")
}
```

Output

```
print to stdout
format example 1
print to stderr
```

back to top

logging

Node.js

```
console.log((new Date()).toISOString(), 'hello world')
```

Output

```
2021-04-11T20:55:07.451Z hello world
```

Go

```
package main

import "log"

func main() {
    log.Println("hello world")
}
```

Output

```
2021/04/11 13:55:07 hello world
```

(Package `log` writes to standard error ánd prints the date and time of each logged message)

⬆ back to top

variables

Node.js

```
// function scoped
var foo = 'foo'

// block scoped
let bar = 'bar'

// constant
const qux = 'qux'
```

Go

(variables are block scoped in Go)

```
package main

func main() {
    // explicit
    var foo string = "foo"

    // type inferred
    var bar = "foo"

    // shorthand
    baz := "bar"

    // constant
    const qux = "qux"
}
```

⬆ back to top

interpolation

Node.js

```
const name = 'bob'
const age = 21
const message = `${name} is ${age} years old`

console.log(message)
```

Output

```
bob is 21 years old
```

Go

```go
package main

import "fmt"

func main() {
    name := "bob"
    age := 21
    message := fmt.Sprintf("%s is %d years old", name, age)

    fmt.Println(message)
}
```

Output

```
bob is 21 years old
```

[back to top]

types

Node.js

```javascript
// primitives
const myBool = true
const myNumber = 10
const myString = 'foo'
const mySymbol = Symbol('bar')
const myNull = null
const myUndefined = undefined

// object types
const myObject = {}
const myArray = []
const myFunction = function() {}
const myError = new Error('error')
const myDate = new Date()
const myRegex = /a/
const myMap = new Map()
const mySet = new Set()
const myPromise = Promise.resolve()
const myGenerator = function *() {}
const myClass = class {}
```

Go

```go
package main

func main() {
    // primitives
    var myBool bool = true
    var myInt int = 10
    var myInt8 int8 = 10
    var myInt16 int16 = 10
    var myInt32 int32 = 10
    var myInt64 int64 = 10
    var myUint uint = 10
    var myUint8 uint8 = 10
    var myUint16 uint16 = 10
    var myUint32 uint32 = 10
    var myUint64 uint64 = 10
    var myUintptr uintptr = 10
    var myFloat32 float32 = 10.5
    var myFloat64 float64 = 10.5
    var myComplex64 complex64 = -1 + 10i
    var myComplex128 complex128 = -1 + 10i
    var myString string = "foo"
    var myByte byte = 10  // alias to uint8
    var myRune rune = 'a' // alias to int32

    // composite types
    var myStruct struct{} = struct{}{}
    var myArray []string = []string{}
    var myMap map[string]int = map[string]int{}
    var myFunction func() = func() {}
    var myChannel chan bool = make(chan bool)
    var myInterface interface{} = nil
    var myPointer *int = new(int)
}
```

back to top

type check

Node.js

```go
function typeOf(obj) {
  return {}.toString.call(obj).split(' ')[1].slice(0,-1).toLowerCase()
}

const values = [
  true,
  10,
  'foo',
  Symbol('bar'),
  null,
  undefined,
  NaN,
```

```
  {},
  [],
  function(){},
  new Error(),
  new Date(),
  /a/,
  new Map(),
  new Set(),
  Promise.resolve(),
  function *() {},
  class {},
]

for (value of values) {
  console.log(typeOf(value))
}
```

Output

```
boolean
number
string
symbol
null
undefined
number
object
array
function
error
date
regexp
map
set
promise
generatorfunction
function
```

Go

```
package main

import (
    "fmt"
    "reflect"
    "regexp"
    "time"
)

func main() {
    values := []interface{}{
        true,
```

```
        int8(10),
        int16(10),
        int32(10),
        int64(10),
        uint(10),
        uint8(10),
        uint16(10),
        uint32(10),
        uint64(10),
        uintptr(10),
        float32(10.5),
        float64(10.5),
        complex64(-1 + 10i),
        complex128(-1 + 10i),
        "foo",
        byte(10),
        'a',
        rune('a'),
        struct{}{},
        []string{},
        map[string]int{},
        func() {},
        make(chan bool),
        nil,
        new(int),
        time.Now(),
        regexp.MustCompile(`^a$`),
    }

    for _, value := range values {
        fmt.Println(reflect.TypeOf(value))
    }
}
```

Output

```
bool
int8
int16
int32
int64
uint
uint8
uint16
uint32
uint64
uintptr
float32
float64
complex64
complex128
string
```

```
uint8
int32
int32
struct {}
[]string
map[string]int
func()
chan bool
<nil>
*int
time.Time
*regexp.Regexp
```

back to top

if/else

Node.js

```
const array = [1, 2]

if (array) {
  console.log('array exists')
}

if (array.length === 2) {
  console.log('length is 2')
} else if (array.length === 1) {
  console.log('length is 1')
} else {
  console.log('length is other')
}

const isOddLength = array.length % 2 == 1 ? 'yes' : 'no'

console.log(isOddLength)
```

Output

```
array exists
length is 2
no
```

Go

```
package main

import "fmt"

func main() {
    array := []byte{1, 2}
```

```
    if array != nil {
        fmt.Println("array exists")
    }

    if len(array) == 2 {
        fmt.Println("length is 2")
    } else if len(array) == 1 {
        fmt.Println("length is 1")
    } else {
        fmt.Println("length is other")
    }

    // closest thing to ternary operator
    isOddLength := "no"
    if len(array)%2 == 1 {
        isOddLength = "yes"
    }

    fmt.Println(isOddLength)
}
```

Output

```
array exists
length is 2
no
```

back to top

for

Node.js

```
for (let i = 0; i <= 5; i++) {
  console.log(i)
}
```

Output

```
0
1
2
3
4
5
```

Go

```
package main

import "fmt"
```

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

Output

```
0
1
2
3
4
5
```

back to top

while

Node.js

```js
let i = 0

while (i <= 5) {
  console.log(i)

  i++
}
```

Output

```
0
1
2
3
4
5
```

Go

(there's no *while* keyword in Go but the same functionality is achieved by using *for*)

```go
package main

import "fmt"

func main() {
    i := 0

    for i <= 5 {
        fmt.Println(i)

        i++
```

```
    }
}
```

Output

```
0
1
2
3
4
5
```

⬆ back to top

switch

Node.js

```
const value = 'b'

switch(value) {
  case 'a':
    console.log('A')
    break
  case 'b':
    console.log('B')
    break
  case 'c':
    console.log('C')
    break
  default:
    console.log('first default')
}

switch(value) {
  case 'a':
    console.log('A - falling through')
  case 'b':
    console.log('B - falling through')
  case 'c':
    console.log('C - falling through')
  default:
    console.log('second default')
}
```

Output

```
B
B - falling through
C - falling through
second default
```

Go

```
package main

import "fmt"

func main() {
    value := "b"

    switch value {
    case "a":
        fmt.Println("A")
    case "b":
        fmt.Println("B")
    case "c":
        fmt.Println("C")
    default:
        fmt.Println("first default")
    }

    switch value {
    case "a":
        fmt.Println("A - falling through")
        fallthrough
    case "b":
        fmt.Println("B - falling through")
        fallthrough
    case "c":
        fmt.Println("C - falling through")
        fallthrough
    default:
        fmt.Println("second default")
    }
}
```

Output

```
B
B - falling through
C - falling through
second default
```

back to top

arrays

Examples of slicing, copying, appending, and prepending arrays.

Node.js

```
const array = [1, 2, 3, 4, 5]
console.log(array)
```

```
const clone = array.slice(0)
console.log(clone)

const sub = array.slice(2,4)
console.log(sub)

const concatenated = clone.concat([6, 7])
console.log(concatenated)

const prepended = [-2,-1,0].concat(concatenated)
console.log(prepended)
```

Output

```
[ 1, 2, 3, 4, 5 ]
[ 1, 2, 3, 4, 5 ]
[ 3, 4 ]
[ 1, 2, 3, 4, 5, 6, 7 ]
[ -2, -1, 0, 1, 2, 3, 4, 5, 6, 7 ]
```

Go

```
package main

import "fmt"

func main() {
    array := []int{1, 2, 3, 4, 5}
    fmt.Println(array)

    clone := make([]int, len(array))
    copy(clone, array)
    fmt.Println(clone)

    sub := array[2:4]
    fmt.Println(sub)

    concatenated := append(array, []int{6, 7}...)
    fmt.Println(concatenated)

    prepended := append([]int{-2, -1, 0}, concatenated...)
    fmt.Println(prepended)
}
```

Output

```
[1 2 3 4 5]
[1 2 3 4 5]
[3 4]
[1 2 3 4 5 6 7]
[-2 -1 0 1 2 3 4 5 6 7]
```

⬆ back to top

uint8 arrays

Node.js

```
const array = new Uint8Array(10)
console.log(array)

const offset = 1

array.set([1, 2, 3], offset)
console.log(array)

const sub = array.subarray(2)
console.log(sub)

const sub2 = array.subarray(2,4)
console.log(sub2)

console.log(array)
const value = 9
const start = 5
const end = 10
array.fill(value, start, end)
console.log(array)

console.log(array.byteLength)
```

Output

```
Uint8Array [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
Uint8Array [ 0, 1, 2, 3, 0, 0, 0, 0, 0, 0 ]
Uint8Array [ 2, 3, 0, 0, 0, 0, 0, 0 ]
Uint8Array [ 2, 3 ]
Uint8Array [ 0, 1, 2, 3, 0, 0, 0, 0, 0, 0 ]
Uint8Array [ 0, 1, 2, 3, 0, 9, 9, 9, 9, 9 ]
10
```

Go

```
package main

import "fmt"

func main() {
    array := make([]uint8, 10)
    fmt.Println(array)

    offset := 1

    copy(array[offset:], []uint8{1, 2, 3})
```

```
    fmt.Println(array)

    sub := array[2:]
    fmt.Println(sub)

    sub2 := array[2:4]
    fmt.Println(sub2)

    fmt.Println(array)
    value := uint8(9)
    start := 5
    end := 10
    for i := start; i < end; i++ {
        array[i] = value
    }
    fmt.Println(array)

    fmt.Println(len(array))
}
```

Output

```
[0 0 0 0 0 0 0 0 0 0]
[0 1 2 3 0 0 0 0 0 0]
[2 3 0 0 0 0 0 0]
[2 3]
[0 1 2 3 0 0 0 0 0 0]
[0 1 2 3 0 9 9 9 9 9]
10
```

back to top

array iteration

Examples of iterating, mapping, filtering, and reducing arrays.

Node.js

```
const array = ['a', 'b', 'c']

array.forEach((value, i) => {
  console.log(i, value)
})

const mapped = array.map(value => {
  return value.toUpperCase()
})

console.log(mapped)

const filtered = array.filter((value, i) => {
  return i % 2 == 0
```

```
})

console.log(filtered)

const reduced = array.reduce((acc, value, i) => {
  if (i % 2 == 0) {
    acc.push(value.toUpperCase())
  }

  return acc
}, [])

console.log(reduced)
```

Output

```
0 'a'
1 'b'
2 'c'
[ 'A', 'B', 'C' ]
[ 'a', 'c' ]
[ 'A', 'C' ]
```

Go

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    array := []string{"a", "b", "c"}

    for i, value := range array {
        fmt.Println(i, value)
    }

    mapped := make([]string, len(array))
    for i, value := range array {
        mapped[i] = strings.ToUpper(value)
    }

    fmt.Println(mapped)

    var filtered []string
    for i, value := range array {
        if i%2 == 0 {
            filtered = append(filtered, value)
        }
    }
```

```
    fmt.Println(filtered)

    var reduced []string
    for i, value := range array {
        if i%2 == 0 {
            reduced = append(reduced, strings.ToUpper(value))
        }
    }

    fmt.Println(reduced)
}
```

Output

```
0 a
1 b
2 c
[A B C]
[a c]
[A C]
```

back to top

array sorting

Examples of how to sort an array

Node.js

```
const stringArray = ['a', 'd', 'z', 'b', 'c', 'y']
const stringArraySortedAsc = stringArray.sort((a, b) => a > b ? 1 : -1)
console.log(stringArraySortedAsc)

const stringArraySortedDesc = stringArray.sort((a, b) => a > b ? -1 : 1)
console.log(stringArraySortedDesc)


const numberArray = [1, 3, 5, 9, 4, 2, 0]
const numberArraySortedAsc = numberArray.sort((a, b) => a - b)
console.log(numberArraySortedAsc)

const numberArraySortedDesc = numberArray.sort((a, b) => b - a)
console.log(numberArraySortedDesc)

const collection = [
    {
        name: "Li L",
        age: 8
    },
    {
        name: "Json C",
```

```
        age: 3
    },
    {
        name: "Zack W",
        age: 15
    },
    {
        name: "Yi M",
        age: 2
    }
]

const collectionSortedByAgeAsc = collection.sort((a, b) => a.age - b.age)
console.log(collectionSortedByAgeAsc)

const collectionSortedByAgeDesc = collection.sort((a, b) => b.age - a.age)
console.log(collectionSortedByAgeDesc)
```

Output

```
[ 'a', 'b', 'c', 'd', 'y', 'z' ]
[ 'z', 'y', 'd', 'c', 'b', 'a' ]
[ 0, 1, 2, 3, 4, 5, 9 ]
[ 9, 5, 4, 3, 2, 1, 0 ]
[ { name: 'Yi M', age: 2 },
  { name: 'Json C', age: 3 },
  { name: 'Li L', age: 8 },
  { name: 'Zack W', age: 15 } ]
[ { name: 'Zack W', age: 15 },
  { name: 'Li L', age: 8 },
  { name: 'Json C', age: 3 },
  { name: 'Yi M', age: 2 } ]
```

Go

```
package main

import (
    "fmt"
    "sort"
)

type Person struct {
    Name string
    Age  int
}

type PersonCollection []Person

func (pc PersonCollection) Len() int {
    return len(pc)
}
```

```
func (pc PersonCollection) Swap(i, j int) {
    pc[i], pc[j] = pc[j], pc[i]
}

func (pc PersonCollection) Less(i, j int) bool {
    // asc
    return pc[i].Age < pc[j].Age
}

func main() {
    intList := []int{1, 3, 5, 9, 4, 2, 0}

    // asc
    sort.Ints(intList)
    fmt.Println(intList)
    // desc
    sort.Sort(sort.Reverse(sort.IntSlice(intList)))
    fmt.Println(intList)

    stringList := []string{"a", "d", "z", "b", "c", "y"}

    // asc
    sort.Strings(stringList)
    fmt.Println(stringList)
    // desc
    sort.Sort(sort.Reverse(sort.StringSlice(stringList)))
    fmt.Println(stringList)

    collection := []Person{
        {"Li L", 8},
        {"Json C", 3},
        {"Zack W", 15},
        {"Yi M", 2},
    }

    // asc
    sort.Sort(PersonCollection(collection))
    fmt.Println(collection)
    // desc
    sort.Sort(sort.Reverse(PersonCollection(collection)))
    fmt.Println(collection)
}
```

Output

```
[0 1 2 3 4 5 9]
[9 5 4 3 2 1 0]
[a b c d y z]
[z y d c b a]
[{Yi M 2} {Json C 3} {Li L 8} {Zack W 15}]
[{Zack W 15} {Li L 8} {Json C 3} {Yi M 2}]
```

⬆ back to top

buffers

Examples of how to allocate a buffer, write in big or little endian format, encode to a hex string, and check if buffers are equal.

Node.js

```
const buf = Buffer.alloc(6)

let value = 0x1234567890ab
let offset = 0
let byteLength = 6

buf.writeUIntBE(value, offset, byteLength)

let hexstr = buf.toString('hex')
console.log(hexstr)

const buf2 = Buffer.alloc(6)

value = 0x1234567890ab
offset = 0
byteLength = 6

buf2.writeUIntLE(value, offset, byteLength)

hexstr = buf2.toString('hex')
console.log(hexstr)

let isEqual = Buffer.compare(buf, buf2) === 0
console.log(isEqual)

isEqual = Buffer.compare(buf, buf) === 0
console.log(isEqual)
```

Output

```
1234567890ab
ab9078563412
false
true
```

Go

```
package main

import (
    "bytes"
    "encoding/binary"
    "encoding/hex"
```

```
    "fmt"
    "log"
    "math/big"
    "reflect"
)

func writeUIntBE(buffer []byte, value, offset, byteLength int64) {
    slice := make([]byte, byteLength)
    val := new(big.Int)
    val.SetUint64(uint64(value))
    valBytes := val.Bytes()

    buf := bytes.NewBuffer(slice)
    err := binary.Write(buf, binary.BigEndian, &valBytes)
    if err != nil {
        log.Fatal(err)
    }

    slice = buf.Bytes()
    slice = slice[int64(len(slice))-byteLength : len(slice)]

    copy(buffer[offset:], slice)
}

func writeUIntLE(buffer []byte, value, offset, byteLength int64) {
    slice := make([]byte, byteLength)
    val := new(big.Int)
    val.SetUint64(uint64(value))
    valBytes := val.Bytes()

    tmp := make([]byte, len(valBytes))
    for i := range valBytes {
        tmp[i] = valBytes[len(valBytes)-1-i]
    }

    copy(slice, tmp)
    copy(buffer[offset:], slice)
}

func main() {
    buf := make([]byte, 6)
    writeUIntBE(buf, 0x1234567890ab, 0, 6)

    fmt.Println(hex.EncodeToString(buf))

    buf2 := make([]byte, 6)
    writeUIntLE(buf2, 0x1234567890ab, 0, 6)

    fmt.Println(hex.EncodeToString(buf2))

    isEqual := reflect.DeepEqual(buf, buf2)
    fmt.Println(isEqual)
```

```
    isEqual = reflect.DeepEqual(buf, buf)
    fmt.Println(isEqual)
}
```

Output

```
1234567890ab
ab9078563412
false
true
```

back to top

maps

Node.js

```
const map = new Map()
map.set('foo', 'bar')

let found = map.has('foo')
console.log(found)

let item = map.get('foo')
console.log(item)

map.delete('foo')

found = map.has('foo')
console.log(found)

item = map.get('foo')
console.log(item)

const map2 = {}
map2['foo'] = 'bar'
item = map2['foo']
delete map2['foo']

const map3 = new Map()
map3.set('foo', 100)
map3.set('bar', 200)
map3.set('baz', 300)

for (let [key, value] of map3) {
  console.log(key, value)
}
```

Output

```
true
bar
false
undefined
foo 100
bar 200
baz 300
```

Go

```go
package main

import "fmt"

func main() {
    map1 := make(map[string]string)

    map1["foo"] = "bar"

    item, found := map1["foo"]
    fmt.Println(found)
    fmt.Println(item)

    delete(map1, "foo")

    item, found = map1["foo"]
    fmt.Println(found)
    fmt.Println(item)

    map2 := make(map[string]int)
    map2["foo"] = 100
    map2["bar"] = 200
    map2["baz"] = 300

    for key, value := range map2 {
        fmt.Println(key, value)
    }
}
```

Output

```
true
bar
false

foo 100
bar 200
baz 300
```

back to top

objects

Node.js

```
const obj = {
  someProperties: {
    'foo': 'bar'
  },
  someMethod: (prop) => {
    return obj.someProperties[prop]
  }
}

let item =  obj.someProperties['foo']
console.log(item)

item = obj.someMethod('foo')
console.log(item)
```

Output

```
bar
bar
```

Go

```
package main

import "fmt"

type Obj struct {
    SomeProperties map[string]string
}

func NewObj() *Obj {
    return &Obj{
        SomeProperties: map[string]string{
            "foo": "bar",
        },
    }
}

func (o *Obj) SomeMethod(prop string) string {
    return o.SomeProperties[prop]
}

func main() {
    obj := NewObj()

    item := obj.SomeProperties["foo"]
    fmt.Println(item)

    item = obj.SomeMethod("foo")
```

```
    fmt.Println(item)
}
```

Output

```
bar
bar
```

⬆ back to top

functions

Node.js

```
function add(a, b) {
  return a + b
}

const result = add(2,3)
console.log(result)
```

Output

```
5
```

Go

```
package main

import "fmt"

func add(a int, b int) int {
    return a + b
}

func main() {
    result := add(2, 3)
    fmt.Println(result)
}
```

Output

```
5
```

⬆ back to top

default values

Node.js

```
function greet(name = 'stranger') {
  return `hello ${name}`
```

```
}

let message = greet()
console.log(message)

message = greet('bob')
console.log(message)
```

Output

```
hello stranger
hello bob
```

Go

use pointers and check for nil to know if explicitly left blank

```
package main

import "fmt"

func greet(name *string) string {
    n := "stranger"
    if name != nil {
        n = *name
    }

    return fmt.Sprintf("hello %s", n)
}

func main() {
    message := greet(nil)
    fmt.Println(message)

    name := "bob"
    message = greet(&name)
    fmt.Println(message)
}
```

Output

```
hello stranger
hello bob
```

back to top

destructuring

Node.js

```
const obj = { key: 'foo', value: 'bar' }
```

```
const { key, value } = obj
console.log(key, value)
```

Output

```
foo bar
```

Go

```
package main

import "fmt"

type Obj struct {
    Key   string
    Value string
}

func (o *Obj) Read() (string, string) {
    return o.Key, o.Value
}

func main() {
    obj := Obj{
        Key:   "foo",
        Value: "bar",
    }

    // option 1: multiple variable assignment
    key, value := obj.Key, obj.Value
    fmt.Println(key, value)

    // option 2: return multiple values from a function
    key, value = obj.Read()
    fmt.Println(key, value)
}
```

Output

```
foo bar
foo bar
```

back to top

spread operator

Node.js

```
const array = [1, 2, 3, 4, 5]

console.log(...array)
```

Output

```
1 2 3 4 5
```

Go

```go
package main

import "fmt"

func main() {
    array := []byte{1, 2, 3, 4, 5}

    var i []interface{}
    for _, value := range array {
        i = append(i, value)
    }

    fmt.Println(i...)
}
```

Output

```
1 2 3 4 5
```

back to top

rest operator

Node.js

```javascript
function sum(...nums) {
    let t = 0

    for (let n of nums) {
        t += n
    }

    return t
}

const total = sum(1, 2, 3, 4, 5)
console.log(total)
```

Output

```
15
```

Go

```go
package main
```

```go
import "fmt"

func sum(nums ...int) int {
    var t int
    for _, n := range nums {
        t += n
    }

    return t
}

func main() {
    total := sum(1, 2, 3, 4, 5)
    fmt.Println(total)
}
```

Output

```
15
```

back to top

swapping

Node.js

```javascript
let a = 'foo'
let b = 'bar'

console.log(a, b);

[b, a] = [a, b]

console.log(a, b)
```

Output

```
foo bar
bar foo
```

Go

```go
package main

import "fmt"

func main() {
    a := "foo"
    b := "bar"

    fmt.Println(a, b)
```

```
    b, a = a, b

    fmt.Println(a, b)
}
```

Output

```
foo bar
bar foo
```

⬆ back to top

classes

Examples of classes, constructors, instantiation, and "this" keyword.

Node.js

```
class Foo {
  constructor(value) {
    this.item = value
  }

  getItem() {
    return this.item
  }

  setItem(value) {
    this.item = value
  }
}

const foo = new Foo('bar')
console.log(foo.item)

foo.setItem('qux')

const item = foo.getItem()
console.log(item)
```

Output

```
bar
qux
```

Go

(closest thing to a class is to use a structure)

```
package main

import "fmt"
```

```go
type Foo struct {
    Item string
}

func NewFoo(value string) *Foo {
    return &Foo{
        Item: value,
    }
}

func (f *Foo) GetItem() string {
    return f.Item
}

func (f *Foo) SetItem(value string) {
    f.Item = value
}

func main() {
    foo := NewFoo("bar")
    fmt.Println(foo.Item)

    foo.SetItem("qux")

    item := foo.GetItem()
    fmt.Println(item)
}
```

Output

```
bar
qux
```

back to top

generators

Node.js

```javascript
function *generator() {
  yield 'hello'
  yield 'world'
}

let gen = generator()

while (true) {
  let { value, done } = gen.next()
  console.log(value, done)

  if (done) {
    break
```

```
  }
}

// alternatively
for (let value of generator()) {
  console.log(value)
}
```

Output

```
hello false
world false
undefined true
hello
world
```

Go

```
package main

import "fmt"

func Generator() chan string {
    c := make(chan string)

    go func() {
        c <- "hello"
        c <- "world"

        close(c)
    }()

    return c
}

func GeneratorFunc() func() (string, bool) {
    s := []string{"hello", "world"}
    i := -1

    return func() (string, bool) {
        i++
        if i >= len(s) {
            return "", false
        }

        return s[i], true
    }
}

func main() {
    gen := Generator()
    for {
```

```
        value, more := <-gen
        fmt.Println(value, more)

        if !more {
            break
        }
    }

    // alternatively
    for value := range Generator() {
        fmt.Println(value)
    }

    // alternatively
    genfn := GeneratorFunc()
    for {
        value, more := genfn()
        fmt.Println(value, more)

        if !more {
            break
        }
    }
}
```

Output

```
hello true
world true
 false
hello
world
hello true
world true
 false
```

back to top

datetime

Examples of parsing, formatting, and getting unix timestamp of dates.

Node.js

```
const nowUnix = Date.now()
console.log(nowUnix)

const datestr = '2019-01-17T09:24:23+00:00'
const date = new Date(datestr)
console.log(date.getTime())
console.log(date.toString())
```

```
const futureDate = new Date(date)
futureDate.setDate(date.getDate()+14)
console.log(futureDate.toString())

const formatted = `${String(date.getMonth()+1).padStart(2,
0)}/${String(date.getDate()).padStart(2, 0)}/${date.getFullYear()}`
console.log(formatted)
```

Output

```
1547718844168
1547717063000
Thu Jan 17 2019 01:24:23 GMT-0800 (Pacific Standard Time)
Thu Jan 31 2019 01:24:23 GMT-0800 (Pacific Standard Time)
01/17/2019
```

Go

```
package main

import (
    "fmt"
    "time"
)

func main() {
    nowUnix := time.Now().Unix()
    fmt.Println(nowUnix)

    datestr := "2019-01-17T09:24:23+00:00"
    date, err := time.Parse("2006-01-02T15:04:05Z07:00", datestr)
    if err != nil {
        panic(err)
    }

    fmt.Println(date.Unix())
    fmt.Println(date.String())

    futureDate := date.AddDate(0, 0, 14)
    fmt.Println(futureDate.String())

    formatted := date.Format("01/02/2006")
    fmt.Println(formatted)
}
```

Output

```
1547718844
1547717063
2019-01-17 09:24:23 +0000 +0000
2019-01-31 09:24:23 +0000 +0000
01/17/2019
```

⬆ back to top

timeout

Node.js

```
setTimeout(callback, 1e3)

function callback() {
  console.log('called')
}
```

Output

```
called
```

Go

```
package main

import (
    "fmt"
    "sync"
    "time"
)

var wg sync.WaitGroup

func callback() {
    defer wg.Done()
    fmt.Println("called")
}

func main() {
    wg.Add(1)
    time.AfterFunc(1*time.Second, callback)
    wg.Wait()
}
```

Output

```
called
```

⬆ back to top

interval

Node.js

```
let i = 0

const id = setInterval(callback, 1e3)
```

```
function callback() {
  console.log('called', i)

  if (i === 3) {
    clearInterval(id)
  }

  i++
}
```

Output

```
called 0
called 1
called 2
called 3
```

Go

```
package main

import (
    "fmt"
    "time"
)

func callback(i int) {
    fmt.Println("called", i)
}

func main() {
    ticker := time.NewTicker(1 * time.Second)

    i := 0
    for range ticker.C {
        callback(i)

        if i == 3 {
            ticker.Stop()
            break
        }

        i++
    }
}
```

Output

```
called 0
called 1
```

```
called 2
called 3
```

⬆ back to top

IIFE

Immediately invoked function expression

Node.js

```
(function(name) {
  console.log('hello', name)
})('bob')
```

Output

```
hello bob
```

Go

```
package main

import "fmt"

func main() {
    func(name string) {
        fmt.Println("hello", name)
    }("bob")
}
```

Output

```
hello bob
```

⬆ back to top

files

Examples of creating, opening, writing, reading, closing, and deleting files.

Node.js

```
const fs = require('fs')

// create file
fs.closeSync(fs.openSync('test.txt', 'w'))

// open file (returns file descriptor)
const fd = fs.openSync('test.txt', 'r+')

let wbuf = Buffer.from('hello world.')
let rbuf = Buffer.alloc(12)
```

```
let off = 0
let len = 12
let pos = 0

// write file
fs.writeSync(fd, wbuf, pos)

// read file
fs.readSync(fd, rbuf, off, len, pos)

console.log(rbuf.toString())

// close file
fs.closeSync(fd)

// delete file
fs.unlinkSync('test.txt')
```

Output

```
hello world.
```

Go

```
package main

import (
    "fmt"
    "os"
    "syscall"
)

func main() {
    // create file
    file, err := os.Create("test.txt")
    if err != nil {
        panic(err)
    }

    // close file
    file.Close()

    // open file
    file, err = os.OpenFile("test.txt", os.O_RDWR, 0755)
    if err != nil {
        panic(err)
    }

    // file descriptor
    fd := file.Fd()

    // open file (using file descriptor)
```

```
    file = os.NewFile(fd, "test file")

    wbuf := []byte("hello world.")
    rbuf := make([]byte, 12)
    var off int64

    // write file
    if _, err := file.WriteAt(wbuf, off); err != nil {
        panic(err)
    }

    // read file
    if _, err := file.ReadAt(rbuf, off); err != nil {
        panic(err)
    }

    fmt.Println(string(rbuf))

    // close file (using file descriptor)
    if err := syscall.Close(int(fd)); err != nil {
        panic(err)
    }

    // delete file
    if err := os.Remove("test.txt"); err != nil {
        panic(err)
    }
}
```

Output

```
hello world.
```

back to top

json

Examples of how to parse (unmarshal) and stringify (marshal) JSON.

Node.js

```
let jsonstr = '{"foo":"bar"}'

let parsed = JSON.parse(jsonstr)
console.log(parsed)

jsonstr = JSON.stringify(parsed)
console.log(jsonstr)
```

Output

```
{ foo: 'bar' }
{"foo":"bar"}
```

Go

```go
package main

import (
    "encoding/json"
    "fmt"
)

type T struct {
    Foo string `json:"foo"`
}

func main() {
    jsonstr := `{"foo":"bar"}`

    t := new(T)
    err := json.Unmarshal([]byte(jsonstr), t)
    if err != nil {
        panic(err)
    }

    fmt.Println(t)

    marshalled, err := json.Marshal(t)
    jsonstr = string(marshalled)
    fmt.Println(jsonstr)
}
```

Output

```
&{bar}
{"foo":"bar"}
```

back to top

big numbers

Examples of creating big number types from and to uint, string, hex, and buffers.

Node.js

```javascript
let bn = 75n;
console.log(bn.toString(10))

bn = BigInt('75')
console.log(bn.toString(10))

bn = BigInt(0x4b)
console.log(bn.toString(10))

bn = BigInt('0x4b')
```

```
console.log(bn.toString(10))

bn = BigInt('0x' + Buffer.from('4b', 'hex').toString('hex'))
console.log(bn.toString(10))
console.log(Number(bn))
console.log(bn.toString(16))
console.log(Buffer.from(bn.toString(16), 'hex'))

let bn2 = BigInt(100)
let isEqual = bn === bn2
console.log(isEqual)

let isGreater = bn > bn2
console.log(isGreater)

let isLesser = bn < bn2
console.log(isLesser)
```

Output

```
75
75
75
75
75
75
4b
<Buffer 4b>
false
false
true
```

Go

```
package main

import (
    "encoding/hex"
    "fmt"
    "math/big"
)

func main() {
    bn := new(big.Int)
    bn.SetUint64(75)
    fmt.Println(bn.String())

    bn = new(big.Int)
    bn.SetString("75", 10)
    fmt.Println(bn.String())

    bn = new(big.Int)
```

```
    bn.SetUint64(0x4b)
    fmt.Println(bn.String())

    bn = new(big.Int)
    bn.SetString("4b", 16)
    fmt.Println(bn.String())

    bn = new(big.Int)
    bn.SetBytes([]byte{0x4b})
    fmt.Println(bn.String())
    fmt.Println(bn.Uint64())
    fmt.Println(hex.EncodeToString(bn.Bytes()))
    fmt.Println(bn.Bytes())

    bn2 := big.NewInt(100)
    isEqual := bn.Cmp(bn2) == 0
    fmt.Println(isEqual)

    isGreater := bn.Cmp(bn2) == 1
    fmt.Println(isGreater)

    isLesser := bn.Cmp(bn2) == -1
    fmt.Println(isLesser)
}
```

Output

```
75
75
75
75
75
75
4b
[75]
false
false
true
```

back to top

promises

Node.js

```
function asyncMethod(value) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('resolved: ' + value)
    }, 1e3)
  })
}
```

```
function main() {
  asyncMethod('foo')
    .then(result => console.log(result))
    .catch(err => console.error(err))

  Promise.all([
    asyncMethod('A'),
    asyncMethod('B'),
    asyncMethod('C')
  ])
  .then(result => console.log(result))
  .catch(err => console.error(err))
}

main()
```

Output

```
resolved: foo
[ 'resolved: A', 'resolved: B', 'resolved: C' ]
```

Go

(closest thing is to use channels)

```
package main

import (
    "fmt"
    "sync"
    "time"

    "github.com/prometheus/common/log"
)

func asyncMethod(value string) chan interface{} {
    ch := make(chan interface{}, 1)
    go func() {
        time.Sleep(1 * time.Second)
        ch <- "resolved: " + value
        close(ch)
    }()

    return ch
}

func resolveAll(ch ...chan interface{}) chan interface{} {
    var wg sync.WaitGroup
    res := make([]string, len(ch))
    resCh := make(chan interface{}, 1)

    go func() {
```

```
        for i, c := range ch {
            wg.Add(1)
            go func(j int, ifcCh chan interface{}) {
                ifc := <-ifcCh
                switch v := ifc.(type) {
                case error:
                    resCh <- v
                case string:
                    res[j] = v
                }
                wg.Done()
            }(i, c)
        }

        wg.Wait()
        resCh <- res
        close(resCh)
    }()

    return resCh
}

func main() {
    var wg sync.WaitGroup
    wg.Add(2)

    go func() {
        result := <-asyncMethod("foo")
        switch v := result.(type) {
        case string:
            fmt.Println(v)
        case error:
            log.Errorln(v)
        }

        wg.Done()
    }()

    go func() {
        result := <-resolveAll(
            asyncMethod("A"),
            asyncMethod("B"),
            asyncMethod("C"),
        )

        switch v := result.(type) {
        case []string:
            fmt.Println(v)
        case error:
            log.Errorln(v)
        }
```

```
        wg.Done()
    }()

    wg.Wait()
}
```

Output

```
resolved: foo
[resolved: A resolved: B resolved: C]
```

back to top

async/await

Node.js

```
function hello(name) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (name === 'fail') {
        reject(new Error('failed'))
      } else {
        resolve('hello ' + name)
      }
    }, 1e3)
  })
}

async function main() {
  try {
    let output = await hello('bob')
    console.log(output)

    output = await hello('fail')
    console.log(output)
  } catch(err) {
    console.log(err.message)
  }
}

main()
```

Output

```
hello bob
failed
```

Go

(closest thing is to use channels)

```go
package main

import (
    "errors"
    "fmt"
    "time"

    "github.com/prometheus/common/log"
)

func hello(name string) chan interface{} {
    ch := make(chan interface{}, 1)
    go func() {
        time.Sleep(1 * time.Second)
        if name == "fail" {
            ch <- errors.New("failed")
        } else {
            ch <- "hello " + name
        }
    }()

    return ch
}

func main() {
    result := <-hello("bob")
    switch v := result.(type) {
    case string:
        fmt.Println(v)
    case error:
        log.Errorln(v)
    }

    result = <-hello("fail")
    switch v := result.(type) {
    case string:
        fmt.Println(v)
    case error:
        log.Errorln(v)
    }
}
```

Output

```
hello bob
failed
```

back to top

streams

Examples of reading and writing streams

Node.js

```
const { Readable, Writable } = require('stream')

const inStream = new Readable()

inStream.push(Buffer.from('foo'))
inStream.push(Buffer.from('bar'))
inStream.push(null) // end stream
inStream.pipe(process.stdout)

const outStream = new Writable({
  write(chunk, encoding, callback) {
    console.log('received: ' + chunk.toString('utf8'))
    callback()
  }
})

outStream.write(Buffer.from('abc'))
outStream.write(Buffer.from('xyz'))
outStream.end()
```

Output

```
foobar
received: abc
received: xyz
```

Go

```
package main

import (
    "bufio"
    "bytes"
    "fmt"
    "io"
    "os"
    "runtime"
)

func main() {
    inStream := new(bytes.Buffer)
    w := bufio.NewWriter(inStream)
    _, err := w.Write([]byte("foo"))
    if err != nil {
        panic(err)
    }
    _, err = w.Write([]byte("bar"))
    if err != nil {
        panic(err)
    }
```

```
    err = w.Flush()
    if err != nil {
        panic(err)
    }

    inStream.WriteTo(os.Stdout)
    fmt.Print("\n")

    outStream := new(bytes.Buffer)
    outStream.Write([]byte("abc\n"))
    outStream.Write([]byte("xyc\n"))
    piper, pipew := io.Pipe()

    go func() {
        sc := bufio.NewScanner(piper)
        for sc.Scan() {
            fmt.Println("received: " + sc.Text())
        }
        if err := sc.Err(); err != nil {
            panic(err)
        }

        os.Exit(0)
    }()

    go func() {
        defer pipew.Close()
        io.Copy(pipew, outStream)
    }()

    defer runtime.Goexit()
}
```

Output

```
foobar
received: abc
received: xyc
```

back to top

event emitter

Node.js

```
const EventEmitter = require('events')
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter()

myEmitter.on('my-event', msg => {
  console.log(msg)
})
```

```
myEmitter.on('my-other-event', msg => {
  console.log(msg)
})

myEmitter.emit('my-event', 'hello world')
myEmitter.emit('my-other-event', 'hello other world')
```

Output

```
hello world
hello other world
```

Go

(closest thing is to use channels)

```
package main

import (
    "fmt"
)

type MyEmitter map[string]chan string

func main() {
    myEmitter := MyEmitter{}
    myEmitter["my-event"] = make(chan string)
    myEmitter["my-other-event"] = make(chan string)

    go func() {
        for {
            select {
            case msg := <-myEmitter["my-event"]:
                fmt.Println(msg)
            case msg := <-myEmitter["my-other-event"]:
                fmt.Println(msg)
            }
        }
    }()

    myEmitter["my-event"] <- "hello world"
    myEmitter["my-other-event"] <- "hello other world"
}
```

Output

```
hello world
hello other world
```

back to top

errors

Node.js

```
const err1 = new Error('some error')

console.log(err1)

class FooError extends Error{
  constructor(message) {
    super(message)
    this.name = 'FooError'
    this.message = message
  }

  toString() {
    return this.message
  }
}

const err2 = new FooError('my custom error')

console.log(err2)
```

Output

```
Error: some error
{ FooError: my custom error }
```

Go

```
package main

import (
    "errors"
    "fmt"
)

type FooError struct {
    s string
}

func (f *FooError) Error() string {
    return f.s
}

func NewFooError(s string) error {
    return &FooError{s}
}

func main() {
    err1 := errors.New("some error")
    fmt.Println(err1)
```

```
    err2 := NewFooError("my custom error")
    fmt.Println(err2)
}
```

Output

```
some error
my custom error
```

⬆ back to top

try/catch

Node.js

```
function foo(fail) {
  if (fail) {
    throw Error('my error')
  }
}

function main() {
  try {
    foo(true)
  } catch(err) {
    console.log(`caught error: ${err.message}`)
  }
}

main()
```

Output

```
caught error: my error
```

Go

```
package main

import (
    "errors"
    "fmt"
)

func foo(fail bool) error {
    if fail {
        return errors.New("my error")
    }

    return nil
}
```

```go
func main() {
    err := foo(true)
    if err != nil {
        fmt.Printf("caught error: %s\n", err.Error())
    }
}
```

Output

```
caught error: my error
```

back to top

exceptions

Node.js

```javascript
function foo() {
  throw Error('my exception')
}

function main() {
  foo()
}

process.on('uncaughtException', err => {
  console.log(`caught exception: ${err.message}`)
  process.exit(1)
})

main()
```

Output

```
caught exception: my exception
```

Go

```go
package main

import (
    "fmt"
)

func foo() {
    panic("my exception")
}

func main() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Printf("caught exception: %s", r)
```

```
        }
    }()

    foo()
}
```

Output

```
caught exception: my exception
```

[back to top](#)

regex

Node.js

```
let input = 'foobar'
let replaced = input.replace(/foo(.*)/i, 'qux$1')
console.log(replaced)

let match = /o{2}/i.test(input)
console.log(match)

input = '111-222-333'
let matches = input.match(/([0-9]+)/gi)
console.log(matches)
```

Output

```
quxbar
true
[ '111', '222', '333' ]
```

Go

```
package main

import (
    "fmt"
    "regexp"
)

func main() {
    input := "foobar"
    re := regexp.MustCompile(`(?i)foo(.*)`)
    replaced := re.ReplaceAllString(input, "qux$1")
    fmt.Println(replaced)

    re = regexp.MustCompile(`(?i)o{2}`)
    match := re.Match([]byte(input))
    fmt.Println(match)
```

```
    input = "111-222-333"
    re = regexp.MustCompile(`(?i)([0-9]+)`)
    matches := re.FindAllString(input, -1)
    fmt.Println(matches)
}
```

Output

```
quxbar
true
[111 222 333]
```

⬆ back to top

exec (sync)

Node.js

```
const { execSync } = require('child_process')

const output = execSync(`echo 'hello world'`)

console.log(output.toString())
```

Output

```
hello world
```

Go

```
package main

import (
    "fmt"
    "os/exec"
)

func main() {
    output, err := exec.Command("echo", "hello world").Output()
    if err != nil {
        panic(err)
    }

    fmt.Println(string(output))
}
```

Output

```
hello world
```

⬆ back to top

exec (async)

Node.js

```
const { exec } = require('child_process')

exec(`echo 'hello world'`, (error, stdout, stderr) => {
  if (error) {
    console.error(err)
  }

  if (stderr) {
    console.error(stderr)
  }

  if (stdout) {
    console.log(stdout)
  }
})
```

Output

```
hello world
```

Go

```
package main

import (
    "os"
    "os/exec"
)

func main() {
    cmd := exec.Command("echo", "hello world")
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr
    cmd.Run()
}
```

Output

```
hello world
```

back to top

tcp server

Node.js

```
const net = require('net')

function handler(socket) {
    socket.write('Received: ')
    socket.pipe(socket)
}

const server = net.createServer(handler)
server.listen(3000)
```

Output

```
$ echo 'hello' | nc localhost 3000
Received: hello
```

Go

```
package main

import (
    "bufio"
    "net"
)

func handler(conn net.Conn) {
    defer conn.Close()
    reader := bufio.NewReader(conn)

    for {
        message, err := reader.ReadString('\n')
        if err != nil {
            return
        }

        conn.Write([]byte("Received: "))
        conn.Write([]byte(message))
    }
}

func main() {
    listener, err := net.Listen("tcp", ":3000")
    if err != nil {
        panic(err)
    }

    defer listener.Close()

    for {
        conn, err := listener.Accept()
        if err != nil {
            panic(err)
        }
```

```
        go handler(conn)
    }
}
```

Output

```
$ echo 'hello' | nc localhost 3000
Received: hello
```

⬆ back to top

udp server

Node.js

```
const dgram = require('dgram')
const server = dgram.createSocket('udp4')

server.on('error', err => {
  console.error(err)
  server.close()
})

server.on('message', (msg, rinfo) => {
  const data = msg.toString('utf8').trim()
  console.log(`received: ${data} from ${rinfo.address}:${rinfo.port}`)
})

server.on('listening', () => {
  const address = server.address()
  console.log(`server listening ${address.address}:${address.port}`)
})

server.bind(3000)
```

Output

```
$ echo 'hello world' > /dev/udp/0.0.0.0/3000

server listening 0.0.0.0:3000
received: hello world from 127.0.0.1:51452
```

Go

```
package main

import (
    "fmt"
    "net"
    "strings"
)
```

```go
func main() {
    conn, err := net.ListenUDP("udp", &net.UDPAddr{
        Port: 3000,
        IP:   net.ParseIP("0.0.0.0"),
    })
    if err != nil {
        panic(err)
    }

    defer conn.Close()
    fmt.Printf("server listening %s\n", conn.LocalAddr().String())

    for {
        message := make([]byte, 20)
        rlen, remote, err := conn.ReadFromUDP(message[:])
        if err != nil {
            panic(err)
        }

        data := strings.TrimSpace(string(message[:rlen]))
        fmt.Printf("received: %s from %s\n", data, remote)
    }
}
```

Output

```
$ echo 'hello world' > /dev/udp/0.0.0.0/3000

server listening [::]:3000
received: hello world from 127.0.0.1:50275
```

⬆ back to top

http server

Node.js

```js
const http = require('http')

function handler(request, response) {
  response.writeHead(200, { 'Content-type':'text/plain' })
  response.write('hello world')
  response.end()
}

const server = http.createServer(handler)
server.listen(8080)
```

Output

```
$ curl http://localhost:8080
hello world
```

Go

```go
package main

import (
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(200)
    w.Write([]byte("hello world"))
}

func main() {
    http.HandleFunc("/", handler)
    if err := http.ListenAndServe(":8080", nil); err != nil {
        panic(err)
    }
}
```

Output

```
$ curl http://localhost:8080
hello world
```

[back to top](#)

url parse

Node.js

```javascript
const url = require('url')
const qs = require('querystring')

const urlstr = 'http://bob:secret@sub.example.com:8080/somepath?foo=bar'

const parsed = url.parse(urlstr)
console.log(parsed.protocol)
console.log(parsed.auth)
console.log(parsed.port)
console.log(parsed.hostname)
console.log(parsed.pathname)
console.log(qs.parse(parsed.search.substr(1)))
```

Output

```
http:
bob:secret
8080
```

```
sub.example.com
/somepath
{ foo: 'bar' }
```

Go

```go
package main

import (
    "fmt"
    "net/url"
)

func main() {
    urlstr := "http://bob:secret@sub.example.com:8080/somepath?foo=bar"

    u, err := url.Parse(urlstr)
    if err != nil {
        panic(err)
    }

    fmt.Println(u.Scheme)
    fmt.Println(u.User)
    fmt.Println(u.Port())
    fmt.Println(u.Hostname())
    fmt.Println(u.Path)
    fmt.Println(u.Query())
}
```

Output

```
http
bob:secret
8080
sub.example.com
/somepath
map[foo:[bar]]
```

back to top

gzip

Node.js

```node
const zlib = require('zlib')

const data = Buffer.from('hello world', 'utf-8')

zlib.gzip(data, (err, compressed) => {
  if (err) {
    console.error(err)
  }
```

```
  console.log(compressed)

  zlib.unzip(compressed, (err, decompressed) => {
    if (err) {
      console.error(err)
    }

    console.log(decompressed.toString())
  })
})
```

Output

```
<Buffer 1f 8b 08 00 00 00 00 00 00 13 cb 48 cd c9 c9 57 28 cf 2f ca 49 01 00 85 11 4a
0d 0b 00 00 00>
hello world
```

Go

```
package main

import (
    "bytes"
    "compress/gzip"
    "fmt"
)

func main() {
    data := []byte("hello world")

    compressed := new(bytes.Buffer)
    w := gzip.NewWriter(compressed)
    if _, err := w.Write(data); err != nil {
        panic(err)
    }
    if err := w.Close(); err != nil {
        panic(err)
    }

    fmt.Println(compressed.Bytes())

    decompressed := new(bytes.Buffer)
    r, err := gzip.NewReader(compressed)
    if err != nil {
        panic(err)
    }

    _, err = decompressed.ReadFrom(r)
    if err != nil {
        panic(err)
    }
```

```
    fmt.Println(string(decompressed.Bytes()))
}
```

Output

```
[31 139 8 0 0 0 0 0 0 255 202 72 205 201 201 87 40 207 47 202 73 1 4 0 0 255 255 133
17 74 13 11 0 0 0]
hello world
```

back to top

dns

DNS lookup examples

Node.js

```
const dns = require('dns')

dns.resolveNs('google.com', (err, ns) => {
  if (err) {
    console.error(err)
  }

  console.log(ns)
})

dns.resolve4('google.com', (err, ips) => {
  if (err) {
    console.error(err)
  }

  console.log(ips)
})

dns.resolveMx('google.com', (err, mx) => {
  if (err) {
    console.error(err)
  }

  console.log(mx)
})

dns.resolveTxt('google.com', (err, txt) => {
  if (err) {
    console.error(err)
  }

  console.log(txt)
})
```

```
dns.setServers(['1.1.1.1'])
console.log(dns.getServers())

dns.resolveNs('google.com', (err, ns) => {
  if (err) {
    console.error(err)
  }

  console.log(ns)
})
```

Output

```
[
  'ns2.google.com',
  'ns3.google.com',
  'ns4.google.com',
  'ns1.google.com'
]
[ '172.217.11.78' ]
[ { exchange: 'alt4.aspmx.l.google.com', priority: 50 },
  { exchange: 'alt2.aspmx.l.google.com', priority: 30 },
  { exchange: 'alt3.aspmx.l.google.com', priority: 40 },
  { exchange: 'aspmx.l.google.com', priority: 10 },
  { exchange: 'alt1.aspmx.l.google.com', priority: 20 } ]
[ [ 'v=spf1 include:_spf.google.com ~all' ],
  [ 'docusign=05958488-4752-4ef2-95eb-aa7ba8a3bd0e' ],
  [ 'facebook-domain-verification=22rm551cu4k0ab0bxsw536tlds4h95' ],
  [ 'globalsign-smime-dv=CDYX+XFHUw2wml6/Gb8+59BsH31KzUr6c1l2BPvqKX8=' ] ]
[ '1.1.1.1' ]
[
  'ns1.google.com',
  'ns2.google.com',
  'ns4.google.com',
  'ns3.google.com'
]
```

Go

```
package main

import (
    "fmt"
    "net"
)

func main() {
    ns, err := net.LookupNS("google.com")
    if err != nil {
        panic(err)
    }
```

```
    fmt.Printf("%s\n", ns)

    ips, err := net.LookupIP("google.com")
    if err != nil {
        panic(err)
    }

    fmt.Println(ips)

    mx, err := net.LookupMX("google.com")
    if err != nil {
        panic(err)
    }

    fmt.Println(mx)

    txt, err := net.LookupTXT("google.com")
    if err != nil {
        panic(err)
    }

    fmt.Println(txt)

    r := &net.Resolver{
        PreferGo: true,
        Dial: func(ctx context.Context, network, address string) (net.Conn, error) {
            d := net.Dialer{
                Timeout: time.Millisecond * time.Duration(10_000),
            }
            return d.DialContext(ctx, "udp", "1.1.1.1:53")
        },
    }

    ns, _ = r.LookupNS(context.Background(), "google.com")
    fmt.Printf("%s", ns)
}
```

Output

```
[%!s(*net.NS=&{ns3.google.com.}) %!s(*net.NS=&{ns4.google.com.}) %!s(*net.NS=&
{ns1.google.com.}) %!s(*net.NS=&{ns2.google.com.})]
[172.217.5.78 2607:f8b0:4007:80d::200e]
[0xc0000ba2e0 0xc0000ba260 0xc0000ba2a0 0xc0000ba280 0xc0000ba300]
[facebook-domain-verification=22rm551cu4k0ab0bxsw536tlds4h95 docusign=05958488-4752-
4ef2-95eb-aa7ba8a3bd0e v=spf1 include:_spf.google.com ~all globalsign-smime-
dv=CDYX+XFHUw2wml6/Gb8+59BsH31KzUr6c1l2BPvqKX8=]
[%!s(*net.NS=&{ns2.google.com.}) %!s(*net.NS=&{ns1.google.com.}) %!s(*net.NS=&
{ns3.google.com.}) %!s(*net.NS=&{ns4.google.com.})]
```

back to top

crypto

Node.js

```
const crypto = require('crypto')

const hash = crypto.createHash('sha256').update(Buffer.from('hello')).digest()

console.log(hash.toString('hex'))
```

Output

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

Go

```
package main

import (
    "crypto/sha256"
    "encoding/hex"
    "fmt"
)

func main() {
    hash := sha256.Sum256([]byte("hello"))

    fmt.Println(hex.EncodeToString(hash[:]))
}
```

Output

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

back to top

env vars

Node.js

```
const key = process.env['API_KEY']

console.log(key)
```

Output

```
$ API_KEY=foobar node examples/env_vars.js
foobar
```

Go

```
package main

import (
```

```
    "fmt"
    "os"
)

func main() {
    key := os.Getenv("API_KEY")

    fmt.Println(key)
}
```

Output

```
$ API_KEY=foobar go run examples/env_vars.go
foobar
```

back to top

cli args

Node.js

```
const args = process.argv.slice(2)

console.log(args)
```

Output

```
$ node examples/cli_args.js foo bar qux
[ 'foo', 'bar', 'qux' ]
```

Go

```
package main

import (
    "fmt"
    "os"
)

func main() {
    args := os.Args[1:]
    fmt.Println(args)
}
```

Output

```
$ go run examples/cli_args.go foo bar qux
[foo bar qux]
```

back to top

cli flags

Node.js

```
const yargs = require('yargs')

const { foo='default value', qux=false } = yargs.argv
console.log('foo:', foo)
console.log('qux:', qux)
```

Output

```
$ node examples/cli_flags.js --foo='bar' --qux=true
foo: bar
qux: true
```

Go

```
package main

import (
    "flag"
    "fmt"
)

func main() {
    var foo string
    flag.StringVar(&foo, "foo", "default value", "a string var")

    var qux bool
    flag.BoolVar(&qux, "qux", false, "a bool var")

    flag.Parse()

    fmt.Println("foo:", foo)
    fmt.Println("qux:", qux)
}
```

Output

```
$ go run examples/cli_flags.go -foo='bar' -qux=true
foo: bar
qux: true
```

back to top

stdout

Node.js

```
process.stdout.write('hello world\n')
```

Output

```
hello world
```

Go

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    fmt.Fprint(os.Stdout, "hello world\n")
}
```

Output

```
hello world
```

▲ back to top

stderr

Node.js

```node
process.stderr.write('hello error\n')
```

Output

```
hello error
```

Go

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    fmt.Fprint(os.Stderr, "hello error\n")
}
```

Output

```
hello error
```

▲ back to top

stdin

Node.js

```
const stdin = process.openStdin()

process.stdout.write('Enter name: ')

stdin.addListener('data', text => {
  const name = text.toString().trim()
  console.log('Your name is: ' + name)

  stdin.pause()
})
```

Output

```
Enter name: bob
Your name is: bob
```

Go

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Enter name: ")

    text, err := reader.ReadString('\n')
    if err != nil {
        panic(err)
    }

    name := strings.TrimSpace(text)
    fmt.Printf("Your name is: %s\n", name)
}
```

Output

```
Enter name: bob
Your name is: bob
```

back to top

modules

Node.js

```
# initializing metadata and dependencies file (package.json)
$ npm init

# installing a module
$ npm install moment --save

# updating a module
$ npm install moment@latest --save

# removing a module
$ npm uninstall moment --save

# pruning modules (removing unused modules)
$ npm prune

# publishing a module
$ npm publish
```

```
// importing a module
const moment = require('moment')

const now = moment().unix()
console.log(now)
```

Output

```
1546595748
```

```
// exporting a module
module.exports = {
  greet(name) {
    console.log(`hello ${name}`)
  }
}
```

```
// importing exported module
const greeter = require('./greeter')

greeter.greet('bob')
```

Output

```
hello bob
```

Go

Setup

```
# enable Go modules support
GO111MODULE=on

# initializing dependencies file (go.mod)
```

```
$ go mod init

# installing a module
$ go get github.com/go-shadow/moment

# updating a module
$ go get -u github.com/go-shadow/moment

# removing a module
$ rm -rf $GOPATH/pkg/mod/github.com/go-shadow/moment@v<tag>-<checksum>/

# pruning modules (removing unused modules from dependencies file)
$ go mod tidy

# download modules being used to local vendor directory (equivalent of downloading
node_modules locally)
$ go mod vendor

# publishing a module:
# Note: Go doesn't have an index of repositories like NPM.
# Go modules are hosted as public git repositories.
# To publish, simply push to the repository and tag releases.
```

```
package main

import (
    "fmt"

    // importing a module
    "github.com/go-shadow/moment"
)

func main() {
    now := moment.New().Now().Unix()
    fmt.Println(now)
}
```

Output

```
1546595748
```

```
package greeter

import (
    "fmt"
)

// exporting a module (use a capitalized name to export function)
func Greet(name string) {
    fmt.Printf("hello %s", name)
}
```

```go
package main

import (
    // importing exported module
    greeter "github.com/miguelmota/golang-for-nodejs-developers/examples/greeter_go"
)

func main() {
    greeter.Greet("bob")
}
```

Output

```
hello bob
```

back to top

stack trace

Node.js

```javascript
function foo() {
  throw new Error('failed')
}

try {
  foo()
} catch(err) {
  console.trace(err)
}
```

Output

```
Trace: Error: failed
    at foo (/Users/bob/examples/stack_trace.js:2:9)
    at Object.<anonymous> (/Users/bob/examples/stack_trace.js:6:3)
    at Module._compile (internal/modules/cjs/loader.js:688:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:699:10)
    at Module.load (internal/modules/cjs/loader.js:598:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:537:12)
    at Function.Module._load (internal/modules/cjs/loader.js:529:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:741:12)
    at startup (internal/bootstrap/node.js:285:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:739:3)
    at Object.<anonymous> (/Users/bob/examples/stack_trace.js:8:11)
    at Module._compile (internal/modules/cjs/loader.js:688:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:699:10)
    at Module.load (internal/modules/cjs/loader.js:598:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:537:12)
    at Function.Module._load (internal/modules/cjs/loader.js:529:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:741:12)
```

```
    at startup (internal/bootstrap/node.js:285:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:739:3)
```

Go

```go
package main

import (
    "errors"
    "fmt"
    "runtime/debug"
)

func foo() {
    panic(errors.New("failed"))
}

func main() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println(string(debug.Stack()))
        }
    }()

    foo()
}
```

Output

```
goroutine 1 [running]:
runtime/debug.Stack(0xc000090eb8, 0x10a8400, 0xc00007e1c0)
        /Users/mota/.gvm/gos/go1.11/src/runtime/debug/stack.go:24 +0xa7
main.main.func1()
        /Users/bob/examples/stack_trace.go:16 +0x46
panic(0x10a8400, 0xc00007e1c0)
        /Users/mota/.gvm/gos/go1.11/src/runtime/panic.go:513 +0x1b9
main.foo(...)
        /Users/bob/examples/stack_trace.go:10
main.main()
        /Users/bob/examples/stack_trace.go:20 +0xa2
```

back to top

databases

Example of creating a table, inserting rows, and reading rows from a sqlite3 database

Node.js

```node
const sqlite3 = require('sqlite3').verbose()
const db = new sqlite3.Database('./sqlite3.db')
```

```
db.serialize(() => {
  db.run('CREATE TABLE persons (name TEXT)')

  const stmt = db.prepare('INSERT INTO persons VALUES (?)')
  const names = ['alice', 'bob', 'charlie']
  for (let i = 0; i < names.length; i++) {
    stmt.run(names[i])
  }

  stmt.finalize()

  db.each('SELECT rowid AS id, name FROM persons', (err, row) => {
    if (err) {
      console.error(err)
      return
    }

    console.log(row.id, row.name)
  })
})

db.close()
```

Output

```
1 'alice'
2 'bob'
3 'charlie'
```

Go

```
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/mattn/go-sqlite3"
)

func main() {
    db, err := sql.Open("sqlite3", "./sqlite3.db")
    if err != nil {
        panic(err)
    }
    defer db.Close()

    _, err = db.Exec("CREATE TABLE persons (name TEXT)")
    if err != nil {
        panic(err)
    }
```

```
    tx, err := db.Begin()
    if err != nil {
        panic(err)
    }

    stmt, err := tx.Prepare("INSERT INTO persons VALUES (?)")
    if err != nil {
        panic(err)
    }
    defer stmt.Close()

    names := []string{"alice", "bob", "charlie"}

    for _, name := range names {
        _, err := stmt.Exec(name)
        if err != nil {
            panic(err)
        }
    }
    tx.Commit()

    rows, err := db.Query("SELECT rowid AS id, name FROM persons")
    if err != nil {
        panic(err)
    }
    defer rows.Close()

    for rows.Next() {
        var id int
        var name string
        err = rows.Scan(&id, &name)
        if err != nil {
            panic(err)
        }
        fmt.Println(id, name)
    }

    err = rows.Err()
    if err != nil {
        panic(err)
    }
}
```

Output

```
1 alice
2 bob
3 charlie
```

back to top

testing

Node.js

```
const test = require('tape')

test(t => {
  const tt = [
        {a:1, b:1, ret:2},
        {a:2, b:3, ret:5},
        {a:5, b:5, ret:10}
  ]

  t.plan(tt.length)

  tt.forEach(tt => {
    t.equal(sum(tt.a, tt.b), tt.ret)
  })
})

function sum(a, b) {
    return a + b
}
```

Output

```
$ node examples/example_test.js
TAP version 13
# (anonymous)
ok 1 should be equal
ok 2 should be equal
ok 3 should be equal

1..3
# tests 3
# pass  3

# ok
```

Go

```
package example

import (
    "fmt"
    "testing"
)

func TestSum(t *testing.T) {
    for _, tt := range []struct {
        a   int
        b   int
        ret int
    }{
```

```
        {1, 1, 2},
        {2, 3, 5},
        {5, 5, 10},
    } {
        t.Run(fmt.Sprintf("(%v + %v)", tt.a, tt.b), func(t *testing.T) {
            ret := sum(tt.a, tt.b)
            if ret != tt.ret {
                t.Errorf("want %v, got %v", tt.ret, ret)
            }
        })
    }
}

func sum(a, b int) int {
    return a + b
}
```

Output

```
$ go test -v examples/example_test.go
=== RUN   TestSum
=== RUN   TestSum/(1_+_1)
=== RUN   TestSum/(2_+_3)
=== RUN   TestSum/(5_+_5)
--- PASS: TestSum (0.00s)
    --- PASS: TestSum/(1_+_1) (0.00s)
    --- PASS: TestSum/(2_+_3) (0.00s)
    --- PASS: TestSum/(5_+_5) (0.00s)
PASS
ok      command-line-arguments  0.008s
```

back to top

benchmarking

Node.js

```
const Benchmark = require('benchmark')

const suite = new Benchmark.Suite
suite.add('fib#recursion', () => {
  fibRec(10)
})
.add('fib#loop', () => {
  fibLoop(10)
})
.on('complete', () => {
  console.log(suite[0].toString())
  console.log(suite[1].toString())
})
.run({
  async: true
```

```
})

function fibRec(n) {
  if (n <= 1) {
    return n
  }

  return fibRec(n-1) + fibRec(n-2)
}

function fibLoop(n) {
  let f = [0, 1]
  for (let i = 2; i <= n; i++) {
    f[i] = f[i-1] + f[i-2]
  }
  return f[n]
}
```

Output

```
$ node examples/benchmark_test.js
fib#recursion x 1,343,074 ops/sec ±1.26% (84 runs sampled)
fib#loop x 20,104,517 ops/sec ±3.78% (78 runs sampled)
```

Go

```
package example

import (
    "testing"
)

func BenchmarkFibRec(b *testing.B) {
    for n := 0; n < b.N; n++ {
        fibRec(10)
    }
}

func BenchmarkFibLoop(b *testing.B) {
    for n := 0; n < b.N; n++ {
        fibLoop(10)
    }
}

func fibRec(n int) int {
    if n <= 1 {
        return n
    }

    return fibRec(n-1) + fibRec(n-2)
}
```

```go
func fibLoop(n int) int {
    f := make([]int, n+1, n+2)
    if n < 2 {
        f = f[0:2]
    }
    f[0] = 0
    f[1] = 1
    for i := 2; i <= n; i++ {
        f[i] = f[i-1] + f[i-2]
    }
    return f[n]
}
```

Output

```
$ go test -v -bench=. -benchmem examples/benchmark_test.go
goos: darwin
goarch: amd64
BenchmarkFibRec-8        5000000               340 ns/op               0 B/op
0 allocs/op
BenchmarkFibLoop-8      30000000                46.5 ns/op            96 B/op
1 allocs/op
PASS
ok      command-line-arguments  3.502s
```

back to top

documentation

Node.js

jsdoc

```javascript
/**
 * Creates a new Person.
 * @class
 * @example
 * const person = new Person('bob')
 */
class Person {
  /**
   * Create a person.
   * @param {string} [name] - The person's name.
   */
  constructor(name) {
    this.name = name
  }

  /**
   * Get the person's name.
   * @return {string} The person's name
   * @example
   * person.getName()
```

```
   */
  getName() {
    return this.name
  }

  /**
   * Set the person's name.
   * @param {string} name - The person's name.
   * @example
   * person.setName('bob')
   */
  setName(name) {
    this.name = name
  }
}
```

Go

[godoc]

`person.go`

```
package person

import "fmt"

// Person is the structure of a person
type Person struct {
    name string
}

// NewPerson creates a new person. Takes in a name argument.
func NewPerson(name string) *Person {
    return &Person{
        name: name,
    }
}

// GetName returns the person's name
func (p *Person) GetName() string {
    return p.name
}

// SetName sets the person's name
func (p *Person) SetName(name string) string {
    return p.name
}
```

`person_test.go`

```
// Example of creating a new Person.
func ExampleNewPerson() {
    person := NewPerson("bob")
```

```
    _ = person
}

// Example of getting person's name.
func ExamplePerson_GetName() {
    person := NewPerson("bob")
    fmt.Println(person.GetName())
    // Output: bob
}

// Example of setting person's name.
func ExamplePerson_SetName() {
    person := NewPerson("alice")
    person.SetName("bob")
}
```

back to top

Contributing

Pull requests are welcome!

Please submit a pull request for new interesting additions or for general content fixes.

If updating code, update both the README and the code in the examples folder.

License

Released under the MIT license.

© Miguel Mota