Golang

Variable

```
var i int // declaration
var i = 1 // declaration et initialisation
k := 3  // declaration et initialisation dans une fonction
```

Type

```
(X = [8, 16, 32, 64]
uintX       unsigned  X-bit integers (0 to 2^X-1)
intX        signed  X-bit integers (-2^(X-1) to 2^(X-1))
( X = [32, 64])
floatX     IEEE-754 X-bit floating-point numbers
( X = [64, 128])
complexX   complex numbers with float(X/2) real and imaginary parts
byte        alias for uint8
rune        alias for int32
```

Tableau / Slice

```
package main
import "fmt"
func main() {
        primes := [6]int{2, 3, 5, 7, 11, 13} // Tableau (n = 6)
        var s []int = primes[1:4]       // Slice (n = ?)
        fmt.Println(s)
}
```

Map

```
package main
import "fmt"
type Vertex struct {Lat, Long float64}
var m map[string]Vertex
func main() {
        m = make(map[string]Vertex)
        m["Bell Labs"] = Vertex{
                40.68433, -74.39967,
        }
        fmt.Println(m["Bell Labs"])
}
```

Structure

```
type Person struct {
  name   string
  email  string
  age    uint8
}
```

Goroutine et channel

```
func main() {
  go func() {fmt.Println("I am in a goroutine")}()
}
```

Avec des channels pour synchroniser

```
func main() {
  stop := make(chan int)
  go func() {
    fmt.Println("I am in a goroutine")
    stop<- 0   // envoie un valeur dans la channel
  }()
  stopper := <-stop   // sors la valeur de la channel
  fmt.Println("Bye-bye: ", stopper)
}
```

Avec channel avec tampon (plusieur valeur possible)

```
func main() {
  c := make(chan int, 10)
  for i := 0; i < 8; i++ {
    go func() {
                if i%2 {
      c<- i
      c<- i * 10
    }}()
  }
        // Ce for s'execute autant de fois que <-c peut se faire
  for {fmt.Println(<-c)}
}
```

Tcp

```
type Message struct {
        ID   string
        Data string
}
func send(conn net.Conn) {
        msg := Message{ID: "Yo", Data: "Hello"}
        bin_buf := new(bytes.Buffer)
        gobobj := gob.NewEncoder(bin_buf)
        gobobj.Encode(msg)
        conn.Write(bin_buf.Bytes())
}
func recv(conn net.Conn) {
        tmp := make([]byte, 500)
        conn.Read(tmp)
        tmpbuff := bytes.NewBuffer(tmp)
        tmpstruct := new(Message)
        gobobjdec := gob.NewDecoder(tmpbuff)
        gobobjdec.Decode(tmpstruct)
        fmt.Println(tmpstruct)
}
func main() {
        conn, _ := net.Dial("tcp", ":8081")
        send(conn)
        recv(conn)
}
```

```
unc read(conn net.Conn) {
  tmp := make([]byte, 500)
  for {
    _, err := conn.Read(tmp)
    if logerr(err) {break}
    tmpbuff := bytes.NewBuffer(tmp)
    tmpstruct := new(Message
    gobobj := gob.NewDecoder(tmpbuff)
    gobobj.Decode(tmpstruct)
    fmt.Println(tmpstruct)
    return
  }
}
func resp(conn net.Conn) {
  msg := Message{ID: "Yo", Data: "Hello back"}
  bin_buf := new(bytes.Buffer)
  gobobje := gob.NewEncoder(bin_buf)
  gobobje.Encode(msg)
  conn.Write(bin_buf.Bytes())
  conn.Close()
}

func handle(conn net.Conn) {
  timeoutDuration := 2 * time.Second
  fmt.Println("Launching server...")
  conn.SetReadDeadline(time.Now().Add(timeoutDuration))
  remoteAddr := conn.RemoteAddr().String()
  fmt.Println("Client connected from " + remoteAddr)
  read(conn)
  resp(conn)
}

func main() {
  server, _ := net.Listen("tcp", ":8081")
  for {
    conn, err := server.Accept()
      if err != nil {
        log.Println("Connection error: ", err)
        return
      }
    go handle(conn)
```

```
        }
}
```