

19. ožujak 2020., 18:15h  
online@Twitch.tv

# Programski jezik Go



# Programski jezik Go

---

# Što je Go(lang)?

- Odmah komplikacija!
- Službeni naziv jezika je „**Go**“, prilikom Internet pretrage "**golang**"
- Unatoč nomenklaturnoj nekonzistentnosti, jezik je rijetko standardiziran i precizan
- Go je **strong static typed kompajlerski** programski jezik dizajniran u Googleu
- **Strong static typed** nam garantira provjeru tipova varijabli prije runtimea, odnosno prilikom samog kompajiranja. Dodatno, znamo da jezik ne dozvoljava implicitni cast varijabli.
- To **volimo** jer nam otežava mogućnost grešaka i olakšava proces debugiranja

# Još malo o Gou? Go-u? Golangu...

- Nit vodilja je bila ponuditi moderni jezik koji će povećati produktivnost u eri lako dostupnih višejezgrenih sustava te mrežne komunikacije
- Od prve verzije 28. ožujka 2012. do trenutne verzije (1.14, 25. veljače 2020.) drži se tzv. „**Go 1 promise of compatibility**”
- Nativan na više desetaka kombinacija arhitektura i operativnih sustava
- Široko podržan od open source zajednice
- Bez relevantnih dijalektalnih implementacija
- Out of the box „paketi”: JSON; XML; csv; Email; tar, zip, gzip, bzip2; SHA, base64...; http, http2, https, tcp, udp...

# Kako početak?

- S obzirom da je Go nativno multiplatformski, možete nativno raditi na gotovo bilo kojoj arhitekturi ili operativnom sustavu
- Prije svega, preuzmite službenu izvršnu instalaciju Go-a za vašu arhitekturu – na kolegiju će se koristiti **verzija 1.13** - <https://golang.org/dl/>
- „Službeni” IDE na ovom kolegiju biti će **Visual Studio Code** - <https://code.visualstudio.com/>
- Online: službeni The Go Playground <https://play.golang.org/>,  
domaći proizvod kolege Došilovića <https://ide.judge0.com/>
- Slobodni ste raditi u čemu god se osjećate ugodno (očekujemo barem jedan seminarski rad na temu „vi vs. Emacs”)
- Nakon instalacije Visual Studio Codea, potrebno je aktivirati ekstenziju naziva ***ms-vscode.go***

# Visual Studio Code - postavke

The screenshot displays the Visual Studio Code interface with the 'Go' extension page open in the Extensions Marketplace. The left sidebar shows a list of extensions, with 'Go' by Microsoft selected. The main panel shows the details for the 'Go' extension, including its logo, version (0.13.1), and a green 'Install' button. Below the main panel, there is a 'Table of Contents' section with links to 'Language Features' (IntelliSense, Code Navigation, Code Editing, Diagnostics, Testing, Debugging, Others) and 'How to use this extension?' (Go Language Server, Settings to control the Go Language Server). At the bottom of the interface, there are two notification banners: one recommending the 'Remote - WSL' extension and another about privacy data collection.

EXTENSIONS: MARKETPLACE

@category:"programming languages"

**Go** 0.13.1  
Rich Go language support for Visual S...  
Microsoft [Install](#)

**HTML CSS Support** 0.2.3  
CSS support for HTML documents  
ecmel [Install](#)

**SQL Server (mssql)** 1.8.0  
Develop Microsoft SQL Server, Azure ...  
Microsoft [Install](#)

**Auto Close Tag** 0.5.6  
Automatically add HTML/XML close ta...  
Jun Han [Install](#)

**Auto Rename Tag** 0.1.1  
Auto rename paired HTML/XML tag  
Jun Han [Install](#)

**TSLint (deprecated)** 1.0.44  
TSLint for Visual Studio Code  
egamma [Install](#)

**PowerShell** 2020.1.0  
Develop PowerShell scripts in Visual S...  
Microsoft [Install](#)

**IntelliSense for CSS class na...** 1.19.0  
CSS class name completion for the HT...  
Zignd [Install](#)

**C++ Intellisense** 0.2.2

**Go** ms-vscode.go  
Microsoft | 2,916,020 | ★★★★★ | Repository | License  
Rich Go language support for Visual Studio Code  
[Install](#)

[Details](#) [Contributions](#) [Changelog](#)

## Go for Visual Studio Code

[gitter](#) [join chat](#) [build](#) [failing](#)

This extension adds rich language support for the [Go language](#) to VS Code.

Read the [Changelog](#) to know what has changed over the last few versions of this extension.

### Table of Contents

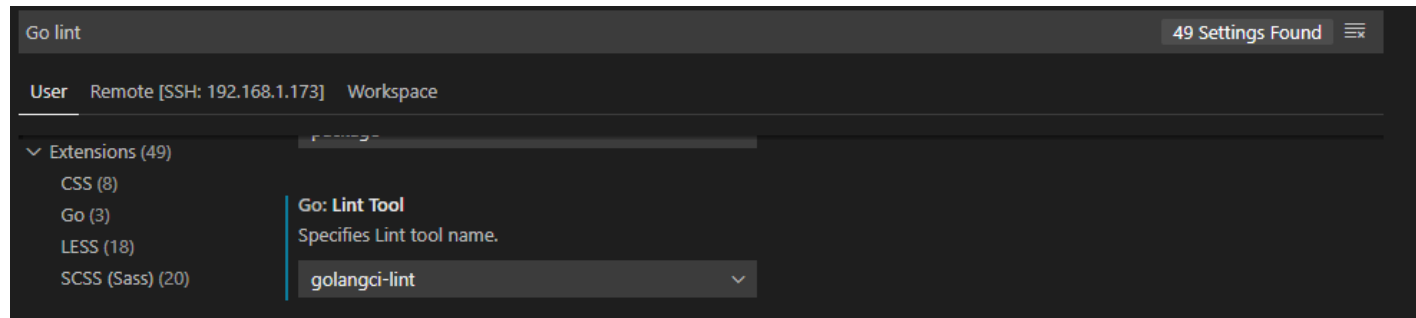
- Language Features
  - IntelliSense
  - Code Navigation
  - Code Editing
  - Diagnostics
  - Testing
  - Debugging
  - Others
- How to use this extension?
  - Go Language Server
    - Settings to control the Go Language Server

The 'Remote - WSL' extension is recommended as you have Windows Linux Subsystem (WSL) installed on your system.  
[Install](#) [Show Recommendations](#)

Help improve VS Code by allowing Microsoft to collect usage data. Read our [privacy statement](#) and learn how to [opt out](#).  
[Read More](#)

# Visual Studio Code - postavke

- Nakon što instalirate Go ekstenziju  
File -> Preferences -> Settings -> u tržilicu upišite "Go lint,, -> odaberite **golangci-lint**
- Na isti način potražite "**Go language server**,, te uključite ovu opciju
- Nakon toga VS Code treba ponovno pokrenuti



# Visual Studio Code - postavke

- Neki alati se instaliraju naknadno tek kada napravite određene akcije, npr:
  - go pls
  - golangci-lint
  - go dlv
- Instalirajte i sve naknadno preporučene ekstenzije
- Nakon toga spremni ste za prvi Go program





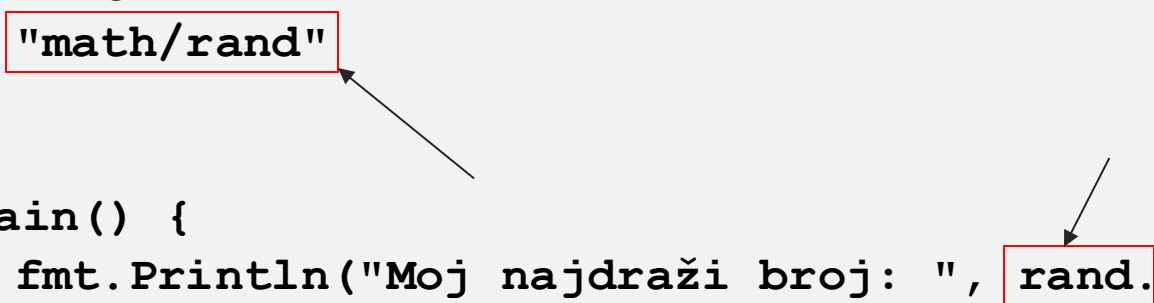
# Osnove strukture i sintakse Goa

- Svaki Go program sačinjen je od ***paketa (package)***
- Program se uvijek počinje izvršavati u paketu ***main***
- Po konvenciji, ime paketa jednako je zadnjem elementu u putanji

```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("Moj najdraži broj: ", rand.Intn(10))
}
```



# Osnove strukture i sintakse Goa

- Sintaksa import naredbe obično se koristi u obliku omeđenom zagradama, no moguće je koristiti i singularni oblik

```
package main
```

```
import "fmt"
```

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

```
func main() {
```

```
    fmt.Println("Moj najdraži broj: ", rand.Intn(10))
```

```
}
```

- U praksi, ovakva sintaksa rijetko se koristi

# Osnove strukture i sintakse Goa

- Go nema standardni pristup *nasljeđivanju*
- Identifikatori mogu biti *izvezeni (exported)*
- Neizvezeni identifikatori nisu sigurnosna mjera te ne skrivaju i ne štite podatke
- Izvezeni identifikatori počinju velikim slovom
- **Pizza** i **Pi** su izvezeni identifikatori, dok **pizza** i **pi** nisu

```
package main
```

```
import (  
    "fmt"  
    "math"  
)
```

```
func main() {  
    fmt.Println(math.pi)  
}
```

# Osnove strukture i sintakse Goa - funkcije

- Funkcije u Go mogu primiti nula (0) ili *n parametara*, odnosno vraćati nula (0) ili *n povratnih vrijednosti*
- Tip varijable u parametru dolazi *nakon* imena
- Generalna sintaksa: **func** *ime*(*parametri*) *povratneVrijednosti* {}

```
package main
```

```
import (  
    "fmt"  
    "math"  
)
```

```
func pKrug(r float64) float64 {  
    return math.Pow(r, 2) * math.Pi  
}
```

```
func main() {  
    fmt.Println(pKrug(2.71))  
}
```

# Osnove strukture i sintakse Goa - funkcije

- Ako više varijabli parametra funkcije dijele zajednički tip, to možemo skratiti:  
***func foo(a, b int, c string) {}***
- Primjer funkcije koja vraća više parametara će biti funkcija koja provjerava je li broj prost (prim). Ako je broj prost, funkcija će vratiti 0 i string „Broj je prost”, a ako nije, vratiti će najmanji djelitelj i string „Broj nije prost”
- U ovom primjeru vidjet ćemo kako **strong static type** Go ne dopušta neke stvari, a s druge strane omogućuje relaksirajuće deklaracije i čini programiranje ugodnijim zadržavajući svoja svojstva
- **Definicija:** Prirodni broj  $p > 1$  se zove **prost** ako  $p$  nema nijednog djelitelja  $d$  takvog da je  $1 < d < p$ . Ako prirodni broj  $a > 1$  nije prost, onda kažemo da je složen.

# Osnove strukture i sintakse Goa - funkcije

```
package main
import (
    "fmt"
    "math"
)

func isPrimeEx(n float64) (float64, string) {
    var i float64
    for i = 2; i <= math.Floor(math.Sqrt(n)); i++ {
        if math.Mod(n, i) == 0 {
            return i, "Broj nije prost"
        }
    }
    return 0, "Broj je prost"
}

func main() {
    n, s := isPrimeEx(21)
    fmt.Printf("%.f %s\n", n, s)
    n, s = isPrimeEx(23)
    fmt.Printf("%.f %s", n, s)
}
```

# Osnove strukture i sintakse Goa - funkcije

```
package main

import (
    "fmt"
    "math"
)

func isPrimeEx(n int) (int, string) {
    var i int
    limit := int(math.Floor(math.Sqrt(float64(n))))
    for i = 2; i <= limit; i++ {
        if n%i == 0 {
            return i, "Broj nije prost"
        }
    }
    return 0, "Broj je prost"
}
```



# Osnove strukture i sintakse Goa - funkcije

```
func main() {  
    n, s := isPrimeEx(21)  
    fmt.Printf("%d %s\n", n, s)  
    n, s = isPrimeEx(23)  
    fmt.Printf("%d %s", n, s)  
}
```

////////////////////////////////////


*3 Broj nije prost*

*0 Broj je prost*

# Osnove strukture i sintakse Goa - funkcije

- ***func isPrimeEx(n int) (int, string) {***  
Funkcija `isPrimeEx` prima jedan argument imena *n*, tipa *int*, a vraća dvije povratne vrijednosti, *int* i *string*
- ***var i int***  
Moramo deklarirati varijablu *i*, tipa *int*, kako bi smo mogli realizirati petlju u sljedećem koraku
- ***limit := int(math.Floor(math.Sqrt(float64(n))))***  
Bilo je nužno napraviti eksplicitni cast varijable *n* prilikom korištenja funkcija `math.Sqrt` koja prima *float64* argument, te dodatno taj rezultat nakon poziva funkcije najveće cijelo *math.Floor* castati u *int* jer uspoređujemo taj rezultat s varijablom *i* koju samo deklarirali kao *int*
- ***for i = 2; i <= int(math.Floor(math.Sqrt(float64(n)))) ; i++ {***  
Sintaksa *for petlje* slična je Cu. O detaljima nešto više kasnije...

# Osnove strukture i sintakse Goa - funkcije

- *n, s := isPrimeEx(21)*
- Primjetite „:” (dvotočku) ispred operatora pridruživanja
- Pozvali smo funkciju *isPrimeEx()* i njezine povratne vrijednosti pridružili varijablama *n* i *s*. Sintaksom „:”, Go je implicitno deklarirao varijable *n* i *s* s očekivanim povratnim tipom podatka kojeg vraća funkcija *isPrimeEx()*
- Time smo izbjegli eksplicitnu deklaraciju s tipom podatka za te vrijednosti, jer nam se može i time smo uštedili vrijeme i razmišljanje, obzirom da vrlo često kao povratne vrijednosti dobivamo kompleksne strukture podataka
- Ova značajka dostupna je samo u dosegu deklaracije unutar funkcije

# Osnove strukture i sintakse Goa - varijable

- Ključna riječ **var** deklarira (listu) varijabli
- Kao i kod funkcija, tip varijable dolazi na kraju
- Ključna riječ **var** može imati dva dosega, paketni i funkcijski

```
package main
```

```
import "fmt"
```

```
var c, python, java bool
```

```
func main() {  
    var golang bool  
    golang = true  
    c = false; python = false; java = false  
    fmt.Println(golang, c, python, java)  
}
```

# Osnove strukture i sintakse Go – tipovi podataka

- Osnovni tipovi podataka u Go

- `bool`

`string`

`int int8 int16 int32 int64`

`uint uint8 uint16 uint32 uint64 uintptr`

`byte // aka uint8`

`rune // aka int32; Unicode code point`

`float32 float64`

`complex64 complex128`

- *int*, *uint* i *uintptr* su, očekivano, dugački 32 odnosno 64 bita za adekvatne sustave
- Nulvrijednosti za varijable koje nisu inicijalizirane su 0 za numeričke tipove, *false* za *bool*, odnosno "" (prazan string) za *stringove*
- Go raspolaže s ključnom riječi *const*, kojom možete deklarirati konstante

# Osnove strukture i sintakse Goa – petlje

- Go raspolaže samo jednim konstruktom za petlje, i to je *for*
- Nema *do(/while)*
- Osnovna *for* petlja ima 3 elementa
  - *Inicijalno stanje*, izvršeno prije iteracije
  - *Uvjetni izraz*, evaluiran prije iteracije
  - *Post-stanje*, izvršeno nakon iteracije
- Inicijalno stanje podržava skraćenu deklaraciju („:”)
- **Važno!** Tako deklarirane varijable vidljive su samo u opsegu petlje
- Petlja će se prestati izvršavati kada uvjetni izraz bude jednak *false*
- Za razliku od nekih drugih sintaktički sličnih jezika (C, Java...), ne postoje zagrade koje okružuju tri gore navedena uvjeta, te su {} uvijek nužne

# Osnove strukture i sintakse Goa – petlje

```
package main

import "fmt"

func main() {
    sum := 0
    for i := 0; i < 10; i++ {
        sum += i
    }
    fmt.Println(sum)
    //fmt.Println(i)
}
```

# Osnove strukture i sintakse Goa – petlje

- Inicijalno i post stanje su opcionalni
- Karakteristika takve *for* petlje najbliža je do/while konstruktu („;” je redundantno u tom slučaju)

```
package main
```

```
import "fmt"
```

```
func main() {  
    sum := 1  
    for ; sum < 1000; {  
        sum += sum  
    }  
    fmt.Println(sum)  
}
```

- Ukoliko uklonite uvjetni dio iz *for* petlje, efektivno dobivate beskonačnu petlju



# Osnove strukture i sintakse Goa – uvjet (if)

- Uvjet *if* u Gou ima ista sintaktička svojstva kao i *for*; zagrade oko uvjeta su opcionalne ali {} je nužno

```
package main

import (
    "fmt"
    "math"
)

func sqrt(x float64) string {
    if x < 0 {
        return sqrt(-x) + "i"
    }
    return fmt.Sprintf(math.Sqrt(x))
}

func main() {
    fmt.Println(sqrt(2), sqrt(-4))
}
```

# Osnove strukture i sintakse Goa – uvjet (if)

- *if* može sadržavati i kratku deklaraciju varijable, dostupnu unutar *if/else* dosega

```
package main

import (
    "fmt"
    "math"
)

func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
        return v
    } else {
        fmt.Printf("%g >= %g\n", v, lim)
    }
    // v se ovdje ne vidi ☹
    return lim
}

func main() {
    fmt.Println(
        pow(3, 2, 10),
        pow(3, 3, 20),
    )
}
```

# Osnove strukture i sintakse Goa – uvjet (switch)

- *switch* omogućuje skraćeni način sekvenci *if-else* blokova
- Izvršava se prvi slučaj čija vrijednost je jednaka uvjetu
- Go *switch* funkcionira jednako kao kod Ca, Jave i PHPa, uz iznimku da nije nužno eksplicitno navoditi *break* obzirom da to Go radi za vas
- Također, *switch* ne mora biti konstanta, niti mora biti integer

```
package main

import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Print("Go runs on ")
    switch os := runtime.GOOS; os {
        case "darwin":
```

```
        fmt.Println("OS X.")
        case "linux":
            fmt.Println("Linux.")
        case "windows":
            fmt.Println("Windows.")
        default:
            fmt.Printf("%s.\n", os)
    }
}
```

# Osnove strukture i sintakse Goa – defer

- *defer* omogućuje odgađanje izvršenja funkcije unutar funkcijskog bloka do kraja njezinog izvršenja
- Poziv *defer* argumenata evaluiira se odmah, no funkcijski poziv se ne izvršava do kraja funkcijskog bloka
- Pozivi *defer* funkcija stavljaju se na stog i izvršavaju se LIFO principom
- Super korisno za oslobađanje resursa

```
package main

import "fmt"

func main() {
    fmt.Println("početak")
    for i := 0; i < 10; i++ {
        defer fmt.Println(i)
    }
    fmt.Println("kraj")
}
```

# Osnove strukture i sintakse Goa - vježbe

## Fibonaccijevi brojevi

Kratko se osvrnimo na jedan niz prirodnih brojeva koji je svoje ime dobio prema poznatom talijanskom matematičaru Leonardu Pisanu Fibonacciju (1170. – 1250.). Fibonacci se smatra najvećim europskim matematičarom srednjeg vijeka. U svojem djelu *Liber Abaci*, napisanom 1202. godine, zalaže se za uporabu arapskog brojevnog sustava, kojim se i danas koristimo, za razliku od dotada korištenog rimskog brojevnog sustava. Ipak, Fibonaccijevo ime najčešće vežemo upravo za niz prirodnih brojeva

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$

Koji se dobiva kao rješenje jednog zadatka iz te iste knjige, i to zadatka o razmnožavanju zečeva. Omjer Fibonaccijevih brojeva  $F_{n+1}/F_n$  predstavlja najbolje racionalne aproksimacije omjera zlatnog reza, tj. iracionalnog broja  $\frac{1+\sqrt{5}}{2}$

# Osnove strukture i sintakse Goa - vježbe

Navedimo sada Fibonaccijev zadatak o razmnožavanju zečeva. Pretpostavimo da je jedan par novookoćenih zečeva doveden na pusti otok 1. siječnja. Taj će par dobiti jedan par mladih zečeva svakog prvog dana u mjesecu, počevši od 1. ožujka. Svaki će novi par također dobiti kao potomke jedan par zečeva svakog prvog dana u mjesecu nakon navršena dva mjeseca života. Treba odrediti koliko će parova zečeva biti na tom otoku 1. siječnja iduće godine.

Na početku drugog mjeseca još uvijek imamo samo jedan par, ali na početku trećeg mjeseca dobivamo novi par, tako da imamo dva para. Označimo broj parova zečeva na početku  $n$ -tog mjeseca s  $F_n$ . Na početku  $(n+1)$ -og mjeseca imamo  $F_{n+1}$  parova. Na početku  $(n+2)$ -og mjeseca još uvijek imamo tih  $F_{n+1}$  parova sada već odrasli zečeva, ali dobivamo i  $F_n$  novookoćenih parova zečeva (jer imamo  $F_n$  parova starih barem dva mjeseca).

# Osnove strukture i sintakse Goa - vježbe

Prema tome,

$$F_{n+2} = F_{n+1} + F_n$$

Pokažimo sada jedan primjer. Moresov kod niz je točaka (.) i crtica (-). Njegovu duljinu definiramo tako da svaka točka pridonosi duljini s 1, a svaka crtica s 2. Dakle, ako imamo Morseov kod duljine  $n$ , onda možemo zamisliti da zapravo imamo  $n$  pozicija od kojih su neke susjedne pozicije spojene crticama, a na ostalim se mjestima nalaze točke. Odredimo broj  $M_n$  Morseovih kodova duljine  $n$ . Na primjer,  $M_4 = 5$ , jer imamo sljedećih 5 kodova duljine 4:

....    ..-    .-.    -..    --

Morseov kod duljine  $n$  može započeti ili točkom (takvih ima  $M_{n-1}$ ) ili crticom (takvih ima  $M_{n-2}$ ). Dakle,  $M_n = M_{n-1} + M_{n-2}$ , pa iz  $M_1 = 1$  i  $M_2 = 2$  slijedi da je  $M_n = F_{n+1}$

# Osnove strukture i sintakse Goa - vježbe

## Zadatak 1.

Implementirajte funkciju koja računa  $n$ -ti Fibonaccijev broj rekursivno i iterativno.

## Zadatak 2.

Implementirajte funkciju koja računa  $n$ -ti Fibonaccijev broj bilo kojom metodom. Ukoliko je uneseni broj prevelik za kapacitet tipa podatka u arhitekturi koju koristite te vratite grešku kao tip podatka string i vrijednost 0 kao za rezultat.

## Zadatak 3.

Napravite „lookup tablicu” za  $F_{33}$ ,  $F_{44}$ ,  $F_{55}$ ,  $F_{66}$  koja će odmah vratiti konstantnu vrijednost za traženi Fibonaccijev broj, a za ostale vrijednosti koristite bilo koju metodu računanja.



# Osnove strukture i sintakse Goa - vježbe

Postavlja se pitanje je li nužno za dani prirodni broj izračunati broj  $F_n$  bez računanja brojeva  $F_1, F_2, \dots, F_{n-1}$ . Odgovor na ovo pitanje je potvrđan i sadržan u tzv. Binetovoj formuli.

$$F_n = \frac{1}{\sqrt{5}} \left[ \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right]$$

## Zadatak 4.

Implementirajte Binetovu formulu za računanje  $n$ -tog Fibonaccijevog broja.

# Osnove strukture i sintakse Goa - pokazivači



- Go ima pokazivače (pointere)
- Pointeri sadrže memorijsku adresu neke vrijednosti
- Tip `*T` je pokazivač na vrijednost `T`
- Nulvrijednost pointera je `nil`
- Za razliku od C++, Go nažalost nema pointersku aritmetiku

# Osnove strukture i sintakse Goa - pokazivači

```
package main
import "fmt"
func main() {
    i, j := 42, 2701

    p := &i           // pokaži na i
    fmt.Println(*p)    // čitaj i preko pokazivača
    *p = 21            // postavi i preko pokazivača
    fmt.Println(i)

    p = &j           // pokaži na j
    *p = *p / 37      // podijeli j preko pokazivača
    fmt.Println(j)
}
```

# Osnove strukture i sintakse Goa – strukture

- **struct** je kolekcija vrijednosti
- Vrijednostima strukture pristupa se putem točke - ". "

```
package main

import "fmt"

type Tocka struct {
    X, Y int
}

var (
    v1 = Tocka {1, 2}           // tip Tocka
    v2 = Tocka {X: 1}           // Y:0 implicitno
    v3 = Tocka {}               // X:0 i Y:0
    p  = & Tocka {1, 2}        // tip *Tocka
)

func main() {
    v2.Y = 2
    fmt.Println(v1, p, v2, v3)
}
```

# Osnove strukture i sintakse Go – polje (array)

- Tip `[n]T` je polje od `n` vrijednosti tipa `T`
- Izraz ***var a [10]int*** označava varijablu ***a*** kao polje od 10 integera
- Duljina polja je dio njegovog tipa, pa se poljima ne može mijenjati vrijednost
- Iako je ovo vrlo ograničavajuće, Go omogućava drugačiji način rada s poljima

```
package main

import "fmt"

func main() {
    var a [2]string
    a[0] = "Hello"
    a[1] = "World"
```

```
    fmt.Println(a[0], a[1])
    fmt.Println(a)

    primes := [6]int{2, 3, 5, 7,
11, 13}
    fmt.Println(primes)
}
```