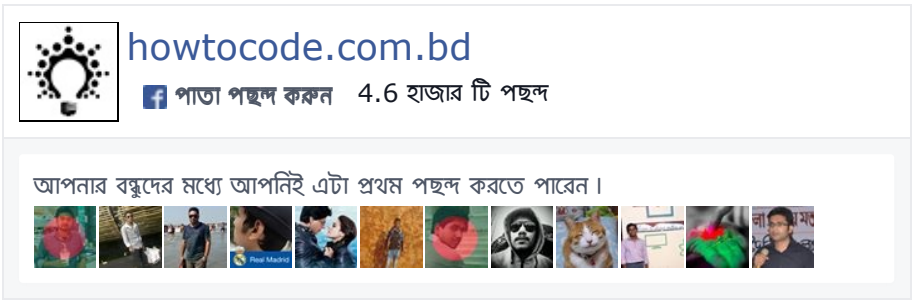
বাংলায় গোল্যাং - golang

howtocode.com.bd

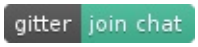
Published
with GitBook

সূচিপত্র



কোর্স এর মুল পাতা | HowToCode মুল সাইট | সবার জন্য প্রোগ্রামিং ব্লগ | পিডিএফ ডাউনলোড

বাংলায় গো-ল্যাং (golang) টিউটোরিয়াল




masnun 11


nuhil 3


howtocode-com-bd 2


gitter-badger 1

প্রারম্ভিকা

Go আসলে বেশি পরিচিত golang নামেই। এটি ২০০৭ সালে Google -এ ডেভেলপ করা হয়েছিল রবার্ট গ্রিজমার, রব পাইক এবং কেইন থম্পসন এর দ্বারা। ২০০৯ সালের দিকে এই ল্যাঙ্গুয়েজটি প্রকাশ করা হয়। ইতোমধ্যে এটি Google এর বেশ কিছু প্রোডাকশন সিস্টেমে ব্যবহৃত হচ্ছে।

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ন ওপেন সোর্স । এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে । আপনার কন্ট্রিবিউশান গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে ।

এটি মূলত একটি গিটহাব রিপোজিটোরি যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে । রিপোজটরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন ।

শুরুর আগে

শুরু করার আগে আমাদের প্রথমেই গো সেটাপ করে নেওয়া এবং এর বিল্ট ইন টুল গুলো সম্পর্কে জানা প্রয়োজন ।

ইনস্টলেশন

যারা ম্যাক ওসএক্স ব্যবহার করেন, তাদের জন্য হোমব্রু ব্যবহার করে ইনস্টল করা খুবই সহজ:

```
brew install go
```

গো প্রোগ্রাম রান করা

গো প্রোগ্রাম রান করা খুবই সহজ, আমাদের মূল গো ফাইলের নাম যদি হয় `main.go` তাহলে টার্মিনালে এই কমান্ডটি ব্যবহার করে আমরা প্রোগ্রামটি রান করাতে পারবো:

```
go run main.go
```

হ্যালো গো

একটি গো প্রোগ্রাম শুরু হয় `package` কিওয়ার্ডটি দিয়ে । প্যাকেজিং বিষয়ে আমরা পরবর্তীতে আরো বিস্তারিত জানবো । আপাতত আমাদের প্যাকেজের নাম ধরে নেই `main` । প্যাকেজ ডিফাইন করার পর আমরা `fmt` মডিউলটি ইম্পোর্ট করে নেই । এই মডিউলে স্ট্রিং ফরম্যাটিং সংশ্লিষ্ট ফাংশন থাকে । এর `Println` ফাংশনটি ব্যবহার করে আমরা আউটপুট দিতে পারি ।

কোড স্যাম্পল:

```
package main

import "fmt"

func main() {
    fmt.Println("hello go")
}
```

এবার প্রোগ্রামটি রান করি:

```
$ go run hello.go
hello go
```

ব্যাস, আমরা আমাদের প্রথম গো প্রোগ্রামটি লিখে ফেললাম এবং রান করে দেখলাম :)

ভ্যালুজ

গো তে বেশ কয়েক ধরনের ভ্যালু টাইপ আছে । এর মধ্যে স্ট্রিং, ইন্টিজার, ফ্লোট, বুলিয়ান ইত্যাদি উল্লেখযোগ্য । আগের চ্যাপ্টারে দেখা `Println` ফাংশনটি ব্যবহার করে কিছু ব্যাসিক টাইপ প্রিন্ট করে দেখি:

```
package main

import "fmt"

func main() {

    // দুইটা স্ট্রিং আমরা `+` ব্যবহার করে জোড়া লাগাতে পারি
    fmt.Println("go" + "lang")

    // ইন্টিজার ও ফ্লোটিং পয়েন্ট নাম্বার
    fmt.Println("1+1 =", 1+1)
    fmt.Println("7.0/3.0 =", 7.0/3.0)

    // বুলিয়ান টেস্ট
    fmt.Println(true && false)
    fmt.Println(true || false)
    fmt.Println(!true)
}
```

আউটপুট:

```
$ go run main.go
golang
1+1 = 2
7.0/3.0 = 2.3333333333333335
false
true
false
```

ভ্যারিয়েবলস

গো তে ভ্যারিয়েবল গুলো এক্সপ্লিসিটলি ডিফাইন করে দিতে হয় । `var` কিওয়ার্ডটি ব্যবহার করে আমরা এক বা একাধিক ভ্যারিয়েবল ডিফাইন করতে পারি । এই কিওয়ার্ডটির পর ভ্যারিয়েবল এর নাম এবং তারপর টাইপ নির্দেশ করতে হয় ।

একই টাইপের একাধিক ভ্যারিয়েবল কমা দিয়ে সেপারেট করে দিয়ে সবগুলোকে একই সাথে ডিফাইন করা সম্ভব । এছাড়া যদি আমরা টাইপ উহ্য রাখি সেক্ষেত্রে গো ভ্যালু থেকে টাইপ টা অনুমান করে নেয় । এছাড়া শর্টহ্যান্ড সিনট্যাক্স ব্যবহার করে `var` কিওয়ার্ড ও টাইপ ডিক্লেয়ারেশন ছাড়াই ভ্যারিয়েবল তৈরি করা সম্ভব ।

নিচের কোড ব্লকে আমরা ভ্যারিয়েবল সংশ্লিষ্ট কিছু উদাহরণ দেখবো:

```
package main

import "fmt"

func main() {

    // `var` এর মাধ্যমে আমরা এক বা একাধিক ভ্যারিয়েবল তৈরি করতে পারি
    var a string = "initial"
    fmt.Println(a)

    // এক সাথে অনেকগুলো ভ্যারিয়েবল ও ডিফাইন করা সম্ভব
    var b, c int = 1, 2
    fmt.Println(b, c)

    // টাইপ উল্লেখ না করলে গো নিজে থেকেই টাইপ অনুমান করে নেয়
    var d = true
    fmt.Println(d)

    // ভ্যালু না দেওয়া থাকলে "জিরো ভ্যালু" ধরে নেওয়া হয়
    var e int
    fmt.Println(e)

    // শর্টহ্যান্ড `:=` সিনট্যাক্স ব্যবহার করে টাইপ ছাড়াই
    // ভ্যারিয়েবল ডিফাইন ও ডিক্লেয়ার করা যায়
    f := "short"
    fmt.Println(f)
}
```

কন্সট্যান্টস

গো তে কনস্ট্যান্টস ডিফাইন করার জন্য `const` কিওয়ার্ডটি ব্যবহার করি আমরা । এই কিওয়ার্ডটির পর কনস্ট্যান্ট এর নাম এবং তারপর এটির ভ্যালু এ্যাসাইন করি ।

আমরা যেসব জায়গায় `var` ব্যবহার করতে পারি তার সব জায়গাতেই কনস্ট্যান্ট ডিফাইন করা সম্ভব ।

```
package main

import "fmt"

// কনস্ট্যান্ট ডিক্লেয়ারেশন
const s string = "constant"

func main() {
    fmt.Println(s)
    const n = 500000000
    fmt.Println(n)
}
```

ফর

অন্য সব ল্যাঙ্গুয়েজে আমরা `ফর` লুপ দেখেছি । মজার ব্যাপার হচ্ছে গো তে সব ধরণের লুপিংই `ফর` দিয়ে করা হয় । অর্থাৎ গো তে `হোয়াইল` , `ডু-হোয়াইল` কিংবা `ফর-ইচ` লুপ দেখা যায় না । কিন্তু এগুলোর কাজ আমরা `ফর` দিয়েই চালিয়ে নিতে পারি বেশ সহজেই!

ফর লুপ

আমরা প্রথমেই আমাদের পরিচিত ফর লুপ টা দেখে নেই । এখানে:

```
for j := 7; j <= 9; j++ {
    fmt.Println(j)
}
```

এটা আমাদের সবার পরিচিত `ফর` লুপ । এখানে `j` এর ভ্যালু শুরুতেই 7 দেওয়া হয়েছে, এর ভ্যালু ৯ হওয়া পর্যন্ত লুপ চলবে । প্রতিবার লুপ শেষে `j` এর ভ্যালু এক বাড়িয়ে দেওয়া হচ্ছে ।

হোয়াইল লুপ

অন্য নানা ল্যাঙ্গুয়েজে একটা শর্তের উপর নির্ভর করে যেভাবে হোয়াইল লুপ ব্যবহার করতাম, গো তেও আমরা কাজটা প্রায় একইভাবে করবো । শুধু পার্থক্য এখানে আমরা `ফর` কিওয়ার্ডটি ব্যবহার করে `হোয়াইল` লুপের কাজ করি ।

```
package main

import "fmt"

func main() {

    i := 1
    for i <= 3 {
        fmt.Println(i)
        i = i + 1
    }


}
```

প্রথমে `for` এবং তারপর মূল কন্ডিশন দিতে হয় । লক্ষ্য করুন এখানে `for` এর কন্ডিশনটি ডিফাইন করার সময় কোন ধরনের ব্র্যাকেট এর প্রয়োজন হয় নি । `ফর` এর পর যদি একটাই শর্ত দেওয়া হয় তবে ঐ শর্তটি যতক্ষণ সত্য হবে ততক্ষণ লুপটি চলতে থাকবে ।

যদি আমরা কোন কন্ডিশন না দেই তাহলে এটা ইনফিনিট লুপ এ চলতে থাকবে ।

```
for {
    fmt.Println("infinite loop")
    break
}
```

ফর ইচ লুপ

গো তে আমরা `ফর` লুপ ব্যবহার করেই ফর ইচ লুপের কাজ করে থাকি । এটির জন্য প্রয়োজন পড়ে `range` ফাংশনের । আমরা তাই `range` চ্যাপ্টারে দেখবো কিভাবে আমরা ফর ইচ ব্যবহার করতে পারি ।

ইফ/এলস

অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজের মতই গো তে ইফ এলস কাজ করে । একটা সিম্পল উদাহরন দেখি:

```
if 7%2 == 0 {
    fmt.Println("7 is even")
} else {
    fmt.Println("7 is odd")
}
```

আগে `for` চ্যাপ্টারে দেখেছি গো তে কন্ডিশন ব্র্যাকেটের রাখার প্রয়োজন পড়ে না । ইফ এলস এর বেলায়ও তাই । ইফ স্টেটমেন্টের পর সরাসরি আমরা কন্ডিশন লিখে দেই । ব্র্যাকেটের প্রয়োজন নেই ।

আমরা চাইলে ইফ স্টেটমেন্ট এর পর এবং মূল কন্ডিশনের আগে আমাদের প্রয়োজনমত অন্য কোন এক্সপ্রেশন ব্যবহার করতে পারি । এখানে ডিক্লেয়ার করা কোন ভ্যারিয়েবল ইফ-এলস এর সকল ব্র্যানচ এ এ্যাক্সেসিবল হবে ।

```
if num := 9; num < 0 {
    fmt.Println(num, "is negative")
} else if num < 10 {
    fmt.Println(num, "has 1 digit")
} else {
    fmt.Println(num, "has multiple digits")
}
```

এখানে আমাদের মূল কন্ডিশন হলো `num < 0` কিন্তু তার আগে আমরা `num` এর ভ্যালু `9` সেট করে নিয়েছি । লক্ষ্য করুন সব গুলো ব্লকেই কিন্তু `num` ভ্যারিয়েবলটি এ্যাক্সেস করা যাচ্ছে ।

এই উদাহরনে আমরা `else if` এর ব্যবহার ও দেখলাম । একাধিক কন্ডিশন চেক করার জন্য আমরা ইফ এর সাথে এই এলস-ইফ ব্যবহার করি ।

গো তে টার্নারী অপারেটর এর কনসেপ্ট নেই, তাই যত সংক্ষিপ্ত লজিকই হোক, পুরো ইফ-এলস ব্লক লিখতে হবে, কোন শর্টকাট নেই ।

সুইচ

সুইচ স্টেটমেন্ট এর ব্যাসিক এক্সাম্পল দেখে নেই:

```
    i := 2
    fmt.Print("write ", i, " as ")
    switch i {
    case 1:
        fmt.Println("one")
    case 2:
        fmt.Println("two")
    case 3:
        fmt.Println("three")
    }
```

এখানে আমার সুইচ স্টেটমেন্টকে একটি এক্সপ্রেশন দিয়ে দেই । এরপর সেই এক্সপ্রেশনের ভ্যালুর সাথে আমাদের কেইসগুলো ম্যাচ করিয়ে দেখি যে শর্ত মেলে কিনা । যে সব কেইস ম্যাচ করে সেইসব কেইসের সাথে থাকা কোড ব্লক রান করে ।

আমরা কেইস স্টেটমেন্ট এ কমা দিয়ে একাধিক কন্ডিশন দিয়ে দিতে পারি, যেমন:

```
package main

import "fmt"
import "time"

func main() {
    switch time.Now().Weekday() {
    case time.Saturday, time.Sunday:
        fmt.Println("it's the weekend")
    default:
        fmt.Println("it's a weekday")
    }
}
```

তবে সুইচ স্টেটমেন্ট এর পাশাপাশি কেইস স্টেটমেন্ট এও আমরা ডাইনামিক এক্সপ্রেশন ব্যবহার করতে পারি । আমরা চাইলে সুইচ স্টেটমেন্টে কোন এক্সপ্রেশন পাস নাও করতে পারি, সেক্ষেত্রে সুইচ কে ইফ-এলস এর মত করে ব্যবহার করা যায় -

```
package main

import "fmt"
import "time"

func main() {
    t := time.Now()
    switch {
    case t.Hour() < 12:
        fmt.Println("it's before noon")
    default:
        fmt.Println("it's after noon")
    }
}
```

এখানে দেখুন `case t.Hour() < 12` ডাইনামিক এক্সপ্রেশন । এবং আমরা সুইচ স্টেটমেন্টে `t` কে পাস না করে, কেইস স্টেটমেন্টে এসে সরাসরি `t.Hour()` ব্যবহার করেছি । জিনিসটা ইফ-এলস দিয়ে আমরা এভাবে করতে পারতাম:

```
package main

import "fmt"
import "time"

func main() {
    t := time.Now()
    if t.Hour() < 12 {
        fmt.Println("it's before noon")
    }

}
```

এ্যারে

গো ল্যাঙ্গুয়েজে এ্যারে হলো একই ধরনের আইটেমের সমষ্টি বা তালিকা । এবং এর আইটেম সংখ্যা নির্দিষ্ট থাকে । অর্থাৎ এ্যারে হয় ফিক্সড সাইজ ।

এ্যারে ডিফাইন করা খুবই সহজ । প্রথমে `var` কিওয়ার্ড, এরপর ভ্যারিয়েবল এর নাম এবং শেষে স্কোয়ার ব্র্যাকেট `[]` এর মধ্যে এ্যারের সাইজ এবং ব্র্যাকেট শেষে এ্যারের টাইপ ।

```
var a [5]int
```

এখানে `a` হলো একটি এ্যারে যেখানে আমরা ৫টি ইন্টিজার ভ্যালু সংরক্ষণ করতে পারি । গো ও জিরো বেইজড ইনডেক্স ফলো করে । অর্থাৎ প্রথম এলিমেন্টটির পজিশন হয় `0` । সুতরাং শেষ এলিমেন্টটি আমরা এ্যাক্সেস করতে পারি এভাবে -

```
a[4] = 100
fmt.Println("get:", a[4])
```

এখানে আমরা শেষ এলিমেন্টটির ভ্যালু হিসেবে `100` সেট করে দিচ্ছি । বিল্ট ইন `len` ফাংশনটি ব্যবহার করে আমরা এ্যারের সাইজ জানতে পারি ।

```
fmt.Println("len:", len(a))
```

আমরা চাইলে এ্যারে ডিক্লেয়ারেশনের পর কার্লি ব্রেইসের মধ্যে আইটেম গুলো ডিফাইন করে দিতে পারি । এভাবে আমরা নিচের মত করে একই সাথে এ্যারে ডিক্লেয়ার এবং তার ভ্যালুগুলো সেট করে দেই:

```
b := [5]int{1, 2, 3, 4, 5}
fmt.Println("dcl:", b)
```

মাল্টিডাইমেনশনাল এ্যারের জন্য প্রয়োজনীয় সংখ্যক স্কোয়ার ব্র্যাকেট যোগ করে সেগুলোর মধ্যে ঐ ডাইমেনশনে আইটেম সংখ্যা নির্দেশ করলেই হয় । যেমন:

```
var twoD [2][3]int
```

আমরা এর ভ্যালু গুলো ফর লুপ ব্যবহার করে এ্যাসাইন করে দিতে পারি:

```
    for i := 0; i < 2; i++ {
        for j := 0; j < 3; j++ {
            twoD[i][j] = i + j
        }
    }
    fmt.Println("2d: ", twoD)
```

তবে সাধারনত গো তে আমরা এ্যারের চেয়ে স্লাইস বেশি ব্যবহার করে থাকি । পরবর্তী চ্যাপ্টারে আমরা স্লাইস নিয়ে দেখবো ।

স্লাইসেস

গো তে স্লাইস খুবই পাওয়ারফুল একটা ডাটা স্ট্রাকচার । এ্যারে যেখানে ফিক্সড লেংথ এর হয়, স্লাইস এর এরকম কোন লিমিটেশন নেই । তবে এ্যারের মত স্লাইসের ক্ষেত্রেও আমাদের টাইপ নির্দিষ্ট করে দিতে হয় ।

আমরা `make` ফাংশনটি ব্যবহার করে স্লাইস তৈরি করতে পারি:

```
s := make([]string, 3)
fmt.Println("emp:", s)
```

এখানে আমরা একটি স্লাইস তৈরি করেছি যেটিতে আমরা স্ট্রিং স্টোর করবো এবং ইনিশিয়ালি ৩টি আইটেমের জন্য মেমোরী এ্যালোকেট করা হবে । এই ৩টি আইটেমের ইনিশিয়াল ভ্যালু হবে "জিরো ভ্যালু" - অর্থাৎ সংশ্লিস্ট টাইপ (এক্ষেত্রে স্ট্রিং) এর শুন্যর সম মানের ভ্যালু । স্ট্রিং এর ক্ষেত্রে জিরো ভ্যালু হলো ফাকা স্ট্রিং ।

আমরা এ্যারের মত করেই ইনডেক্স ব্যবহার করে স্লাইসের আইটেমগুলো এ্যাক্সেস করতে পারি:

```
fmt.Println("set:", s)
s[0] = "a"
s[1] = "b"
s[2] = "c"
fmt.Println("get:", s[2])
```

স্লাইসের লেংথ জানতে আমরা `len` ফাংশনটি ব্যবহার করি:

```
fmt.Println("len:", len(s))
```

এবং স্লাইসের শেষে যোগ করতে আমরা `append` ফাংশনটি ব্যবহার করতে পারি:

```
s = append(s, "d")
s = append(s, "e", "f")
fmt.Println("apd:", s)
```

`copy` ফাংশনটি ব্যবহার করে আমরা স্লাইস টু স্লাইস কপি করতে পারি:

```
c := make([]string, len(s))
copy(c, s)
fmt.Println("cpy:", c)
```

স্লাইসের সবচেয়ে গুরুত্বপূর্ন বৈশিষ্ট্য হলো এর স্লাইসিং অপারেটর । যেমন:

```
l := s[2:5]
fmt.Println("sl1:", l)
```

এখানে আমরা `s` স্লাইসের ইনডক্স `2` (অর্থাৎ ৩য় আইটেম) থেকে শুরু করে ইনডেক্স `5` এর আগ পর্যন্ত, অর্থাৎ ইনডেক্স `4` বা ৫ম আইটেম পর্যন্ত `l` এ এ্যাসাইন করবো । এই অপারেশনের পর `l` হবে একটি নতুন স্লাইস যেটায় থাকবে `s[2]`, `s[3]` এবং `s[4]` এর ভ্যালু ।

আমরা যদি প্রথম অংশ উহ্য রাখি সেক্ষেত্রে স্লাইসের একেবারে শুরু থেকে নতুন স্লাইস শুরু হবে:

```
l = s[:5]
fmt.Println("sl2:", l)
```

এটার ভ্যালু হবে `s[0]` থেকে `s[4]` পর্যন্ত ।

মনে রাখতে হবে স্লাইসের প্রথম অংশ ইনক্লুসিভ, অর্থাৎ প্রথম অংশে যেই ইনডেক্স থাকে সেটি নতুন স্লাইসের শুরু, আর শেষ ভাগে যেই ইনডেক্স থাকে সেটা এক্সক্লুসিভ, অর্থাৎ সেটির আগ পর্যন্ত নতুন স্লাইসের অন্তর্গত হয় ।

```
l = s[2:]
fmt.Println("sl3:", l)
```

`make` ফাংশনটি ব্যবহার না করেও আমরা একই লাইনে স্লাইস ডিক্লেয়ার এবং ইনিশিয়ালাইজ করতে পারি এই শর্টহ্যান্ড সিনট্যাক্স ব্যবহার করে:

```
t := []string{"g", "h", "i"}
fmt.Println("dcl:", t)
```

ম্যাপস

ম্যাপ হচ্ছে গো-এর এ্যাসোসিয়েটিভ ডাটা টাইপ । ম্যাপ এ আমরা একটি কি এর বিপরীতে একটি ভ্যালু স্টোর করি ।

`make` ফাংশনটি ব্যবহার করে আমরা এভাবে একটি ম্যাপ তৈরি করতে পারি:

```
m := make(map[string]int)
```

ম্যাপ এর ক্ষেত্রে `make` ফাংশনটির ফরম্যাটটি হলো - `map[ <কি-এর টাইপ> ] <ভ্যালু টাইপ>`

কি-ভ্যালু সেট করা খুবই সহজ:

```
m["k1"] = 7
m["k2"] = 13
fmt.Println("map:", m)
```

ভ্যালু রিট্রিভ করাও খুব জটিল নয়:

```
v1, pres := m["k1"]
fmt.Println("v1: ", v1)
fmt.Println("pres: ", pres)
```

ভ্যালু রিট্রিভ করার সময় প্রথম রিটার্ন ভ্যালুটি হয় ঐ কি-এর ভ্যালু । দ্বিতীয় রিটার্ন ভ্যালুটি হয় বুলিয়ান। এটি নির্দেশ করে ঐ কি-টি ম্যাপে ছিলো কিনা । এটির মাধ্যমে জিরো ভ্যালুর ক্ষেত্রে সহজে বোঝা যায় ঐ কি-এর ভ্যালু কি সত্যিই জিরো ভ্যালু নাকি ঐ নামে কোন কি-ই ছিলো না ম্যাপে ।

ম্যাপ এর লেংথ জানতে আছে আমাদের পরিচিত `len` ফাংশনটি ।

```
fmt.Println("len:", len(m))
```

কোন আইটেম ডিলিট করতে রয়েছে `delete` ফাংশনটি:

```
delete(m, "k2")
fmt.Println("map:", m)
```