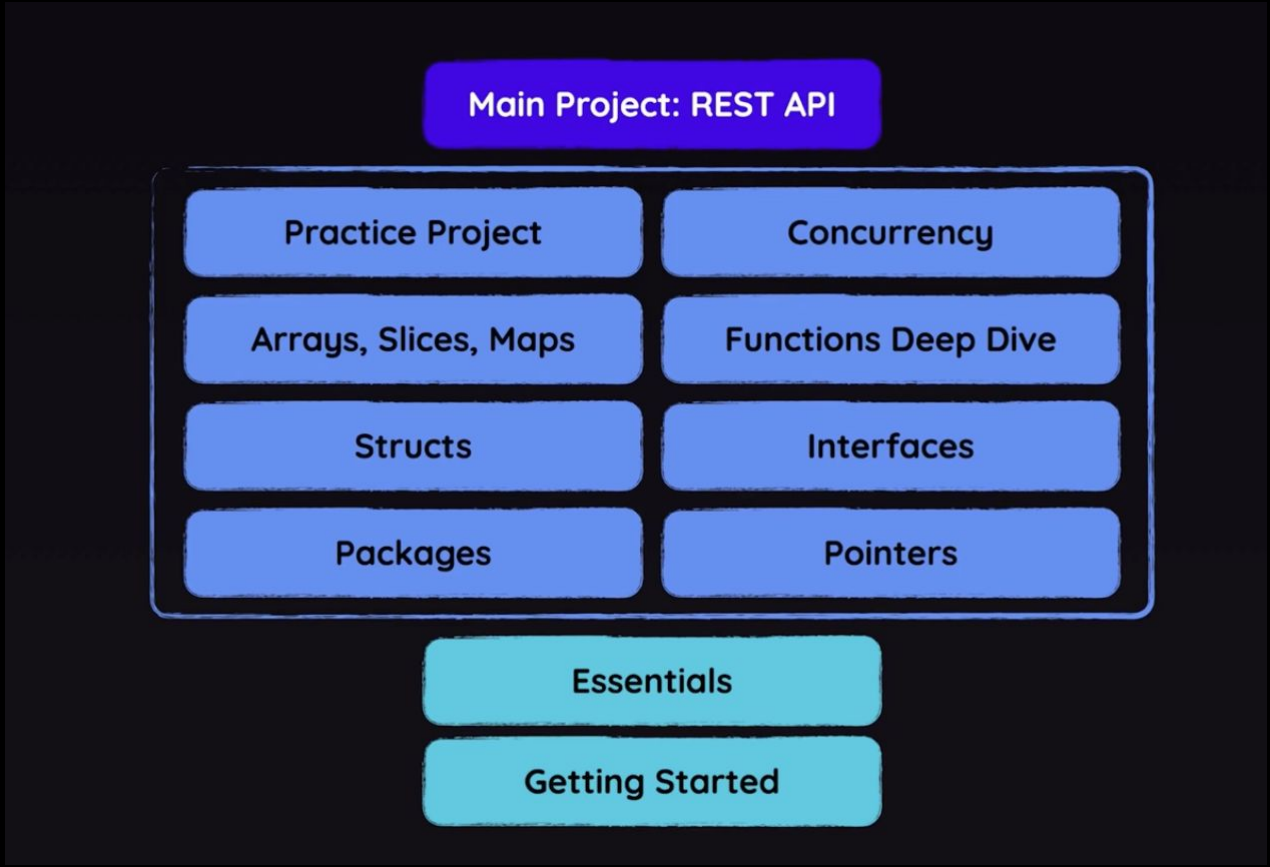
Golang

By Harshit Batra

About this course



How to get the most out of this course?

- Do not overlook Syntax
- Complete Golang LearningPath
- Use Code attachments
- Contact :
  - harshit.batra@polariscampus.com
  - +91-7701809516

https://www.codechef.com/learn/course/polaris-golang4

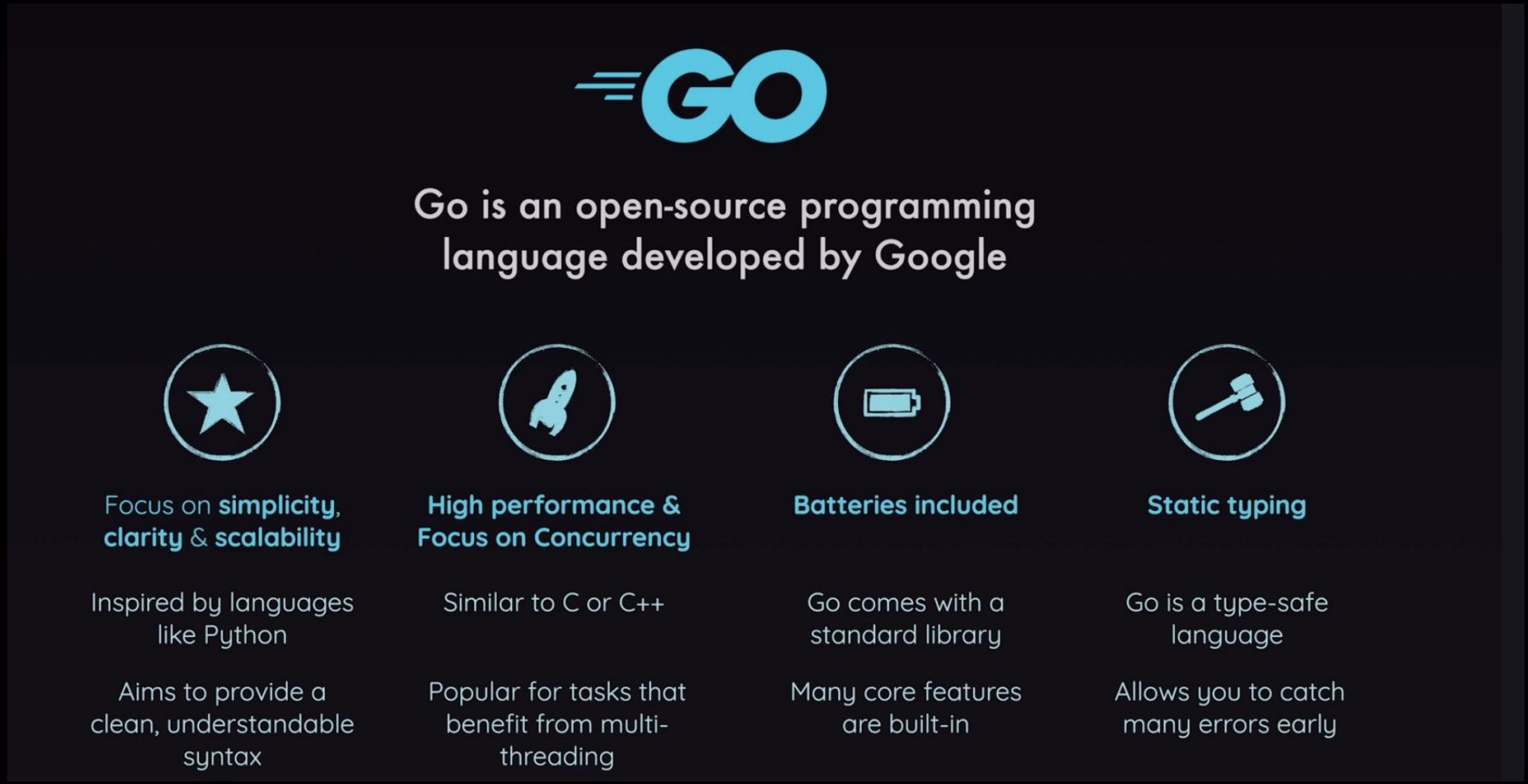
Go Introduction

- *C++ (1979), Java (1995), Go (2007)*
- Why Go?
  - *Problems with C++*
    - Dependency Bloat
    - Complexity of C++ and Java
    - Complicated Concurrency
  - *Why not Java*
    - Slow startup (because of JVM)
    - Too verbose
    - Complicated Concurrency

What is Go?

- Go is an open-source, statically typed programming language developed at Google in 2007 to improve programming productivity
- Created by
  - *Rob Pike*
  - *Ken Thompson*
  - *Robert Griesemer*
- Ideal for:
  - *scalable backend services,*
  - *cloud-native applications,*
  - *networking tools, and*
  - *DevOps infrastructure.*

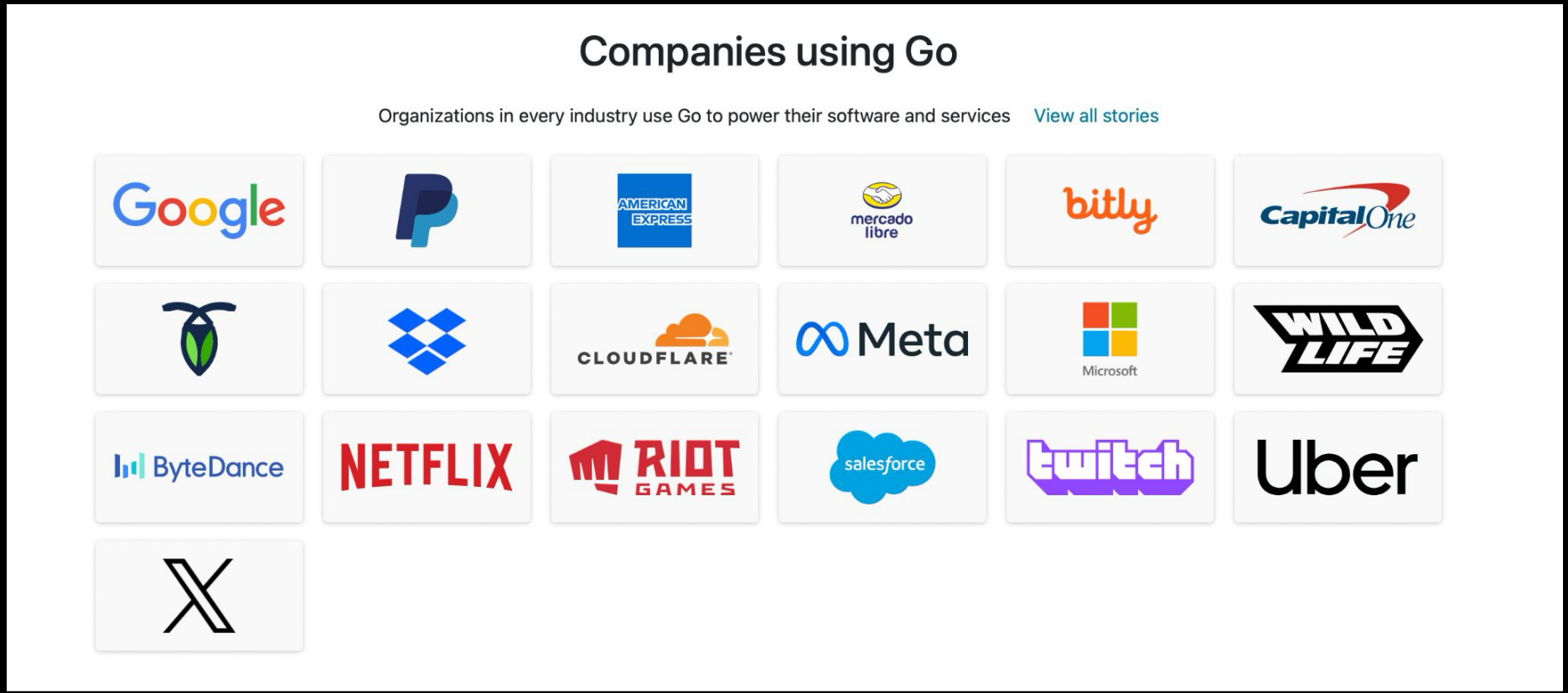
Features of Go



Features of Go

- Fast Compilation
- Simple Syntax
- Built-in concurrency
- strong standard library
- statically typed

State of Go


Companies using Go
Organizations in every industry use Go to power their software and services
View all stories
Google
AMERICAN EXPRESS
mercado libre
bitly
CapitalOne
CLOUDFLARE
Meta
Microsoft
WILD LIFE
ByteDance
NETFLIX
RIOT GAMES
salesforce
twitch
Uber

State of Go

- 2007: Go Designed, exclusive for Google
- 2009: public announcement
- 2012: Stable release Go 1.0

Cloud era : Go becomes the language of Cloud era

- 2013: Docker
- 2014: Kubernetes

https://go.dev/blog/survey2025

https://blog.jetbrains.com/go/2021/02/03/the-state-of-go/

Go vs java vs Node.js

- Java has JVM overheads
- Node.js has event-loop not suitable for CPU intensive tasks

Go setup

- GOPATH
- GOROOT

First Go Program

- Print Hello World!
- Go run
- Go build

Go Module & Package

- What is Module?
  - Module is a collection of Go packages
  - Module = Entire Project
- What is Package?
  - Package is a collection of Go files (.go) in the same folder
  - Package = Folder

Interview Questions

- What is special about Package main?
- What is special about Func main?
- Can you run program without main()?
- Can you have multiple main() functions?

Golang-

Basic Types, Control flow statements

Go Fundamentals

- Values
  - *Int, float, boolean, string*
- Variables
  - *Basic declaration, short-hand declaration, type inference, block declaration, zero value*
- Constants
  - *Basic declaration, short-hand declaration, type inference, block declaration, zero value*
- For
  - *Classic for, while loop, range for loop, infinite loop*
- If/Else
  - *Basic if-else, statement if-else*
- Switch
  - *Basic switch, Without condition*

Zero Value Concept

Every variable has default value.

Type	Zero Value
int	0
float	0.0
bool	false
string	""

Go: What is the output?

```
x := 10


if x := 20; x > 15 {
    fmt.Println(x)
}


fmt.Println(x)
```

Output:

Answer

```
x := 10

if x := 20; x > 15 {
    fmt.Println(x)
}

fmt.Println(x)
```

Output: 20 10

Go: What is the output?

```
const x int

fmt.Println(x)
```

Output:

Remember

1. := works only inside function
2. Go has only for loop
3. Switch has automatic break
4. Zero value exists
5. Constants cannot change and must be initialized at the time of declaration
6. Go is statically typed
7. Scope of variables declared inside if, for, or switch is limited to that block only
8. Short-hand declaration (:=) performs both declaration and initialization together

Type Conversion and Type Inference

Summary

int => A number WITHOUT decimal places (e.g., -5, 10, 12 etc)

float64 => A number WITH decimal places (e.g., -5.2, 10.123, 12.9 etc)

string => A text value (created via double quotes or backticks: "Hello World", `Hi everyone')

bool => true or false

uint => An unsigned integer which means a strictly non-negative integer (e.g., 0, 10, 255 etc)

Range Calculation: Simple Approach

Type: int

Type	Range	Simple Use Case Example
int8	-128 to 127	Temperature (-20°C, 50°C)
uint8 (byte)	0 to 255	File data, image pixel, ASCII character
int16	-32,768 to 32,767	Small sensor data
uint16	0 to 65,535	Port number (8080)
int32 (rune)	-2 billion to +2 billion	Unicode character
uint32	0 to 4 billion	File size (small files)
int64	-9 quintillion to +9 quintillion	Database ID, Unix timestamp
uint64	0 to 18 quintillion	Large file size
int	system dependent	Loop counter, general use
uint	system dependent	Counter (non-negative only)
uintptr	memory address	Pointer address

Float types:

Type	Range	Use Case
float32	±3.4E38	Game graphics, temperature
float64	±1.8E308	Money, GPS, backend calculations

Float32 vs float64

Feature	float32	float64
Size	4 bytes	8 bytes
Bits	32-bit	64-bit
Precision	~6–7 decimal digits	~15–16 decimal digits
Range	Smaller	Much larger
Memory usage	Less	More
Default in Go	No	Yes
Accuracy	Less accurate	More accurate

Rune

What is Rune?

- Rune is built-in data type that represents a single Unicode code point

ASCII Character

Unicode Character

What are Unicode Characters?

Byte vs rune

Type	Use Case
byte	File, network data
rune	Unicode character
string	Name, email, phone

Summary

Scenario	Correct Type
Age	uint8
Temperature	int8
User ID	int64
File data	byte
Phone number	string
GPS	float64
Port number	uint16
Unicode char	rune

Practice Scenario: Database User ID

User ID: 9834567891

Answer:

Type: int64

Reason: Value is very large, exceeds int32 limit

Practice Scenario: Age Storage

Age range: 0 to 120

Answer:

Type: uint8

Reason: Age is never negative and fits in 0–255

Practice Scenario: GPS Location

Latitude: 19.076090

Answer:

Type: float64

Reason: GPS requires high precision

Practice Scenario: Bank Balance

Balance: 1000.75

Answer:

Type: float64

Reason: Decimal values present

Interview Questions

You are designing a backend system.

Which type is BEST for database User ID?

Options:

A. int8

B. uint32

C. int64

D. uint8

Answer

Correct Answer: C. int64

Explanation:

Database BIGINT = signed int64

Industry standard for IDs

Main Function

- package main → the starting package of the application.
- func main → where program execution starts
- Why fmt package has no main function?
    - *Answer: it is a library, not an executable program.*
- Library : fmt (no main function required)
- Executable: Go Project (requires main to execute as program)

Libraries

Fmt

Math

Practice Problem

Using break and continue, write program for

- Print odd numbers till 10
- Print using infinite loop till 5

Array, Slices, Maps

Arrays

- Take 5 integers from user

Slices

Maps

Array, Slices, Maps

Array

An array in Go is a fixed-size collection of elements of the same type stored in contiguous memory locations.

Slices

A slice in Go is a dynamic, flexible view over an underlying array that can grow and shrink in size.

Maps

A map in Go is a collection of key-value pairs where each key is unique and used to quickly access its associated value.

Practice Questions

1) Create a new array that contains three hobbies you have

Output (print) that array in the command line.

2) Also output more data about that array:

- The first element (standalone)

- The second and third element combined as a new list

3) Create a slice based on the first element that contains

the first and second elements.

Create that slice in two different ways (i.e. create two slices in the end)

4) Re-slice the slice from (3) and change it to contain the second

and last element of the original array.

5) Create a "dynamic array" that contains your course goals (at least 2 goals)

6) Set the second goal to a different one AND then add a third goal to that existing dynamic array

7) Bonus: Create a "Product" struct with title, id, price and create a

dynamic list of products (at least 2 products).

Then add a third product to the existing list of products.

Problem Statement: Uber

You are building a simplified Uber backend system to **manage drivers and rides.**

Design the system by choosing appropriate Go data structures (array, slice, or map) for each requirement below.

Requirements

1. Track Ride Count Per Hour (Last 24 Hours)
    a. *Store number of rides completed in each of the last 24 hours.*
2. Store Available Drivers
    a. *Maintain a list of drivers who are currently online.*
    b. *Drivers can come online or go offline anytime.*
3. Store Active Rides
    a. *Each ride contains*:
    b. *rideID, userID, driverID*
    c. *Store all ongoing rides.*

Task: For each requirement, choose: array, slice, maps

Problem Statement: Uber

1. Find Driver by DriverID Quickly
    a. *Given a driverID, retrieve driver details instantly.*
2. Track Total Rides Completed Per Driver
    a. *Example:*
    b. *DR1 → 15 rides*
    c. *DR2 → 8 rides*
3. Store Last 5 Locations of Each Driver
    a. *Each driver has exactly 5 recent location coordinates.*

Task: For each requirement, choose: array, slice, maps

Solution

Ride count per hour (24 hrs)	**array**	Fixed size (always 24)
Available drivers	**slice**	Dynamic (drivers come/go)
Active rides	**slice**	Dynamic (rides start/end)
Find driver by driverID	**map**	Fast lookup using key
Rides completed per driver	**map**	Key-value counting
Last 5 locations per driver	**map of array**	Map → find driver, Array → fixed 5 locations

make()

- make() is a built-in function used to initialize slices, maps, and channels by allocating memory and preparing their internal structure.
- If you don't initialize them, they are nil, and writing to them will cause runtime errors.
  - *panic: assignment to entry in nil map*
- make() is used only for:
  - *slice*
  - *map*
- make() is not used for:
  - *arrays* → size

make([]int, 2, 9)

make()

Without make():

- slice → nil → unusable
- map → nil → unusable

With make():

- slice → memory ready → usable
- map → memory ready → usable

Arrays, Slices, Maps: Range & loop

Syntax:

- for index, value := range arr

Interview Questions

Q1. Why are arrays rarely used directly in backend Go, but slices are used everywhere?

Q2. What is the difference between nil slice and empty slice?

Q3. Why does Go provide both length and capacity in slices?

Q4. Why does writing to a nil map cause panic, but reading does not?

Q5. When should you use array instead of slice in real backend systems?

Interview Questions: Solutions

Q1. Why are arrays rarely used directly in backend Go, but slices are used everywhere?

- Arrays are fixed-size and inflexible, while slices are dynamic and can grow, which matches real backend data needs

Q2. What is the difference between nil slice and empty slice?

- A nil slice has no underlying array, while an empty slice has an allocated array but zero elements.

Q3. Why does Go provide both length and capacity in slices?

- Length shows current elements, while capacity shows how many elements can be added without reallocating memory.

Q4. Why does writing to a nil map cause panic, but reading does not?

- Reading returns default value safely, but writing fails because nil map has no allocated memory.

Q5. When should you use array instead of slice in real backend systems?

- Use array when size is fixed and must not change, ensuring predictable memory and performance.

Interview Questions

What will be the output?

```
a := [3]int{1,2,3}

b := a

b[0] = 100


fmt.Println(a)
```

Interview Questions

What will be the output?

```
s := []int{1,2,3}

t := s


t[0] = 100


fmt.Println(s)
```

Interview Questions

What will be the output?

```
s := make([]int, 3)


s = append(s, 10)


fmt.Println(len(s), cap(s))
```

Interview Questions

What will happen?

var m map[string]int

m["a"] = 10

Interview Questions

What will be the output?

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


fmt.Println(m["unknown"])
```

Interview Questions

1. Why does slice append sometimes modify original array and sometimes create new array?
2. Why is slice faster than map for iteration, but map is faster for lookup?
3. Why does make() exist when append() can grow slice automatically?

Interview Questions: Solutions

1. Why does slice append sometimes modify original array and sometimes create new array?
    a. *append modifies the original array if slice has spare capacity; otherwise, it allocates a new array.*
2. Why is slice faster than map for iteration, but map is faster for lookup?
    a. *Slice is faster for iteration because memory is contiguous, while map is faster for lookup because it uses hashing for direct access.*
3. Why does make() exist when append() can grow slice automatically?
    a. *make is needed to initialize maps and to preallocate memory for performance, which append alone cannot do.*

- Go Intro
- Fundamentals →
- Data Str → Array, Slice, Maps

OOPS
- Structs & Pointers

Structs & Custom Types

Structs

- ~~Why Structs~~
- ~~What are Structs~~
- ~~Creating Structs~~ & Use
- Adding methods to structs

Structs

Why

What is structs?

- A struct is a collection of fields.
- Struct fields are accessed using a dot

int[]

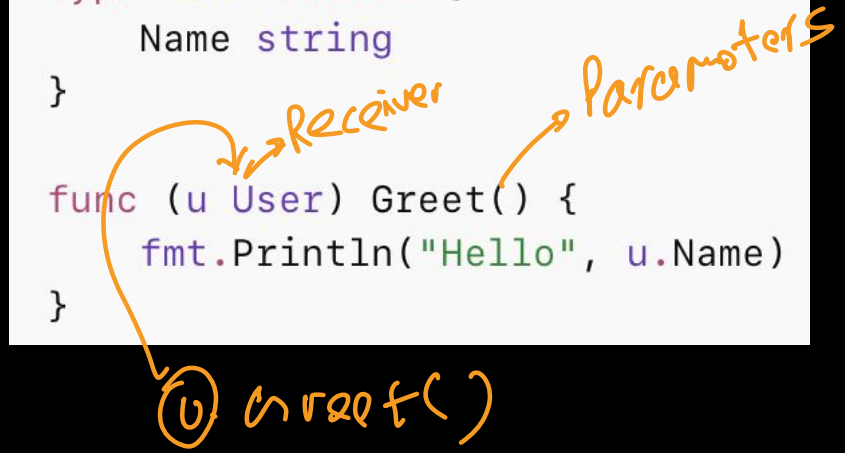
[] int

Functions vs Methods

Function: A function is a standalone block of code not attached to any type

Method: A method is a function **attached to a struct** (or type) **via receiver**

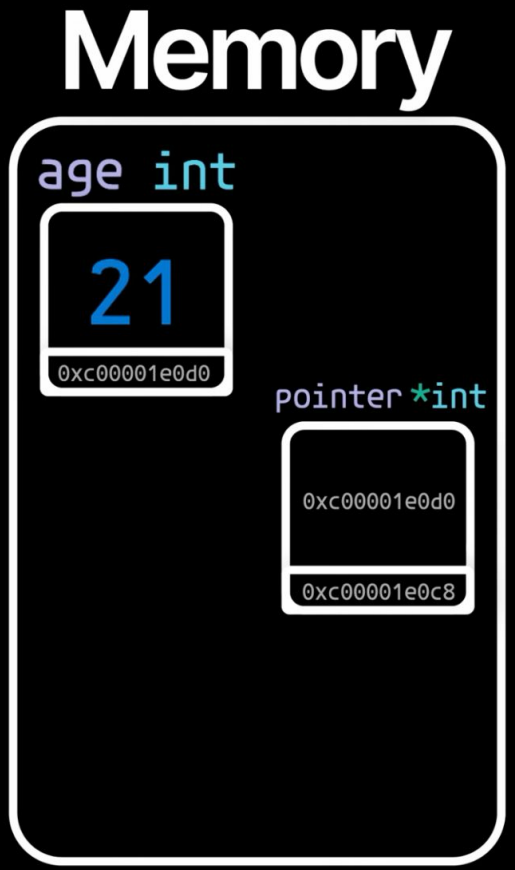
```go
type User struct {
    Name string
}

func (u User) Greet() {
    fmt.Println("Hello", u.Name)
}
```



Pointers

Pointers

```
var age int
age = 42
fmt.Println(age)
var pointer *int
fmt.Println(&age)
pointer =  &age
fmt.Println( *pointer )
*pointer = 21
fmt.Println(age)
```



Pointers

Why Pointers?

- Avoid unnecessary copies (large copies)
- Directly **mutate values**
- Less code

What is a pointer?

- A pointer is a variable that stores the memory address of another variable, not the value itself

Pointers

Declaration of pointer

Dereferencing pointer

&

*

Interview Questions

1. How to declare pointer?
2. How to store value in pointer
3. How to perform Dereferencing?
4. Meaning of & symbol?
5. Meaning of * symbol?
6. What is default value of pointer?

Interview Question

1. How Go passes arguments?

Functions vs Methods

Function	Method
Standalone block of code	Function attached to struct/type
No receiver	Has receiver
Not part of OOP behavior	Enables OOP behavior

Practice Question

‹/› Go

```go
package main

import "fmt"

func update(x int) {
    x = 20
}

func main() {
    a := 10
    update(a)
    fmt.Println(a)
}
```

Interview Question

```go
package main

import "fmt"

func main() {
    x := 5
    p := &x
    pp := &p

    **pp = 50

    fmt.Println(x)
}
```

Interview Question

```
func main() {

    var p *int

    *p = 10

}
```