Practice Problem

Using break and continue, write program for

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

break

continue

for :

if n%.2 == 0

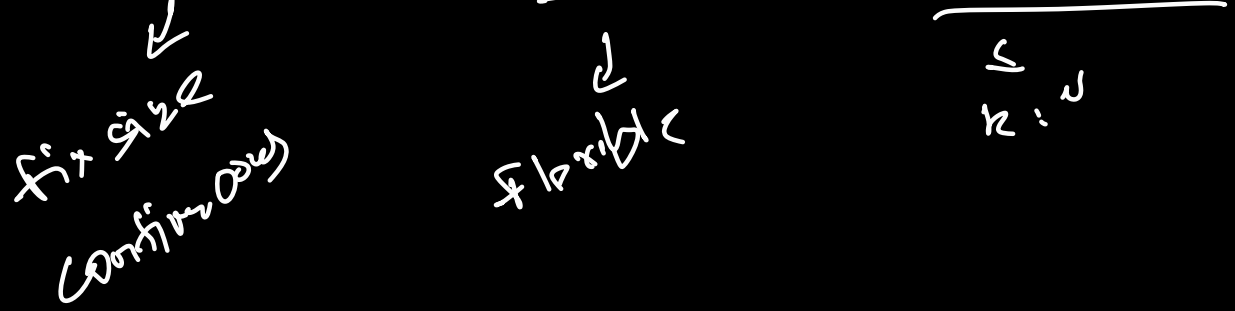
for i: range 10

for { if ——

break

}

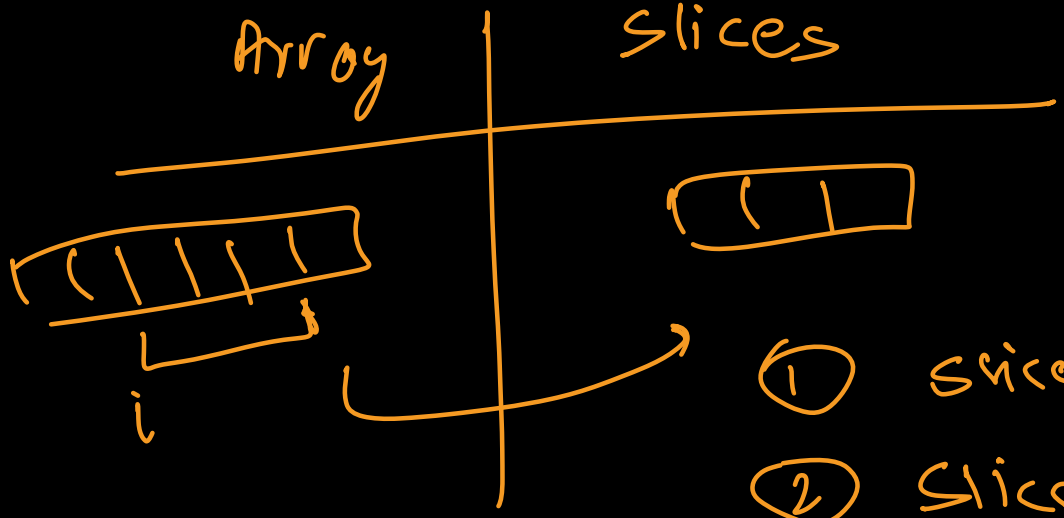
Array, Slices, Maps



Arrays

- Take 5 integers from user

Slices



1. slice on Array.
2. Slice to dynamic Array.

Ono

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

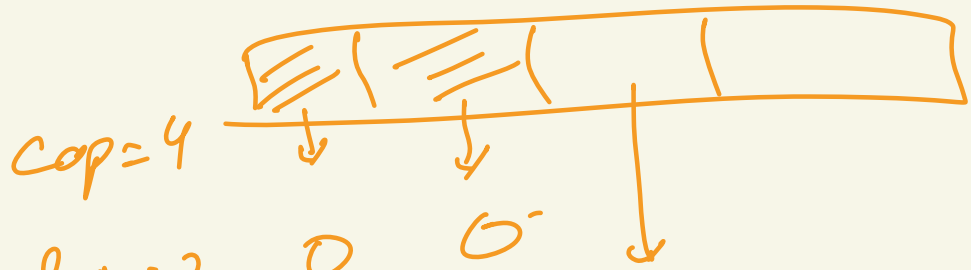
A map in Go is a collection of key-value pairs where each key is unique and used to quickly access its associated value.

arr

slice.

make([]int, 2, 4)

↳ memory



append( ) 3.

mutate. of sliced array.

Original array changed?

yes | NO

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.*

→ Array ? | Slice ✓

→ slice

→ Map

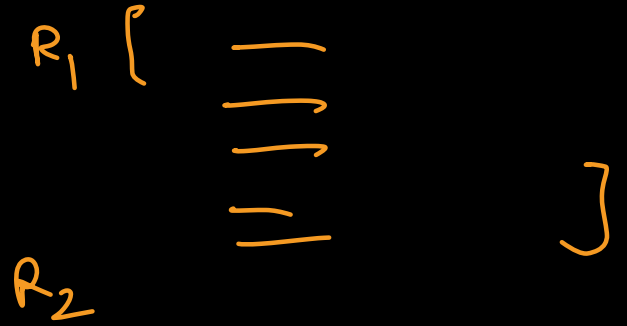
Task: For each requirement, choose: array, slice, maps

Problem Statement: Uber

1. Find Driver by DriverID Quickly
   a. *Given a driverID, retrieve driver details instantly.* → map
2. Track Total Rides Completed Per Driver
   a. *Example:*
   b. *DR1 → 15 rides*
   c. *DR2 → 8 rides*

   → Array / map
3. Store Last 5 Locations of Each Driver
   a. *Each driver has exactly 5 recent location coordinates.* → Array / slice / ~~map~~

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*

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.*