**Declaring an array:**

package main

import "fmt"

func main() {

var array\_int[10] int

var array\_float[10] float32

var array\_string[10] string

var array\_complex[10] complex128

fmt.Println("Int type array:", array\_int)

fmt.Println("Float type array:", array\_float)

fmt.Println("String type array:", array\_string)

fmt.Println("Complex type array:", array\_complex)

}

**Initialize an array:**

var arr [5] int = [5] int{1, 2, 3, 4 , 5}

fmt.Println(arr) // [1 2 3 4 5]

var arr [5]int

arr[0] = 1

arr[1] = 2

arr[2] = 3

arr[3] = 4

arr[4] = 5

fmt.Println(arr) // [1 2 3 4 5]

arr := [5]int{1, 2, 3, 4, 5}

fmt.Println(arr) // [1 2 3 4 5]

var arr [5]int

arr[1] = 2

arr[3] = 4

fmt.Println(arr) // [0 2 0 4 0]

arr := [5]int{1, 2, 3}

fmt.Println(arr) // [1 2 3 0 0]

arr := [...] int {1, 4, 7, 2, 7, 9, 3, 5}

fmt.Println(arr) // [1 4 7 2 7 9 3 5]

**Need of an array:**

package main

import "fmt"

func main() {

var name1 string = "aman"

var name2 string = "rahul"

var name3 string = "geet"

var name4 string = "preeti"

fmt.Printf("Names are %q, %q, %q and %q", name1, name2, name3, name4) //Names are "aman", "rahul", "geet" and "preeti"

}

package main

import "fmt"

func main() {

names:=[...] string{"aman", "rahul", "geet", "preeti"}

fmt.Printf("Names are %q", names)

}

**Multidimensional array:**

arr := [3][2] int {

{1, 2},

{4, 5},

{7, 8},

}

fmt.Println(arr) // [[1 2] [4 5] [7 8]]

arr := [3][2] int {

{1, 2},

{7},

}

fmt.Println(arr) // [[1 2] [7 0] [0 0]]

var arr [3] [3] int

arr[0] = [3] int {1, 2, 3}

arr[1] = [3] int {4, 5, 6}

arr[2] = [3] int {7, 8, 9}

fmt.Println(arr) //[[1 2 3] [4 5 6] [7 8 9]]

**Length of an array:**

func main() {

arr1D := [...] int {1, 4, 7, 2}

arr2D := [3][2] int {

{1, 2},

{7},

}

fmt.Println("Length of 1D array", len(arr1D))

fmt.Println("Length of 2D array", len(arr2D)) //tells number of rows it can have.

}

**Array comparison:**

func main() {

arr1 := [4] int {1, 4, 7, 2}

arr2 := [4] int {1, 4, 7}

arr3 := [3][2] int {

{1, 2},

{7},

}

arr4 := [3][2] int {

{1, 1},

{7},

}

arr5 := [4] int{1, 4, 7, 0}

fmt.Println("arr1 and arr2", arr1==arr2)

fmt.Println("arr3 and arr4", arr3==arr4)

fmt.Println("arr2 and arr5", arr2==arr5)

}

**Copying an array:**

func main() {

arr1 := [4] int {1, 4, 7, 2}

arr2 := arr1

arr2[2] = 90

fmt.Println("arr1 is", arr1) // [1 4 7 2]

fmt.Println("arr2 is", arr2) // [1 4 90 2]

}

**Passing array to functions:**

func main() {

arrval := [4] int {1, 4, 7, 2}

fmt.Println("sum of array elements is:", sum(arrval))

}

func sum(arr [4]int) int{

sum := 0

for i:=0; i<len(arr); i++{

sum = sum + arr[ i ]

}

return sum

}

**Iteration over an array:**

func main() {

arr :=[...] int {1, 4, 7, 2, 7, 9, 3, 5}

sum := 0

for i:=0; i<len(arr); i++{

sum = sum +arr[i]

}

fmt.Println("sum of values:", sum)

}

func main() {

arr := [...] int {1, 4, 7, 2}

sum := 0

for index, value := range arr{

fmt.Println("index is:", index)

fmt.Println("value is:", value)

sum = sum + value

}

fmt.Println("sum of values:", sum)

}

func main() {

arr := [...] int {1, 4, 7, 2}

sum := 0

for \_, value := range arr{

sum = sum + value

}

fmt.Println("sum of values:", sum) // sum of values: 14

}

func main() {

var size int = 4

arrval := [size] int {1, 4, 7, 2}

fmt.Println("sum of array elements is:", arrval)

}

**Initialize slices:**

func main() {

var slice1 [] int = [] int {1, 2, 3, 5}

slice2 := [] int {1, 2, 3, 5}

fmt.Println("slice with literal", slice1) // [1 2 3 5]

fmt.Println("slice short hand declaration", slice2) // [1 2 3 5]

}

**Slices from an array:**

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[3:7]

slice2 := arrayval[:6]

slice3 := arrayval[5:]

slice4 := arrayval[:]

fmt.Println("slice from array:", slice1)

fmt.Println("slice with default low indice:", slice2)

fmt.Println("slice with default high indice:", slice3)

fmt.Println("slice with default low and high indice:", slice4)

}

**Slices from slice:**

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[1:8]

slice2 := slice1[:6]

fmt.Println("slice from array:", slice1)

fmt.Println("slice from existing slice", slice2)

}

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[1:8]

slice2 := slice1[:6]

fmt.Println("slice from array:", slice1)

fmt.Println("slice from existing slice", slice2)

arrayval[2] = 90

fmt.Println("slice from array after modifying an array:", slice1)

fmt.Println("slice from existing slice after modifying an array", slice2)

}

**Slices from structs:**

func main() {

record := []struct {

roll\_no int

name string

present bool

}{

{1, "Rahul", true},

{3, "Akansha", false},

{5, "Ram", true},

}

fmt.Println("sructs type slice is:",record) //sructs type slice is: [{1 Rahul true} {3 Akansha false} {5 Ram true}]

}

**Pointer:**

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[1:8]

slice2 := slice1[:6]

fmt.Println("original array", arrayval)

fmt.Println("slice from array:", slice1)

fmt.Println("slice from existing slice", slice2)

slice1[0] = 178

fmt.Println("array after modifying slice", arrayval)

fmt.Println("slice from array after modifying slice:", slice1)

fmt.Println("slice from existing slice after modifying slice", slice2)

}

**Length:**

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[1:8]

fmt.Println("length of slice", len(slice1)) //length of slice 7

}

**Capacity:**

func main() {

var arrayval [10] int = [10] int {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

slice1 := arrayval[2:5]

fmt.Println("length of slice", len(slice1)) // length of slice 3

fmt.Println("capacity of slice", cap(slice1)) // capacity of slice 8

}

**Slice using make function:**

func main() {

slice1 := make([] int, 2,4)

fmt.Println("length of slice", len(slice1)) // length of slice 2

fmt.Println("capacity of slice", cap(slice1)) // capacity of s/lice 4

fmt.Println("value of slice", slice1) // value of slice [0 0]

slice1[0] = 194

slice1[1] = 18

fmt.Println("value of slice", slice1) // value of slice [194 18]

slice1[2] = 78 // error as length of slice is 2

}

**Extending slice:**

func main() {

slice1 := []int {12, 65, 34, 78, 23, 67}

fmt.Println("capacity of slice1:", cap(slice1))

slice2 := make([]int, len(slice1), (cap(slice1)+1)\*2)

for i:= range slice1{

slice2[i] = slice1[i]

}

slice1 = slice2

fmt.Println("capacity of slice1:", cap(slice1))

}

**Two dimensional slices:**

slice1 := [][] int{

[] int {1, 2},

[] int {3, 4},

[] int {5, 6},

}

func main() {

slice1 := [][] int{

[] int {1, 2, 78, 54},

[] int {0, 3, 4},

[] int {5, 6},

}

fmt.Println("slice1", slice1) // slice1 [[1 2 78 54] [0 3 4] [5 6]]

slice2 := make([][]string, 4)

slice2[0] = []string{"College", "School"}

slice2[1] = []string{"University", "Institute"}

fmt.Println("slice2 using make function:", slice2) // slice2 using make function: [[College School] [University Institute] [] []]

}

**Copying slice using copy function:**

func main() {

slice1 := [] int{23, 56, 47, 58}

slice2 := [] int{90, 107, 0, 67, 28, 89}

// if(len(des\_slice)>len(src\_slice))

fmt.Println("destination slice if it's length is greater than source slice>>>>>>>>>>>>>")

fmt.Println("destination slice before copying elements:", slice2)

n := copy(slice2, slice1)

fmt.Println("destination slice after copying elements:", slice2)

fmt.Println("length of destination slice after copying elements:", len(slice2))

fmt.Println("Numnber of copied elements after copying elements:", n)

// if(len(des\_slice) = len(src\_slice))

slice3 := [] int{34, 20, 1, 91}

fmt.Println("destination slice if it's length is equal to source slice>>>>>>>>>>>>>")

fmt.Println("destination slice before copying elements:", slice3)

n = copy(slice3, slice1)

fmt.Println("destination slice after copying elements:", slice3)

fmt.Println("length of destination slice after copying elements:", len(slice3))

fmt.Println("Numnber of copied elements after copying elements:", n)

// if(len(des\_slice)<len(src\_slice))

slice4 := [] int{34, 20}

fmt.Println("destination slice if it's length is less than source slice>>>>>>>>>>>>>")

fmt.Println("destination slice before copying elements:", slice4)

n = copy(slice4, slice1)

fmt.Println("destination slice after copying elements:", slice4)

fmt.Println("length of destination slice after copying elements:", len(slice4))

fmt.Println("Numnber of copied elements after copying elements:", n)

// copying in nil slice

slice5 := []int{}

fmt.Println("On copying elements to nil slice >>>>>>>>>>>>>")

fmt.Println("destination slice before copying elements:", slice5)

n = copy(slice5, slice1)

fmt.Println("destination slice after copying elements:", slice5)

fmt.Println("length of destination slice after copying elements:", len(slice5))

fmt.Println("Numnber of copied elements after copying elements:", n)

}

**Sort a slice:**

package main

import ("fmt"

"sort"

)

func main() {

slice1 := [] int{23, 0, 105, 58}

slice2 := [] string{"Ram", "Akansha", "Mayank"}

slice3 := [] float64{75.84, 0.007, 85.62, 0.74637}

fmt.Println("slice1 before sorting:", slice1)

sort.Ints(slice1)

fmt.Println("slice1 after sorting:", slice1)

fmt.Println("slice1 before sorting:", slice2)

sort.Strings(slice2)

fmt.Println("slice1 after sorting:", slice2)

fmt.Println("slice1 before sorting:", slice3)

sort.Float64s(slice3)

fmt.Println("slice1 after sorting:", slice3)

}

**Compare slices:**

package main

import (

"fmt"

"reflect"

)

func main() {

slice1 := [] int{23, 0, 105, 0}

slice2 := [] string{"Ram", "Akansha", "Mayank"}

slice3 := [] float64{23, 0, 105, 0}

slice4 := make([]int, 4)

slice4[0] = 23

slice4[1] = 0

slice4[2] = 105

fmt.Println("Is slice1 equal to slice2:", reflect.DeepEqual(slice1, slice2))

fmt.Println("Is slice1 equal to slice3:", reflect.DeepEqual(slice1, slice3))

fmt.Println("Is slice1 equal to slice4:", reflect.DeepEqual(slice1, slice4))

}

**Searching in slice:**

package main

import ("fmt"

"sort"

)

func main() {

slice1 := [] int{0, 9,16,105, 345}

slice2 := [] string{"Akansha", "Mayank", "Ram"}

slice3 := [] float64{0, 0.001, 5.7}

fmt.Println(sort.SearchInts(slice1, 105)) // 3

fmt.Println(sort.SearchInts(slice1, 78)) // 3

fmt.Println(sort.SearchInts(slice1, 0.0)) // 0

fmt.Println(sort.SearchStrings(slice2, "Ram")) // 2

fmt.Println(sort.SearchStrings(slice2, "ram")) // 3

fmt.Println(sort.SearchFloat64s(slice3, 0.000)) // 0

fmt.Println(sort.SearchFloat64s(slice3,5.7)) // 2

}

**Reverse slice:**

package main

import ("fmt"

"sort"

)

func main() {

slice1 := [] int{0, 888, 9,16,105, 345}

slice2 := [] string{"Soni", "Akansha", "Charu", "Mayank", "Ram"}

sort.Sort(sort.Reverse(sort.IntSlice(slice1)))

fmt.Println("reverse of slice1:", slice1)

sort.Sort(sort.Reverse(sort.StringSlice(slice2)))

fmt.Println("reverse of slice2:", slice2)

}

**Appending slice element:**

func main() {

slice1 := [] int{0, 888, 9, 16, 105, 345}

slice1 = append(slice1, 345, 646, 0)

fmt.Println("slice1 after appending few elements:", slice1)

slice3 := make([]int, 2)

slice3 = append(slice3, 56, 67, 48, 90, 46, 26, 57)

fmt.Println("slice3 after appending elements more than it's capacity:", slice3)

slice4 := []int{}

slice4 = append(slice4, 45, 67)

fmt.Println("slice4 after appending elements in nil slice:", slice4)

}

func main() {

slice1 := [] int{0, 888, 9, 16, 105, 345}

slice2 := []int{78, 34, 9, 57}

slice1 = append(slice1, slice2...)

fmt.Println("slice1 after appending slice2", slice1)

}