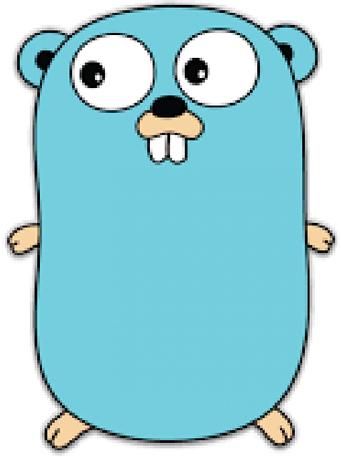
KBA



Go Programming Language

Introduction

KBA

Go is a statically-typed, compiled programming language designed for simplicity and efficiency.

Developed by 

Github Stats

Go used by other internet giants

Agenda

- Project Setup
- Variables
- Functions and Control Structures
- Structs and Interfaces
- Pointers
- Methods

Project Setup

K B A

1. To initialize the go project , cd into the directory to project folder

2. Now enter : go mod init <project name>

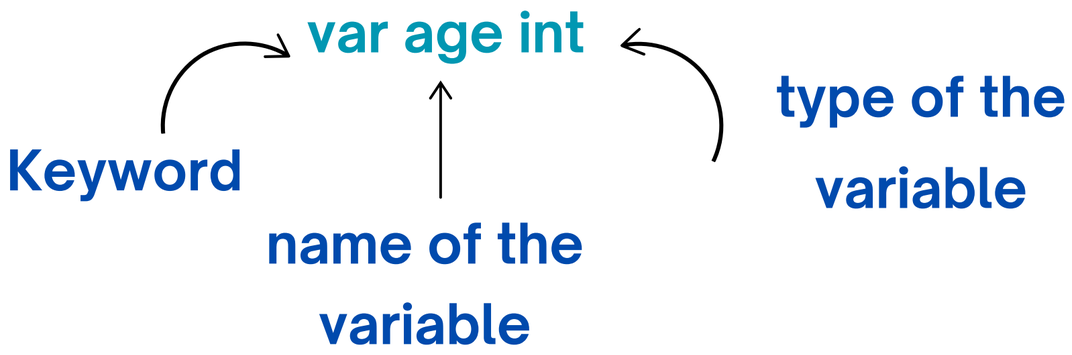
3. Now Every go project need a main package with an entry function named main, which we define in the primary file.

```
package main
import "fmt"
func main() {
    fmt.Println("Hello World!")
}
```

4. For executing the code use, go run <file name>

Initializing a variable

Go is a statically typed language so we have to specify the type of variable we are declaring




```
var user = "Tommy"      // declaration along with initialisation
```

```
email := "email@example.com" //shorthand for declaration with initialisation
```

Input and Output Statements

KBA

```
package main
import "fmt"
func main() {
   var name string
   fmt.Println("Enter Your Name")
   fmt.Scan(&name)
   fmt.Println("Name: ",name)
}
```

Output:

```
Enter Your Name
Jain
Name: Jain
```

Datatypes and Default Values

KBA

	Default Values
bool boolean string int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 uintptr byte // alias for uint8 rune // alias for int32 float float32 float64 complex complex64 complex128	• 0 for numeric types,float • false for the boolean type • "" (the empty string) for strings.

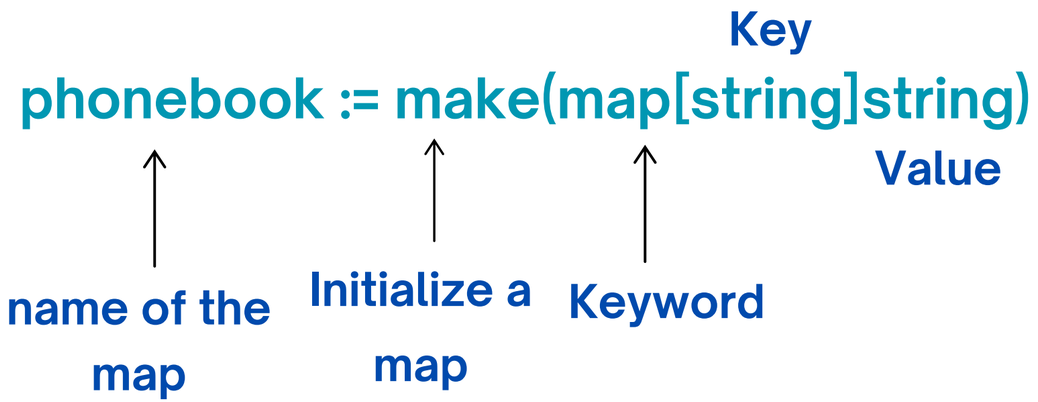
Array and Slices

K B A

Array	Slice
Collection of same type of elements of fixed size. array := [5]int{1,2,3,4,5}	Collection of same type of elements of dynamic size. slice := []int{1,2,3,4,5,6,7}

Maps

Maps are used to store data values in key:value pairs.




```
phonebook := map[string]string{"Tony": "5764784"}   // declaration along with initialisation

phonebook["Tommy"] = "123435"   //Add a key value pair
```

Control Structures

- IF -ELSE
- FOR loop
- SWITCH
- DEFER

DEFER

KBA

defer is used to delay the execution of a function or statement until the surrounding function completes.

It’s often used for cleanup tasks like closing files or releasing resources.

```
package main
import "fmt"
func main() {
    fmt.Println("Start cooking")
    // Defer cleanup
    defer fmt.Println("Clean up the kitchen")
    fmt.Println("Finish cooking")
}
```

Output

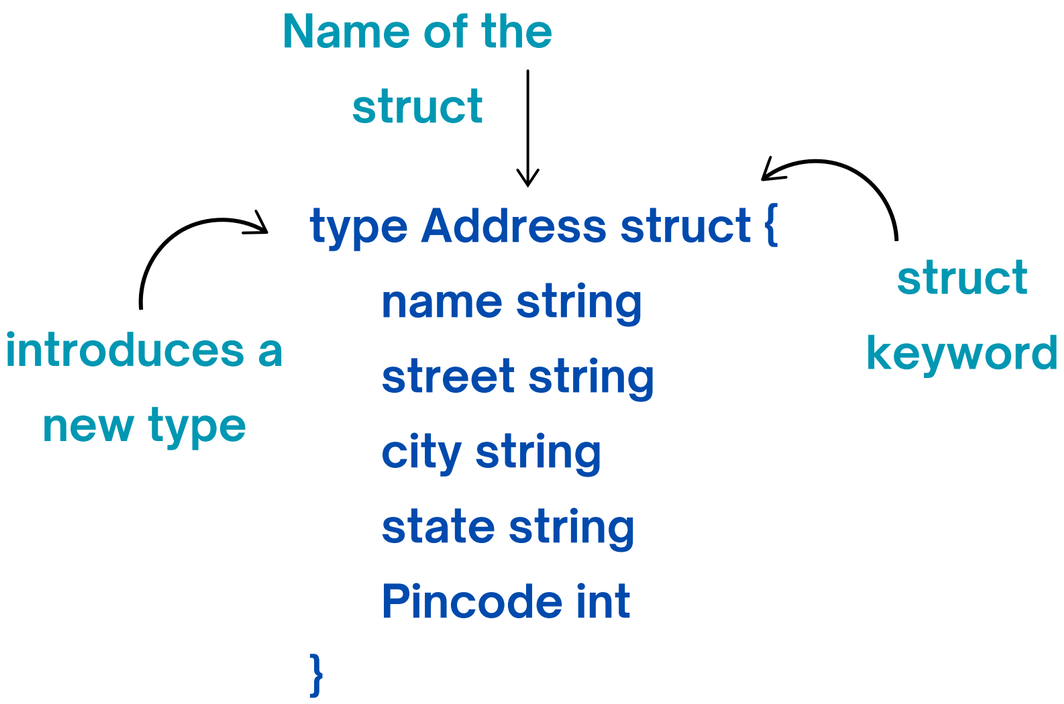
```
Start cooking
Finish cooking
Clean up the kitchen
```

Structs

K B A

struct is a user-defined type that allows to group items of possibly different types into a single type.

Declare a struct



Initialize a struct

```
var a = Address{"Akash", "Kazhakootam",  "Trivandrum", 690081}
```

Structs

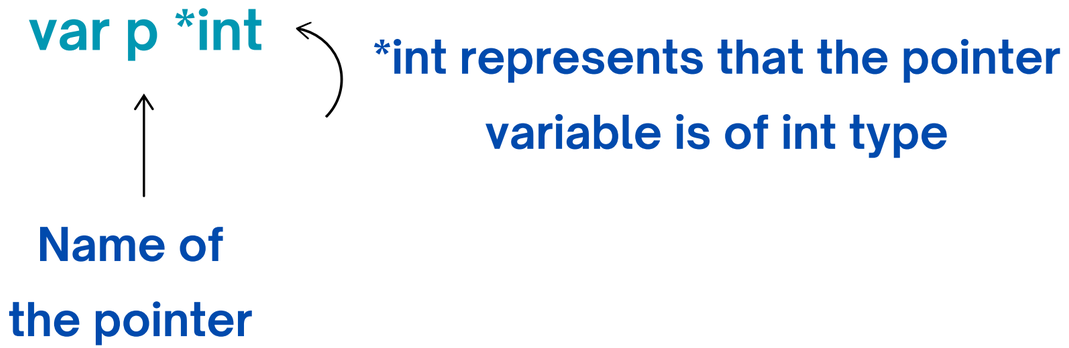
```
type Employee struct{
        Name string
        Id int
        Email string
        Department string
        }
u1:=
Employee{"Alice",2006,"alice@gmail.
com","Fabric"}
fmt.Println("Employee Details",u1)
fmt.Println("Employee
Name",u1.Name)
```

Output:

```
Struct in Golang
Employee Details {Alice 2006
alice@gmail.com Fabric}
Employee Name Alice
```

Pointer

Pointer is a variable that holds the memory address of another variable.




```
var p *int
i := 42
p = &i
fmt.Println(*p)
*p = 21
fmt.Println(i)
```

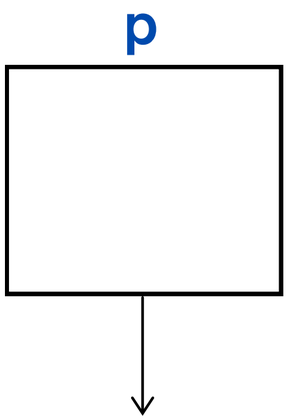
Output

```
42
21
```

How Pointers Work

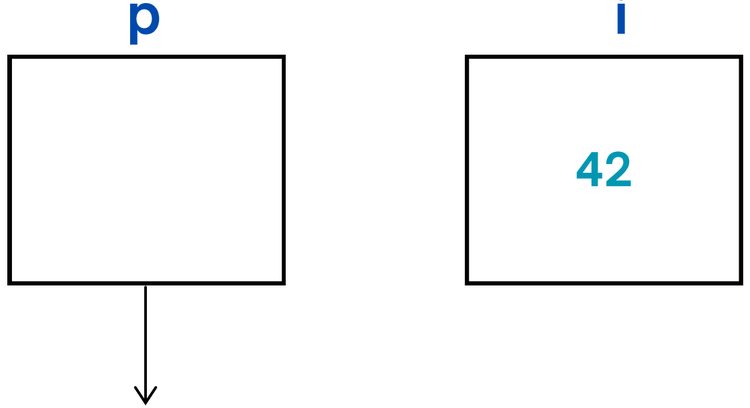
KBA

var p *int



var p *int

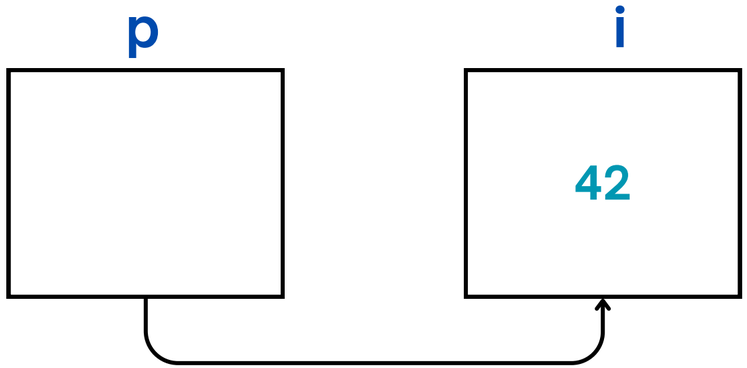
i := 42



var p *int

i := 42

p = &i

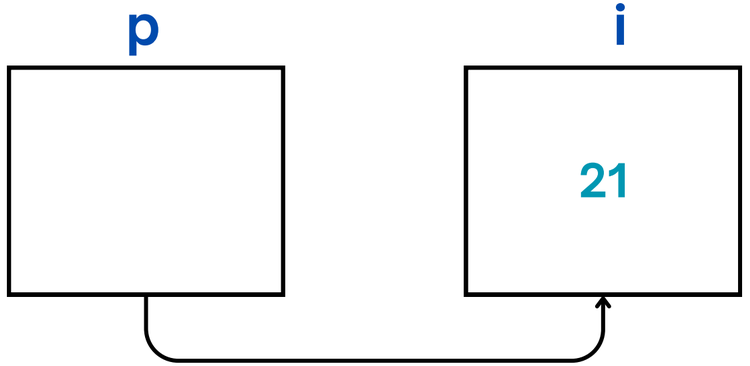


var p *int

i := 42

p = &i

*p = 21



Method

Method is just a function with a special receiver type between the func keyword and the method name. The receiver can either be a struct type or non-struct type.

```
type Employee struct {
 name     string
 salary   int
 currency string
}
func (e Employee) displaySalary() {
 fmt.Printf("Salary of %s is %s%d", e.name, e.currency, e.salary)
}
```

e -> Receiver

Employee -> Receiver type

Functions and Methods

K B A

Functions

```
type Vertex struct {
 a1, b1 int
}
func add(v Vertex) int{
 return(v.a1 + v.b1)
}
func main() {
 v := Vertex{20,50}
 fmt.Println(add(v))
}
```

Methods

```
type Vertex struct {
 a1, b1 int
}
func (v Vertex) add() int {
 return(v.a1 + v.b1)
}
func main() {
 v := Vertex{20,50}
 fmt.Println(v.add())
}
```

Value Receivers vs Pointer Receivers

KBA

Value Receivers	Pointer Receivers

type Employee struct {
name string
age int
}
func (e Employee)
changeName(newName string) {
e.name = newName
}
	type Employee struct {
	name string
	age int
	}
	func (e *Employee)
	changeName(newName string) {
	e.name = newName
	}

JSON

KBA

JSON is a widely used format for data interchange.

encoding/json is the package used to encode/decode JSON data.

Marshal function is used to convert data into json

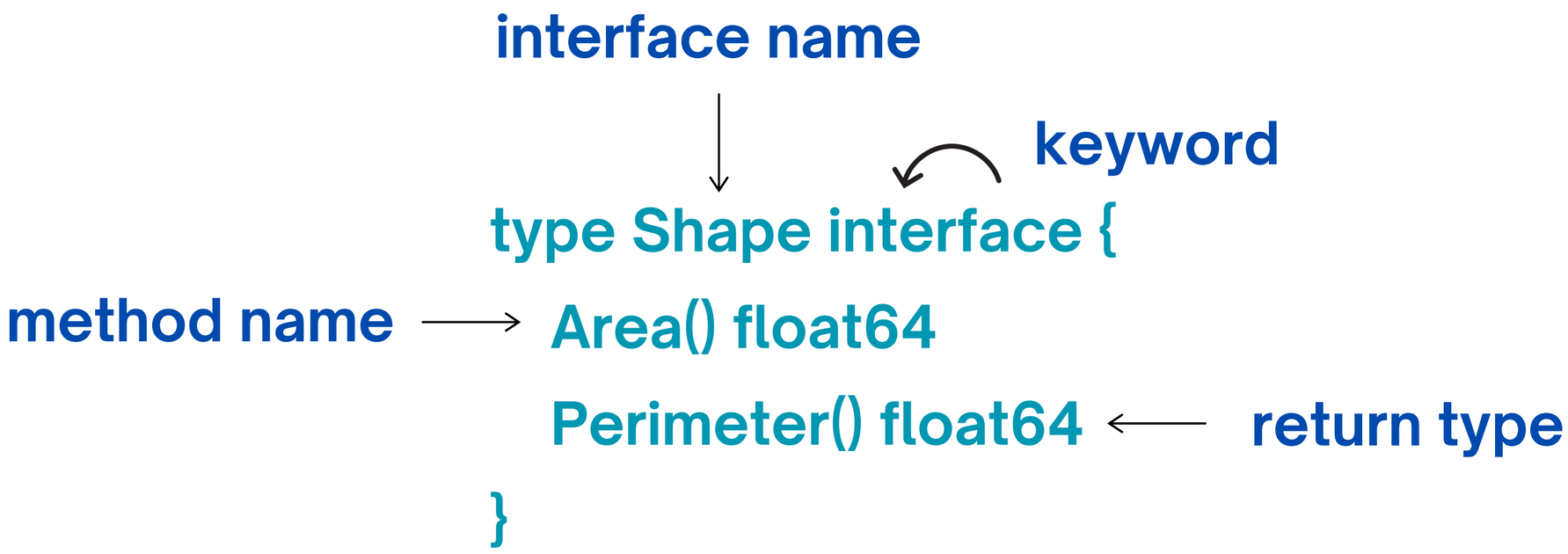
```
type Person struct {
 Name    string `json:"name"`
 Age     int    `json:"age"`
 Address string `json:"address"`
}
func main() {
  p1 := Person{Name: "John", Age: 30,
Address: "123 Main St"}
 jsonData, err := json.Marshal(p1)
}
```

Unmarshal function is used to decode json data

```
type Person struct {
 Name string `json:"name"`
 Age int `json:"age"`
 Address string `json:"address"`
}
func main() {
jsonString :=
`{"Name":"Alice","age":25,"address":"456 Elm St"}`
var p2 Person
err = json.Unmarshal([]byte(jsonString), &p2)
}
```

Interface

Interface is a type that lists methods without providing their code. To implement an interface, a type must define all methods declared by the interface.



Importing a File in Go

KBA

To import a file we have to specify the package name and the folder name in which the file is located.

Even if multiple files are there (it should be with the same package name) it will import all the files in it

```
import ( "kbaauto/greetings")
```

Name of the module

Name of the folder

Unit Testing in Go

KBA

- A way of automated testing
- A unit test is a way of testing a unit of the code
- Tests are written in files that end with _test.go and are placed in the same package as the code they are testing.
- The test functions have a specific signature: they must start with Test and take a single argument of type *testing.T.