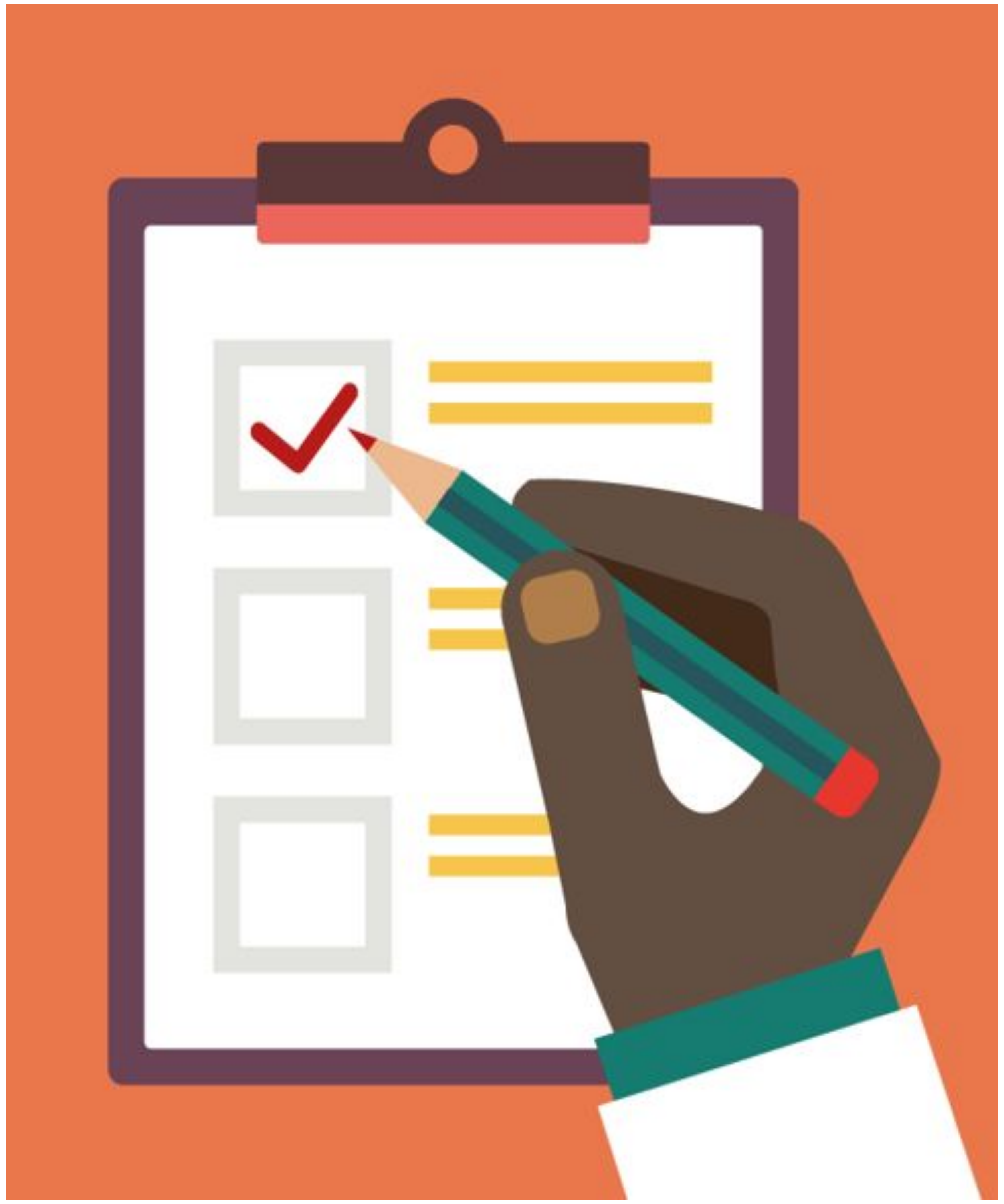
Welcome To

JOSH

DISRUPT YOUR INDUSTRY

covered…



- Methods
- Interfaces

Methods

- Go methods are similar to Go function with one difference, i.e, the method contains a receiver argument in it.
  Syntax:
  ```
  func(reciver_name Type) method_name(parameter_list)(return_type){
  // Code
  }
  ```

- What is a receiver?

- Points to note:
  - Receiver can be of struct type or non-struct type
  - Cannot declare a method with a receiver whose type is defined in another package (which includes the built-in types such as int).

Methods

- Pointer receivers:
  - Used to modify values in methods itself
  - Receiver type itself should not be a pointer

- Difference between a function that takes a pointer argument and a pointer receiver method

- Added advantage: For methods, pointer receivers and value receivers can be used interchangeably, but not a mixture of both on a type.

Interface

- In Go language, the interface is a custom type that is used to specify a set of zero or more method signatures
- Interfaces can be seen as a protocol or a contract. It doesn’t provide any implementation, it only describes the behaviour of a type
- To create interface use **interface** keyword, followed by curly braces containing a list of method names, along with any parameters or return values the methods are expected to have.

```
// Declare an Interface Type and methods does not have a body
type Employee interface {
   PrintName() string                  // Method with string return type
   PrintAddress(id int)                // Method with int parameter
   PrintSalary(b int, t int) float64 // Method with parameters and return type
```

Interface

- An interfaces act as a blueprint for method sets, they must be implemented in order to satisfy the interface.
- Unlike Java, Interfaces in golang are implicitly implemented.
- you are not allowed to create an instance of the interface. But you are allowed to create a variable of an interface type and this variable can be assigned with a concrete type that satisfies the interface.

Run In Go Playground

Advantages of having interfaces

- To help reduce duplication or boilerplate code.
- To make it easier to use mocks instead of real objects in unit tests.
- As an architectural tool, to help enforce decoupling between parts of your codebase.

Interface

Go Stringer interface example

- The Stringer interface is defined in the fmt package. Its String function is invoked when a type is passed to any of the print functions. We can customize the output message of our own types.

```
type Stringer interface {
    String() string
}
```

- This is the Stringer interface.

Run In Go Playground

Interfaces

Empty Interface

- An interface that has zero methods is called an empty interface. It is represented as interface{}.

- An empty interface is used to accept values of any type.
  The empty interface doesn't have any methods that are required to satisfy it, and so every type satisfies it.

  Syntax : `var temp` **`interface{}`**

  Run In Go Playground

Interfaces

Type Assertion

- A type assertion provides access to an interface's concrete value.

  Syntax : `t := i.(T)`

This statement asserts that the interface value i holds the concrete type T and assigns the underlying T value to the variable t.

- If **i** does not hold a **T**, the statement will trigger a panic.
- type assertion can return two values

  `t, ok := i.(T),` here **t** holds the underlying value and **ok** is a bool value indicating if assertion succeeded

Run In Go Playground

Interfaces

Type Switches

- A type switch performs several type assertions in series and runs the first case with a matching type

```
var x interface{} = "foo"

switch v := x.(type) {
case nil:
    fmt.Println("x is nil")            // here v has type interface{}
case int:
    fmt.Println("x is", v)             // here v has type int
case bool, string:
    fmt.Println("x is bool or string") // here v has type interface{}
default:
    fmt.Println("type unknown")        // here v has type interface{}
}
```

THANK YOU