**Basic Go types**

**Types and Interfaces**

Go is statically typed. Statically typed is a methodology for language design aiming to provide clarity and reduce confusing syntax and unexpected side effects

For example

Go allows no mixing of numeric types.

**var k float64 = 3.1634**

**var m int = 2**

**var result = k+m** // invalid operation: k + m (mismatched types float64 and int)

**var result2 = k\*m** // invalid operation: k \* m (mismatched types float64 and int)

**The Go Playground https://play.golang.org/**

**Constants**

**const** is a keyword introducing a name for a scalar value, a simple, unchanging value

**const str = "Hello" // str := string "Hello"**

**str** is an untyped string constant, just a value, not given a defined type that would force it to obey the strict rules that prevent combining differently typed values.

**Default type**

**str := "Hello"**

here we are not declaring the type, every constant value has a default type implicitly transferred when none is provided, here it is string

Like imports we can aggregate are const expressions

**const (**

**GET = "GET"**

**POST = "POST"**

**)**

**We can create our own types using Interfaces, this is the subject of this topic**

Interface types represent a fixed sets of methods.

An interface variable can store any concrete (non-interface) value as long as that value implements the interface's methods.

**type Handler interface {  
 ServeHTTP(ResponseWriter, \*Request)  
}**

An extremely important example of an interface type is the empty interface:

**interface{}**

It represents the empty set of methods and is satisfied by any value at all, since any value has zero or more methods.

**// Get Supported is the interface that provides the Get**

**// method a resource must support to receive HTTP GETs.**

**type GetSupported interface {**

**Get(url.Values, http.Header) (int, interface{}, http.Header)**

**}**

**// Post Supported is the interface that provides the Post**

**// method a resource must support to receive HTTP POSTs.**

**type PostSupported interface {**

**Post(url.Values, http.Header) (int, interface{}, http.Header)**

**}**

We can do this:

**type API struct {**

**mux \*http.ServeMux**

**muxInitialized bool**

**}**

**func (api \*API) requestHandler(resource interface{}) http.HandlerFunc {**

**return func(rw http.ResponseWriter, request \*http.Request) {**

**var handler func(url.Values, http.Header) (int, interface{}, http.Header)**

**switch request.Method {**

**case GET:**

**if resource, ok := resource.(GetSupported); ok {**

**handler = resource.Get }**

**case POST:**

**if resource, ok := resource.(PostSupported); ok {**

**handler = resource.Post }**

**}**

**}**

We can see that while Go's interfaces are not dynamically typed, the value stored in the interface variable may change type, that value will always satisfy the interface because an empty interface can hold any value

One important detail is that the pair inside an interface always has the form

(value, concrete type) and cannot have the form (value, interface type).

Interfaces do not hold interface values.

**Since almost anything can have methods attached, almost anything can satisfy an interface.**

<https://golang.org/doc/effective_go.html#interfaces_and_types>

This is an important syntax we need to learn as it is the basis for many package types implementation and behaviour.

**Type Binding**

**type API struct {**

**mux \*http.ServeMux**

**muxInitialized bool**

**}**

**// binding struct input function parameters**

**func (api \*API) AddResource(resource interface{}, paths ...string) {**

**for \_, path := range paths {**

**api.Mux().HandleFunc(path, api.requestHandler(resource))**

**}**

**}**

For example we can employ the ‘empty’ strategy using structs

**type Item struct { }**

**func (item Item ) Get(values url.Values, headers http.Header) (int, interface{}, http.Header) {**

**items := []string{"item1", "item2"}**

**data := map[string][]string{"items": items}**

**return 200, data, http.Header{"Content-type": {"application/json"}}**

**}**

**item:= new(Item)**

**api := new(API )**

**api.AddResource(item, "/items")**