Data type	Zero value	nil assignable? (and so also could be compared against nil)	Comparable? == (can be used as the key in the map)	Reference Type? (pass by "reference" [1])	Declaration	Initialization	%v value in fmt
1. Basic data types							
numbers (int,int8,int16,int32, int64, uint,uint8,uint16,uint32, uint64, byte, float32, float64)	0	No	Yes	No	`var x int`	`x := 10`	10
string	""	No	Yes	No	`var s string`	`s := "go"`	go
bool	false	No	Yes	No	`var b bool`	`b := true`	true
2. Composite types							
2.1. Aggregate types							

array
(fixed size)
	Zero value of all it elements (len() returns the size of the array declared). So can be used immediately after declaration without prior initialization	No	Yes
			(when array type is the comparable type)
				No	`var a [3]int`	`a := [3]int{10, 20, 30}`
						or
						`a := [...]int{10, 20, 30}`
						or
						`a := [...]{99: -1}`
						(which will create the array of 100 elements, with last one initialized to -1, while the rest set to their zero value, 0 in the case of the int-s) **[2]**
							[10 20 30]
							or
							[0 0 0] for zero value

struct
(fixed size)
	Zero value of all it fields. So can be used immediately after declaration without prior initialization	No	Yes
			(when all fields are comparable)
				No	`type point struct {`
					`  x, y int`
					`  label string`
					`  visited bool`
					`}`
					`var p point`
						`p := point{10, 20, "p", true}`
						or partially initialize only relevant fields
						`p := point{y: 20, visited: true}`
						or the special case when the struct is part of the map
						(and so the type can be omitted), i.e.:
						`m := map[string]point{ "p": {10, 20, "p", true}, }`
							{10 20 p true}
							or
							{0 0 false} for zero value

2.2. Reference types							

slice
(dynamic)
	nil
	(len() returns 0).
	Must be initialized before the use
		Yes	No
			(only legal comparison is with nil)
				Yes	`var a []int`	`a := []int{10, 20, 30}`
						or
						`a := make([]int, 0, 10)`
						If the slice is initialized with make and the length is set, i.e.
						`a := make([]int, 3, 10)`
						then len(a) will return that length, i.e. 3
							[10, 20, 30]
							or
							[] for zero value (even though the zero value is nil)
							or
							[0 0 0] when length was provided for the make

map
(dynamic)
(the key must be comparable using ==)
	nil
	(len() returns 0).
	Must be initialized before the use
		Yes	No
			(only legal comparison is with nil)
				Yes	`var m map[string]int`	`m := make(map[string]int)`
						or
						`m := map[string]int{}`
						or
						`m := map[string]int{"a": 1, "b": 2}`
							[]map
							(for empty or nil map)
							or
							map[a: 1 b: 2]

pointer	nil	Yes	Yes
			(point to the same variable or both are nil)
				Yes	`var p *int`	`p := &x`
						or
						`p := new(int)`
						(although rarely used, also in this form new() creates the new address, so those pointers are always different, i.e. == returns false)
							<nil>
							or
							0xc420074068
							(address pointer points to)

chan	nil	Yes	Yes
			(reference to the same channel data structure or nil)
				Yes	`var c chan int`	`c := make(chan int)`
						for unbuffered channel
						or
						`c := make(chan int, 5)`
						for buffered channel
							<nil>
							or
							0xc42001e180
							(address of the channel)

func	nil	Yes	No
			(only legal comparison is with nil)
				Yes **[3]**	`var f func(int) int`	`f := func(x int) int { return x + 1 }`
						for function value
						or normally declared function
						`func inc(x int) int { return x + 1 }`
						`f := inc`
							<nil>
							or
							0x48b500
							(address of the defined function)

3. Interface type							

interface{}
(abstract type)
	nil	Yes	No
			(only legal comparison is with nil)
			[4]
				No	`var any interface{}`	`any := 5`
						or
						`any := false`
						or
						`any := "string"`
						And etc. as you can assign value of any type to the interface{} type
							<nil>
							or
							the %v of the underlying type

4. Bonus							
bytes.Buffer (is actually a struct, so everything about struct applies here)	Empty buffer ready to use					`var b bytes.Buffer` A Buffer needs no initialization, ready to use	
strings.Builder (is actually a struct, so everything about struct applies here) (added in Go 1.10)	Empty builder ready to use					`var b strings.Builder` A Builder needs no initialization, ready to use	

[1] Variables of these types are passed by value, as everything in the Go, although because they are "reference" types, which mean they hold the pointer (reference) to the underlying data structure, so it means that the pointer gets copied (i.e. *alias* is created), which allows to make the modifications to the underlying data structure pointer points (references) to.

[2] Another interesting example (taken from The Go Programming Language book):

```
type Currency int
const (
    USD Currency = iota
    EUR
    GBP
    RMB)
symbol := [...]string{USD: "$", EUR: "€", "GBP": "£", "RMB": "¥"}
```

[3] It said that the func is the reference type, and mentions it is possible to affect the original function, although it is not clear how. Any ideas?

[4] Although it is possible to compare interfaces with each other, it might fail at runtime when interface type (dynamic types are the same) represents the non comparable type, then it fails with *panic*. So compare interface values only if you are certain they contain dynamic values of comparable types.