Paradigm of Go

Go is an imperative language. It has terminating statements, including “return” and “goto” statements, “if” statements, “for” statements, “switch” statements, “select” statements, and so on. It also has a statement for looping - “for” statement.

Ref:<https://golang.org/ref/spec#Statements>

Ref:<https://golang.org/ref/spec#For_statements>

Go is not exactly an object-oriented language. Although Go has types and methods and allows an object-oriented style of programming, there is no type hierarchy. Go also provides the implementation of the interface. Go allows the user to define the “struct” type as an object, which is very similar to a struct in C. However, unlike in C, Go also allows users to write methods for struct type. In Go, there is also access control at the package level defined by whether the method or attributes in a package is capitalized or not. Due to the lack of “class” features, access control is unlike many other object-oriented languages, where access control is at the class level.

Ref:<https://golang.org/doc/faq#Is_Go_an_object-oriented_language>

Go allows both explicit and implicit declaration:

|  |
| --- |
| x := 1 var y = 2 |

Go allows both explicit and implicit typed:

|  |
| --- |
| var x int = 10 y := 10 |

Go is statically typed

|  |
| --- |
| x := 1 x := "Hello World!" // Raises Error |

Go has aliases

|  |
| --- |
| var i int = 1; var p \*int; p = &i; fmt.Println(\*p) // Print 1 |

Dangling references in Go

|  |
| --- |
| var i int = 1 var p \*int p = &i p = nil // When clearing a pointer by setting it to nil, the pointed object-again-will be taken care of by the Garbage Collection fmt.Println(\*p) // Cause Error |

Memory leak in Go

Subslices may also cause kind-of memory leaking. In the following code, after the g function is called, most memory occupied by the memory block hosting the elements of s1 will be lost (if no more values reference the memory block).

|  |
| --- |
| var s0 []int  func g(s1 []int) {   // Assume the length of s1 is much larger than 30.   s0 = s1[len(s1)-30:]  } |

Ref: <https://go101.org/article/memory-leaking.html>