

# Learning Go

The differences of Go by comparing to language C and similarities to Python:

<https://blog.go-zh.org/gos-declaration-syntax>

<https://tour.golang.org>

## 1. Structure of Go program

*//Go program is made up of packages.*

*package main*

*//Use the packages with import paths. Similar to Python.*

```
import (  
    "fmt"  
    "math"  
)
```

*//Go programs start running in package main.*

```
func main() {  
    fmt.Printf("Now you have %g.", math.Sqrt(7))  
}
```

## 2. Go's Declaration Syntax

- Variable declaration inside and outside a function:

*//Use var when declaring variables, then name variables, give variable type in the end.*

*var x, y int*

- Variable declaration inside a function:

*//No var, variable name, :=, initial value. Similar to Python.*

*//Equal to var k int = 3*

*k := 3*

- Function declaration:

*//int is the result value type*

```
func add() int {  
}
```

Reason: Go's declarations read left to right.

## 3. The usage of {}

{' must be put at the end of the line with the function name, else, for, etc.

## 4. 'for' is the while loop in Go

```
func main() {  
    sum := 1
```

```

    for sum < 1000 {
        sum += sum
    }
    fmt.Println(sum)
}

```

## 5. Rules of 'for' and 'if'

```

func main() {
    sum := 0
    //No parentheses surrounding the three components of the for statement and the
    braces {} are always required.
    for i := 0; i < 10; i++ {
        sum += i
    }
    fmt.Println(sum)
}

```

## 6. No need semicolons at the end of each statements.

## 7. 'defer' in Go

A defer statement defers the execution of a function until the surrounding function returns.

## 8. 'slices' in Go

A slice does not store any data, it just describes a section of an underlying array. Changing the elements of a slice modifies the corresponding elements of its underlying array.

```

func main() {
    //primes is an array as the size is set to a value 6
    primes := [6]int{2, 3, 5, 7, 11, 13}

    //s is a slice as the size is not set to any value
    var s []int = primes[1:4]

    //primes[1] has been changed to 1
    s[0] = 1
    fmt.Println(primes)
}

```

## 9. 'range' in Go

```

var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

```

```

func main() {
    // When ranging over a slice, two values are returned for each iteration. The first

```

is the index, and the second is a copy of the element at that index.

```
    for i, v := range pow {  
        fmt.Printf("2**%d = %d\n", i, v)  
    }  
}
```

## 10. 'map' in Go

```
type Vertex struct {  
    Lat, Long float64  
}
```

*//A map maps keys to values.*

```
var m map[string]Vertex
```

```
func main() {  
    //The make function returns a map of the given type, initialized and ready for  
    use.  
    m = make(map[string]Vertex)  
    m["Bell Labs"] = Vertex{  
        40.68433, -74.39967,  
    }  
    fmt.Println(m["Bell Labs"])  
}
```

## 11. 'method' in Go

A method is a function with a special receiver argument. The receiver appears in its own argument list between the func keyword and the method name.

```
type Vertex struct {  
    X, Y float64  
}
```

*//abs() is a method with receiver v which has a type Vertex*

```
func (v Vertex) Abs() float64 {  
    return math.Sqrt(v.X*v.X + v.Y*v.Y)  
}
```

```
func main() {  
    v := Vertex{3, 4}  
    fmt.Println(v.Abs())  
}
```

## 12. 'goroutine' in Go

A goroutine is a lightweight thread managed by the Go runtime.