# Go

## Syntax

A Go file consists of the following parts:

- Package declaration
- Import packages
- Functions
- Statements and expressions

Example:

```go
package main
import "fmt"

func main() {
  fmt.Println("Hello World!")
}
```

## Comments

Go supports single-line or multi-line comments.

### Single-line Comments

```go
// This is a comment
```

### Multi-line Comments

```
/*
 * This is a Multi-line comment
 */
```

## Variables

### Types

- `int` - stores integers (whole numbers), such as 123 or -123
- `float32` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

### Declaring variables

In Go, there are two ways to declare a variable:

1. With the `var` keyword:

    ```
    var variablename type = value
    ```

    **Note**: You always have to specify either `type` or `value` (or both).

2. With the `:=` sign:

    ```
    variablename := value
    ```

    **Note**: In this case, the type of the variable is **inferred** from the value (means that the compiler decides the type of the variable, based on the value).

    **Note**: It is not possible to declare a variable using `:=`, without assigning a value to it.

## Variable Declaration With Initial Value

```go
var student1 string = "John" //type is string
var student2 = "Jane" //type is inferred
x := 2 //type is inferred
```

**Note**: The variable types of `student2` and `x` is **inferred** from their values.

## Variable Declaration Without Initial Value

```go
package main
import ("fmt")

func main() {
    var a string
    var b int
    var c bool
    fmt.Println(a)
    fmt.Println(b)
    fmt.Println(c)
}
```

By running the code, we can see that they already have the default values of their respective types:

- a is `""`
- b is `0`
- c is `false`

### Value Assignment After Declaration

```
func main() {
    var student1 string
    student1 = "John"
    fmt.Println(student1)
}
```

### Difference Between var and :=

| `var` | `:=` |
|---|---|
| Can be used **inside** and **outside** of functions | Can only be used **inside** functions |
| Variable declaration and value assignment **can be done separately** | Variable declaration and value assignment **cannot be done separately** (must be done in the same line) |

## Multiple Variable Declaration

```
var a, b, c, d int = 1, 3, 5, 7
```

**Note**: If you use the `type` keyword, it is only possible to declare **one type** of variable per line.

If the `type` keyword is not specified, you can declare different types of variables in the same line:

```
var a, b = 6, "Hello"
c, d := 7, "World!"
```

### Variable Declaration in a Block

Multiple variable declarations can also be grouped together into a block for greater readability:

```go
var (
    a int
    b int = 1
    c string = "hello"
)
```

### Variable Naming Rules

Go variable naming rules:

- A variable name must start with a letter or an underscore character (_)
- A variable name cannot start with a digit
- A variable name can only contain alpha-numeric characters and underscores (`a-z`, `A-Z`, `0-9`, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- There is no limit on the length of the variable name
- A variable name cannot contain spaces
- The variable name cannot be any Go keywords

## Constants

If a variable should have a fixed value that cannot be changed, you can use the `const` keyword.

```go
const CONSTNAME type = value
```

**Note**: The value of a constant must be assigned when you declare it.

```go
const PI = 3.14
```

### Constant Rules

- Constant names follow the same naming rules as variables
- Constant names are usually written in uppercase letters (for easy identification and differentiation from variables)

- Constants can be declared both inside and outside of a function

### Constant Types

There are two types of constants:

- Typed constants
- Untyped constants

### Multiple Constants Declaration

Multiple constants can be grouped together into a block for readability:

```go
const (
    A int = 1
    B = 3.14
    C = "Hi!"
)
```

## Output

### Output Functions

Go has three functions to output text:

- `Print()`
- `Println()`
- `Printf()`

#### The Print() Function

The `Print()` function prints its arguments with their default format.

- If we want to print the arguments in new lines, we need to use `\n`.

```go
var i,j string = "Hello","World"
fmt.Print(i, "\n")
fmt.Print(j, "\n")
```

- `Print()` inserts a space between the arguments if **neither** are strings.

### The Println() Function

The `Println()` function is similar to `Print()` with the difference that a whitespace is added between the arguments, and a newline is added at the end:

```go
var i,j string = "Hello","World"
fmt.Println(i, j)
```

### The Printf() Function

The `Printf()` function first formats its argument based on the given formatting verb and then prints them.

Here we will use two formatting verbs:

- `%v` is used to print the **value** of the arguments
- `%T` is used to print the **type** of the arguments

```go
var i string = "Hello"
var j int = 15

fmt.Printf("i has value: %v and type: %T\n", i, i)
fmt.Printf("j has value: %v and type: %T", j, j)
```

## Formatting Verbs

Go offers several formatting verbs that can be used with the `Printf()` function.

## General Formatting Verbs

The following verbs can be used with all data types:

| Verb | Description |
|---|---|
| `%v` | Prints the value in the default format |
| `%#v` | Prints the value in Go-syntax format |
| `%T` | Prints the type of the value |
| `%%` | Prints the % sign |

## Integer Formatting Verbs

The following verbs can be used with the integer data type:

| Verb | Description |
|---|---|
| `%b` | Base 2 |
| `%d` | Base 10 |
| `%+d` | Base 10 and always show sign |
| `%o` | Base 8 |
| `%O` | Base 8, with leading 0o |
| `%x` | Base 16, lowercase |
| `%X` | Base 16, uppercase |
| `%#x` | Base 16, with leading 0x |
| `%4d` | Pad with spaces (width 4, right justified) |
| `%-4d` | Pad with spaces (width 4, left justified) |
| `%04d` | Pad with zeroes (width 4) |

## String Formatting Verbs

The following verbs can be used with the string data type:

| Verb | Description |
|---|---|
| `%s` | Prints the value as plain string |
| `%q` | Prints the value as a double-quoted string |
| `%8s` | Prints the value as plain string (width 8, right justified) |
| `%-8s` | Prints the value as plain string (width 8, left justified) |
| `%x` | Prints the value as hex dump of byte values |

| Verb | Description |
|---|---|
| `% x` | Prints the value as hex dump with spaces |

### Boolean Formatting Verbs

The following verb can be used with the boolean data type:

| Verb | Description |
|---|---|
| `%t` | Value of the boolean operator in true or false format (same as using `%v`) |

### Float Formatting Verbs

The following verbs can be used with the float data type:

| Verb | Description |
|---|---|
| `%e` | Scientific notation with 'e' as exponent |
| `%f` | Decimal point, no exponent |
| `%.2f` | Default width, precision 2 |
| `%6.2f` | Width 6, precision 2 |
| `%g` | Exponent as needed, only necessary digits |

# Data Types

Data type is an important concept in programming. Data type specifies the size and type of variable values.

## Basics

Go is statically typed, meaning that once a variable type is defined, it can only store data of that type. It has three basic data types:

- **bool**: represents a boolean value and is either true or false
- **Numeric**: represents integer types, floating point values, and complex types
- **string**: represents a string value

```go
var a bool = true     // Boolean
var b int = 5         // Integer
var c float32 = 3.14  // Floating point number
var d string = "Hi!"  // String
```

## Boolean

A boolean data type is declared with the `bool` keyword and can only take the values `true` or `false`.

The default value of a boolean data type is `false`.

```go
var b1 bool = true // typed declaration with initial value
var b2 = true // untyped declaration with initial value
var b3 bool // typed declaration without initial value
b4 := true // untyped declaration with initial value
```

**Note**: Boolean values are mostly used for conditional testing.

## Integer

Integer data types are used to store a whole number without decimals, like 35, -50, or 1345000.

The integer data type has two categories:

- **Signed integers** - can store both positive and negative values
- **Unsigned integers** - can only store non-negative values

**Tip**: The default type for integer is **int**. If you do not specify a type, the type will be **int**.

### Signed Integers

Signed integers, declared with one of the `int` keywords, can store both positive and negative values:

```go
var x int = 500
var y int = -4500
```

Go has five keywords/types of signed integers:

| Type | Size | Range |
|---|---|---|
| `int` | Depends on platform: 32 bits in 32 bit systems and 64 bit in 64 bit systems | -2147483648 to 2147483647 in 32 bit systems and -9223372036854775808 to 9223372036854775807 in 64 bit systems |
| `int8` | 8 bits/1 byte | -128 to 127 |
| `int16` | 16 bits/2 byte | -32768 to 32767 |
| `int32` | 32 bits/4 byte | -2147483648 to 2147483647 |
| `int64` | 64 bits/8 byte | -9223372036854775808 to 9223372036854775807 |

### Unsigned Integers

Unsigned integers, declared with one of the `uint` keywords, can only store non-negative values:

```
var x uint = 500
var y uint = 4500
```

Go has five keywords/types of signed integers:

| Type | Size | Range |
|---|---|---|
| `uint` | Depends on platform: 32 bits in 32 bit systems and 64 bit in 64 bit systems | 0 to 4294967295 in 32 bit systems and 0 to 18446744073709551615 in 64 bit systems |
| `uint8` | 8 bits/1 byte | 0 to 255 |
| `uint16` | 16 bits/2 byte | 0 to 65535 |
| `uint32` | 32 bits/4 byte | 0 to 4294967295 |
| `uint64` | 64 bits/8 byte | 0 to 18446744073709551615 |

## Float

The float data types are used to store positive and negative numbers with a decimal point, like 35.3, -2.34, or 3597.34987.

The float data type has two keywords:

| Type | Size | Range |
|---|---|---|
| `float32` | 32 bits | -3.4e+38 to 3.4e+38 |
| `float64` | 64 bits | -1.7e+308 to +1.7e+308 |

**Tip**: The default type for float is `float64`. If you do not specify a type, the type will be `float64`.

### The float32 Keyword

```
var x float32 = 123.78
var y float32 = 3.4e+38
```

### The float64 Keyword

The `float64` data type can store a larger set of numbers than `float32`.

```
var x float64 = 1.7e+308
```

## String

The `string` data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

```
var txt1 string = "Hello!"
var txt2 string
txt3 := "World 1"
```