MAVENIR™
Trust the Future

Golang Training

24/01/2023

Golang: Identifiers

Identifiers are the user-defined name of the program components

> In Go Language Identifiers a

- variable name
- function name
- constant
- statement labels
- package name
- Types

[explain with helloworld.go]

Golang: Variables

A variable is the name of a memory location, that stores a data of a specific data type

> Each variable has
  - type
  - size
  - range
  - operations

[explain with helloworld.go]

Golang: Variable Naming convention

- Variable name can only start with a letter or an underscore
- It can be followed by any number of letters, numbers or underscores after that
- Go is case sensitive so uppercase and lowercase letters are treated differently
- The variable name cannot be any keyword name in Go
- There is no limit on the length of the variable name.

Golang: Variable declaration

> Variable can be declared using keyword ***var***

Syntax: **var <variable_name> <type>**

var <variable_name> <type> = <value>

var <name1>, <name2>,….<nameN> <type>

var <name1>, <name2>, …..,<nameN> <type> = <value1>, <value2>, …..,<valueN>

var <varialbe_name> = <value>

Golang: Scope of variable

- Local variable
  - Variables which are defined within a block or a function level
  - Only be accessed from within their block or function
- Global variable
  - Global variable are available throughout the lifetime of a program
  - A variable declared at the top of a file, outside the scope of any function or block.

Packages and Modules

- Package is a way to reuse the code
- Package is way to group code
  - Eg. math package having functions related with maths
- Every go source(.go) belongs to a package
  - Go source code will have first line as package, indicates source code in the belonging to package
  - Specified with key word package <package_name>

Golang: Keywords

- Keywords are basic constructs of program
  - Go has 25 keywords
    - Eg. Break, continue, etc...

https://github.com/mail2sada/GolangTraining/blob/main/training_material/GolangTrainingContent.xlsx

Visit keyword sheet

Golang: Data Type

- Go is a statically typed langueage
  - Every variable must have type
- Go has below built in data types
  - **Basic type:** Numbers, strings, and booleans come under this category.
  - **Aggregate type:** Array and structs come under this category.
  - **Reference type:** Pointers, slices, maps, functions, and channels come under this category.
  - **Interface type**

https://github.com/mail2sada/GolangTraining/blob/main/training_material/GolangTrainingContent.xlsx

Refer sheet Datatypes

Golang: Constant

- A constant is anything that doesn't change its value
- Go const can be of type string, numeric, boolean, and characters.

Syntax: const <const-name> <data-type> = <value>

const <const-name> = <value>

Eg. const c string = "circle"

const c = "circle"

Scope of constants is same as variable scope (local and global)

Golang: Enum

- Enums in Golang are not supported
- We can simulate Enums using iota

https://github.com/mail2sada/GolangTraining/blob/main/src/training/day1/enum.go

Golang: Operators

- Go lang supports below type of operators
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Misc Operators

Golang:Arithmatic Operators

- **Addition:**
  - ‘+’ operator adds two operands. For example, x+y.
- **Subtraction:**
  - ‘-‘ operator subtracts two operands. For example, x-y.
- **Multiplication:**
  - ‘*’ operator multiplies two operands. For example, x*y.
- **Division:**
  - ‘/’ operator divides the first operand by the second. For example, x/y.
- **Modulus:**
  - ‘%’ operator returns the remainder when the first operand is divided by the second. For example, x%y.

Golang:Relational Operators

- **==‘(Equal To)**
  - example, 5==5 will return true.
- **‘!=‘(Not Equal To)**
  - example, 5!=5 will return false.
- **‘>‘(Greater Than)**
  - example, 6>5 will return true.
- **‘<‘(Less Than)**
  - example, 6<5 will return false.
- **‘>=‘(Greater Than Equal To)**
  - example, 5>=5 will return true.
- **‘<=‘(Less Than Equal To)**
  - 5<=5 will also return true.

MAVENIR

Golang:Relational Operators

- **==‘(Equal To)**
  - example, 5==5 will return true.
- **‘!=‘(Not Equal To)**
  - example, 5!=5 will return false.
- **‘>‘(Greater Than)**
  - example, 6>5 will return true.
- **‘<‘(Less Than)**
  - example, 6<5 will return false.
- **‘>=‘(Greater Than Equal To)**
  - example, 5>=5 will return true.
- **‘<=‘(Less Than Equal To)**
  - 5<=5 will also return true.

MAVENIR

Golang:Logical Operators

- **AND(**&&) operator returns true when both the conditions in consideration are satisfied.
  - a && b returns true when both a and b are true (i.e. non-zero).
- **Logical(**'||') operator returns true when one (or both) of the conditions in consideration is satisfied.
  - a || b returns true if one of a or b is true (i.e. non-zero). Of course, it returns true when both a and b are true.
- **NOT(**'!') operator returns true the condition in consideration is not satisfied.
  - !a returns true if a is false, i.e. when a=0.

Golang:Bitwise Operators

- **bitwise AND(&):** Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
- **bitwise OR(|):** Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.
- **bitwise XOR(^):** Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
- **left shift(<<):** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
- **right shift(>>):** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
- **AND NOT(&^):** This is a bit clear operator.

MAVENIR

Golang:Assignment Operators

- “=” : is used to assign the value on the right to the variable on the left.
- **“+=”: f**irst adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.
- **“-=”:**first subtracts the current value of the variable on left from the value on the right and then assigns the result to the variable on the left.
- **“*=”:** first multiplies the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.
- **“/=”:** first divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.
- **“%=”: f**irst modulo the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.
- **“&=”:** first “Bitwise AND” the current value of the variable on the left by the value on the right and then assigns the result to the variable on the left.
- **“^=”:** first “Bitwise Exclusive OR” the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.
- **“|=”:** first “Bitwise Inclusive OR” the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.
- **“<<=”:** first “Left shift AND” the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.
- **“>>=”:** first “Right shift AND” the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.

MAVENIR

Golang: Misc Operators

- **&:** This operator returns the address of the variable.
- ***:** This operator provides pointer to a variable.
- **<-:** The name of this operator is receive. It is used to receive a value from the channel.

Golang: Scope of Variable

Local Variable

- Variables that are declared inside a function or a block are termed as Local variables.
- Not accessible outside the function or block.
- Can also be declared inside the for. inside a function.
- Can be accessed by the nested code blocks inside a function.
- Also known as block variables.
- These variables are garbage collected after the function’s execution is over.
- A variable which is declared inside a loop body will not be visible to the outside of loop body.

Golang:Scope of Variable

Global Variable

- The variables which are defined outside of a function or a block is termed as Global variables.
- These are available throughout the lifetime of a program.
- These are declared at the top of the program outside all of the functions or blocks.
- These can be accessed from any portion of the program.

Golang:Type casting

- Golang doesn't support implicit type conversion.
- Golang thows error, if variable/exception returns different type than that of the assigned variable
- Type casting can be achived as below

  Type(Variable-name)

Eg.

```
var i int
var f float64 = 10.55
i = int(f)
```

Golang:Declaring variable with :=

- **:=** is known as short variable declaration operator
- Creates the variables having a proper name and initial value
- Declare and initialize the **local variables** inside the functions
- Type of the variable is determined by the type of the expression

Syntax ***variable_name := expression or value***

  - https://github.com/mail2sada/GolangTraining/blob/main/training_material/GolangTrainingContent.xlsx
  - Refer sheet var vs SDO

MAVENIR

Golang: Decision Making statement

- Golang supports decision making with if/if-else/if-elseif and switch case statements
- **if** statement - executes some code if one condition is true
- **if...else** statement - executes some code if a condition is true and another code if that condition is false
- **if...else if....else** statement - executes different codes for more than two conditions
- The **switch...case** statement - selects one of many blocks of code to be executed

MAVENIR

Golang: Decision Making statement

if statement

> The if statement is used to execute a block of code only if the specified condition evaluates to true.

Syntax

```
if  condition {
  // code to be executed if condition is true
}
```

Golang: Decision Making statement

if else

> if....else statement allows to execute one block of code, if the specified condition is evaluates to true and another block of code if it is evaluates to false..

Syntax

```
if  condition {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}
```

Golang: Decision Making statement

if...else if....else

> if...else if...else statement allows to combine multiple if...else statements.

Syntax

```
if  condition-1 {
   // code to be executed if condition-1 is true
} else if condition-2 {
  // code to be executed if condition-2 is true
} else {
  // code to be executed if both condition1 and condition2 are false
}
```

Golang: Decision Making statement

Golang - if statement initialization

> The if statement supports a composite syntax where the tested expression is preceded by an initialization statement.

Syntax

```
if var declaration; condition {
    // code to be executed if condition is true
}
```

Golang: Decision Making statement

Golang - switch Statement

> The switch statement is used to select one of many blocks of code to be executed.

Syntax

```
switch <switch-variable> {
    case 1:
     <stmt>
    case 2:
      <stmt>
    default:
      <stmt>
}
```

Golang: loop

For loop

- Golang supports only for loop
- While loop can be achieved with for condition loop
- Do while loop can't be achieved in Golang
- break and continue holds the same significas as that of C/C++/Java

Syntax

```
 for <initialiser>;<condition>; increment {
   <stmts>
 }
for <condition> {
   <stmts>
}
```