Programming Language: Go – IDE: GoLang

1. Print the string:

A screen shot of a computer

Description automatically generated

1. Variable

* If you define a variable and do not use it, you will get an error. In fact, you cannot define a useless array
  + Declaration without assigning

A group of white text on a black background

Description automatically generated

If you declare the var without assigning and u want assign later, you have to use “=” instead of “:=”

A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

* + Declaration with assigning: you can use the short variable declaration syntax (:=) to declare and initialize variables

A black rectangular object with a white background

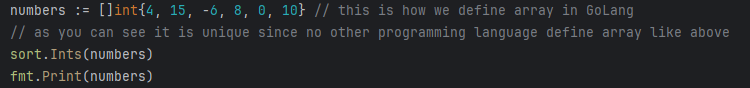
Description automatically generated

* + Other examples:

A screenshot of a computer program

Description automatically generated

* Similar to Python there is no semicolon (;) at the end of your code
* You do not need to manually enter the packages, just use the method you want and the SDK will automatically import the package for you
* In GoLang, we do not have the = operator, we have := instead
* Example



sort.Ints is the function Ints of package sort to sort Integer Number

Error if we declare a variable or an array without using it

A screen shot of a computer program

Description automatically generated

Correction

A black screen with green text

Description automatically generated

Empty array

A black screen with blue and green text

Description automatically generated

Empty array with specific length

A black background with white text

Description automatically generated

1. Struct

* GoLang does not have class, but it has struct only
* Declaration of struct

A black background with white text

Description automatically generated

* + Id and Name is a Property of struct Employee
  + It is possible to define a struct without using **type** keyword and we can also declare a struct inside another struct. The next {} bracket is initializing value

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated



* Constructor
  + In Go, there are no constructors in the traditional sense as in some other languages like Java or C++. However, you can create functions that act as constructors by convention. These functions typically return an instance of the struct with some default or initial values set.

A computer screen shot of a black screen

Description automatically generated

* + Default value with user-defined constructor

A screenshot of a computer

Description automatically generated

* Create an object of struct: (In GoLang, every variable or object has value type, only pointer has reference type)
  + Without Assigning Values (Zero Value):
    - **Declare a Variable**: You can declare a variable without initializing it, and its fields will be assigned their zero values. In this case emp has value type



A black background with colorful text

Description automatically generated

* + - **Using the new Keyword**: The new keyword allocates memory for a new zero-initialized instance of the struct and returns a pointer to it. In this case emp is a pointer



A black background with colorful text

Description automatically generated

* + With Assigning Values:
    - **Struct Literal**: You can use a struct literal to initialize the fields explicitly.

A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of text

Description automatically generated

* + - **Using the Address Operator (&) with Struct Literal**: You can create an instance using a struct literal and take its address to get a pointer to it. In this case emp is a pointer

A black background with white text

Description automatically generated

* + - **Using make()** (for composite types like slices, maps, and channels, not for structs): This is used for slices, maps, and channels, but **not** for struct types like Employee.
* Methods of Struct:
  + Not like other programming languages, methods of struct in GoLang don’t declare inside Struct, but outside it and express by receiver

A screen shot of a computer

Description automatically generated

* + As we know the parameter is inside the parathesis which is after the function name. However, receiver will be inside the parathesis which is before the function name
  + We can only have 1 receiver

A black screen with white text

Description automatically generated

* + Think the receiver as the signature that method is belong to the struct type of that receiver. This method is equivalent to this class and method in C#

A computer screen shot of code

Description automatically generated

* + - You can consider “emp” as “this”
* We also can declare a custom type for any primitive type, list, array, dictionary, tree, struct, collection of struct. This gives us a huge flexibility to define any types we want.
  + Example with primitive type:



A black background with white text

Description automatically generated



* + Example with user-defined struct:

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated

* Hard Example:

A black background with white text

Description automatically generated



A black background with blue and orange text

Description automatically generated

Because Swap() is a method of object type ByAge which represents []Person, and the method Swap affect directly to object type ByAge. Therefore, 2 of the items will swap together in the list

A black background with white text

Description automatically generated

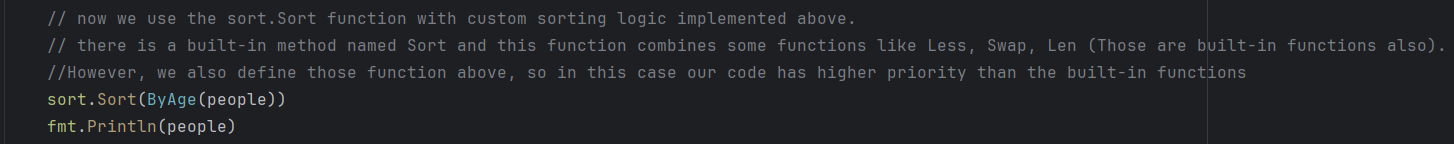
A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

* + Now we use the sort.Sort function with custom sorting logic implemented above. There is a built-in method named Sort and this function combines some functions like Less, Swap, Len (Those are built-in functions also). However, we also define those function above, so in this case our code has higher priority than the built-in functions



* + 'sort.Sort' function will pass 'ByAge(people)' instead of 'ByAge{people}'. This ensures that the people slice is converted to 'ByAge' type for sorting
* Return Type of method or function as a Pointer:
  + Here's why returning a pointer is beneficial:
    - Nil Handling: By returning a pointer, you can directly return nil when no employee is found, indicating the absence of a valid employee.
    - Avoiding Copying: When you return a pointer, you're essentially returning a reference to the original Employee struct in the slice. This avoids unnecessary copying of the struct, which can be more efficient, especially for large structs.
    - Flexibility: Returning a pointer provides flexibility if you need to modify the employee object later. Since you have the memory address of the original struct, any changes made through the pointer will directly affect the original struct.

A black background with white text

Description automatically generated

A screen shot of a computer code

Description automatically generated

* + First reason: Returning a pointer to an Employee instead of the Employee struct directly allows you to return nil when no employee is found. This is because a pointer type in Go can have a value of nil, indicating that it doesn't point to anything. If you were to return the Employee struct directly, there would be no way to differentiate between a valid employee with zero values and the absence of an employee.
    - Return the Employee struct directly will have error

A screen shot of a computer

Description automatically generated

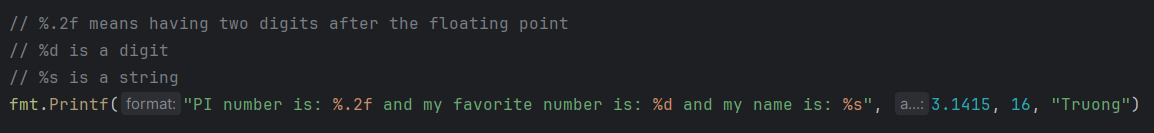
1. Print String:

* Print normal string or any variable or object

A screen shot of a computer program

Description automatically generated

* Print string with format





If we put incorrect order, it will show error

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

1. Function:

* In Go, a function is a block of code that performs a specific task. Functions are fundamental building blocks in Go programming and are defined using the func keyword. They can take zero or more input parameters and optionally return one or more values.
* Declaration:

A black screen with white text

Description automatically generated

* + **func**: This is the keyword used to declare a function.
  + **functionName**: This is the name of the function. It should follow Go's naming conventions.
  + **parameter1**, **parameter2**, ...: These are the input parameters (also called arguments) that the function expects. Each parameter is followed by its type declaration.
  + **returnType**: This is the type of value that the function returns. If the function doesn't return any value, you can omit the return type. Note: return type is added after the name of the method not before it
  + **Function** body: This is the block of code enclosed within curly braces {}. It contains the instructions that the function executes when it's called.
  + **return**: This keyword is used to return a value from the function. You can return multiple values separated by commas. Void function does not return anything. However, no need to mention "void" keyword

A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

* In Go, a function can return multiple values. This is one of the language's distinctive features. When a function returns multiple values, they are separated by commas in the return statement. Here's a basic example to illustrate this concept:

A computer screen with text on it

Description automatically generated

* You can also name the returned values in the function signature to make the return values more self-explanatory:

A computer screen shot of a black rectangle with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. If condition

A black background with white text

Description automatically generated

1. For loop:

A computer screen with white text

Description automatically generated

1. Foreach loop

* This is a pair key-value foreach loop which the first argument “\_” is a index and “num” is value of that index. Since we don’t use index so we use “\_” to ignore it

A screenshot of a computer

Description automatically generated

1. Concurrency (sự đồng thời) and Goroutines and Channels in GoLang

* A goroutine is a lightweight thread of execution in Go. It's a fundamental building block of concurrent programs in Go, allowing you to execute code concurrently and concurrently without the overhead of traditional operating system threads (Goroutine là một luồng thực thi nhẹ trong Go. Đây là một khối xây dựng cơ bản của các chương trình đồng thời trong Go, cho phép bạn thực thi mã đồng thời và đồng thời mà không cần chi phí cho các luồng hệ điều hành truyền thống)
  + Goroutine giúp chúng ta tạo ra những tiến trình chạy độc lập và cùng lúc với các tiến trình khác.
  + Keyword for goroutine is “go”
  + Why I need goroutine? In the image below, as usual, the function in main() will run from top to bottom, so the function count(“sheep”) must be done then the function count(“fish”) can be started. However, if I want they run together and parallel, I need to use goroutine

A screen shot of a computer code

Description automatically generated

Note: time.Sleep(time.Second) is the function helps to delay specific time in this case is 1 second delay

A screenshot of a computer screen

Description automatically generated

* + Therefore if I use goroutine by using keyword “go”

A computer screen shot of code

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Because the function count(“sheep”) will run parallel with function count(“fish”) instead of finish the function count(“sheep”) then start function count(“fish”). T

* + There is a question from here is that why we don’t put go for count(“fish”) as well to consider they are running parallel

A screen shot of a computer program

Description automatically generated

A black background with white text

Description automatically generated

As you can see, it finish the running without print out anything. Why? Because func main() is also a goroutine so in this case we have like 3 program running parallel which are count(“sheep”), count(“fish”) and main(). Therefore, the function main() won’t wait until those functions finish; instead, it will end the code

* + In advanced situations, I have 2 functions running parallel (in this case I will use example of count() above).

A screen shot of a computer program

Description automatically generated

So to wait until those 2 functions run parallel and end properly, I will time.Sleep(time.Second \* 10) like 10 seconds because 2 loops which iterates 5 times of each

A computer screen shot of code

Description automatically generated

This case is not optimal since we have to calculate the time stop. Therefore, to solve this problem we could use struct WaitGroup to deal with it

A screenshot of a computer

Description automatically generated

A close up of a screen

Description automatically generated

Firstly, we need to import another package named “sync”, then we create an object of struct sync.WaitGroup named “wg” without assigning it. Next, wg.Add(2) means that it will add how many goroutine that I need to wait (in this case I have 2 goroutines so I pass 2)

A screen shot of a computer code

Description automatically generated

Here I modify a little bit, I put the function count() in the anonymous function which is also a goroutine. Then I use wg.Done(), this method is used to indicate that this goroutine is done, and it will reduce the number of goroutines in wg.Add() by 1. If wg.Add() reaches to 0, it means that there is no goroutines left so it will continue the code after wg.Wait() ( in this case it must execute wg.Done() 2 times to finish the WaitGroup because we add 2 goroutines to wg.Add(2) ). wgWait() method is used to hold the running program until finish all the goroutine then after every goroutine is done, it will execute the code after wg.Wait() (in this case it will execute the fmt.Println("Truong") after all goroutine is done)

A black background with white text

Description automatically generated

* + - Some notes of WaitGroup

A screen shot of a computer

Description automatically generatedA black screen with text

Description automatically generated

* + RWMutex (RW stands for Read Write Mutex) is used to synchronize the common attributes or objects between 2 or more goroutines (in the example below the common objects between 2 goroutines that need to be sync is “counter”). With RWMutex, goroutines still run parallel but they will run in the order that users who uses RWMutex wants not randomly anymore

A screen shot of a computer code

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

As you can see the result that the “counter” variable is used in 2 goroutines and it isn’t synchronized since it keeps printing and updating in unorderly. To fix that we will use RWMutex.

* + - Step 1: firstly, we will lock the Goroutines to read any variable (in this case is “counter”) by using RLock() method (it stands for Read Lock). When it locks, no one can change the value of any variable (in this case is “counter”)

A screen shot of a computer code

Description automatically generated

* + - Step 2: After finish reading some variables, we need to unlock it to update those variables. In this case, only function sayHello() is a function to read a common variable “counter” so we should unlock it in this method. The method RUnlock() stands for Read Unlock which is used to unlock to allow updating the variable (in this case is “counter”)

A computer screen shot of a computer code

Description automatically generated

* + - Step 3: To update the variable (in this case I s “counter”) that doesn’t allow anyone to read that variable, we will use method Lock(). The method Lock() without R meaning this is Write Lock to write the “counter”

A screen shot of a computer code

Description automatically generated

* + - Step 4: After finish writing some variables, we need to unlock it to allow others to read those variables. In this case, only function increment is a function to write a common variable “counter” so we should unlock it in this method. The method Unlock() without R is Write Unlock which is used to unlock to allow reading the variable (in this case is “counter”)

A screen shot of a computer code

Description automatically generated

* + - Step 5: it will run properly and “counter” is synchronized

A screenshot of a phone

Description automatically generated

* Channels in Go are a powerful tool for communication and synchronization between goroutines (concurrently executing functions). They allow goroutines to safely send and receive data to and from each other, facilitating communication and coordination in concurrent programs.
  + Declare a channel with any type like struct, float64, int, string, … (in this case I will declare with type int) without using buffer:

A screen shot of a computer program

Description automatically generated

* + We should have two goroutine functions (could be anonymous or solid function) which are receiver and sender and they will use channel to communicate each other:
    - Receiver: the arrow will point to the local variable from the channel variable. It means that it will receive the value from the channel variable and store it to local variable “i”

A computer screen shot of text

Description automatically generated

* + - Sender: the arrow will point to the channel variable from any value that has the same type of channel variable. It means that it will send the value type int to the channel variable

A black background with white text

Description automatically generated

* + - Explanation: As we know that goroutine are running parallel (in this case are 2 anonymous functions sender and receiver). When it reaches to receiver “i := <-ch” , because there is no value in channel variable, so it has to wait, then next the sender run parallel and it store the value 42 to channel variable. Now whenever the channel variable has value, it will go directly to receiver “i := <-ch” to store the value 42 into variable “i”. Then it will continue the printing the value to the console. However, if the channel variable does not have value, the receiver “i := <-ch” has to wait until the channel variable has it
    - Another example: this is the wrong purpose of sender and receiver function, but just to illustrate the way the channel variable has to wait to receive the value to send it

A computer screen with text

Description automatically generated

A computer screen with text

Description automatically generated

As you can see here, firstly when reaching to “i := <-ch” it has to wait to because there is no value inside. Then “ch <- 42” store the value 42 into the channel variable then it will comeback to “i := <-ch” to store it into “i” and display it. Next, it send the value 27 to channel which will be displayed in the receiver “<-ch”



* + - To avoid having sending and receiving channel in the same function and to know which function is sender and receiver of channel, we will need a signature of sender and receiver

Receiver: The arrow will not point to chan type, and make sure that you will pass the channel variable “ch” at the end inside the parathesis



A screen shot of a computer

Description automatically generated

It has the error that since this is the receiver function but it include the sender so it is wrong.

A screen shot of a computer

Description automatically generated

Receiver: The arrow will point to chan type, and make sure that you will pass the channel variable “ch” at the end inside the parathesis



A screen shot of a computer program

Description automatically generated

It has the error that since this is the sender function but it include the receiver so it is wrong.

A black screen with white text

Description automatically generated

* + - To store multiple sender or receiver, we need to have a buffer for channel. To do that we will need to pass 1 more argument to a function make(). In this case, I set 50 which means that it can contain 50 objects of channel type (in this case channel type is int)



When you run it again, it won’t have any error

A screen shot of a computer program

Description automatically generated



* + - Range in channel is used to receive multiple values from sender.

A screen shot of a computer program

Description automatically generated

A black background with white text

Description automatically generated

As we can see, with range, channel variable receive maximum 50 values of channel since we declare buffer in make(). But we only send two values which are 42, so after displaying them, it will have an error because channel variable keeps receiving even though there is no sender

To fix this problem, we need to call function close() channel variable

A screen shot of a computer code

Description automatically generated



If we send value to channel after closing it, it will have error

A screen shot of a computer program

Description automatically generated



* + - Beside using range to receive multiple values from sender, we could use like this

A computer screen with white text

Description automatically generated



The <-channel will has two return values which are the value received from the channel (“i” in this case) and a Boolean value (ok in this case) indicating whether the channel is open (true) or closed (false)

* + - If we have multiple channel variable using the same receiver and sender

A screen shot of a computer program

Description automatically generated

* + Some error of channel:
    - Having more sender than receiver and vice versa because it has no buffer to store multiple sender or receiver. You can see solution of using buffer above

A screen shot of a computer program

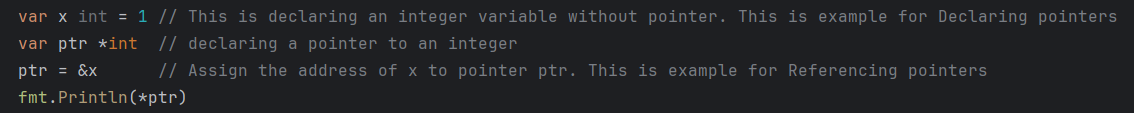
Description automatically generated

A black background with white text

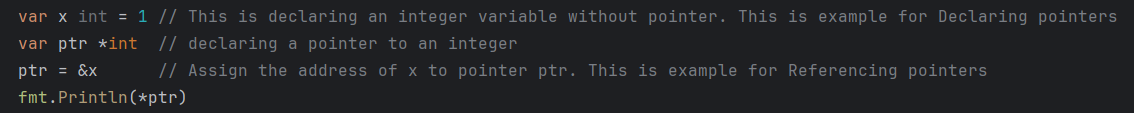
Description automatically generated

1. Pointer

* In golang, a pointer is a variable that stored a memory address of another variable. It points to the memory location where the actual value is stored. Pointers allow us to indirectly access and modify the value of a variable by referring to its memory address. Here are some key concepts related to pointers:
  + Declaring pointer:
    - To declare a pointer, we use \* symbol followed by the type of the variable it will point to



* + Referencing pointers:
    - The & operator is used to get the memory address of a variable
    - Quick note:
      * “\*” operator for getting or setting the value of a pointer
      * “&” for getting or setting the address of a pointer



* + Dereferencing (getting the value):
    - The \* operator is used to dereference a pointer which means obtaining the value stored at the memory address pointed to by the pointer

A black screen with white text

Description automatically generated

* + Null pointers:
    - GoLang has a nil value (the same as null in Java) that represents a null pointer.
    - A pointer is nil if it does not point to any memory address

A screen shot of a computer code

Description automatically generated

* + new() Function:
    - The new function allocates memory for a variable and returns a pointer to that memory.
    - The allocated memory is initialized to zero
    - This is the way that creating pointer and initializing the value to it

A black background with white text

Description automatically generated

* + Pointers as function parameters:
    - Pointers are often used as parameters to functions to allow the function to modify the original value

A black background with blue text

Description automatically generated

A screenshot of a computer code

Description automatically generated

A black background with white numbers

Description automatically generated

1. Input from console

A screen shot of a computer program

Description automatically generated

A black background with white text

Description automatically generated

1. Check Prime or not

A computer screen with text and numbers

Description automatically generated

Or use built-in package

A screenshot of a computer program

Description automatically generated

A computer screen with text

Description automatically generated