Intelligent programming

Lecture one

Topics

- Intelligent programming with Go language
- Concurrency in Go language
- Knowledge based systems
- Genetic algorithms,
- Fuzzy Logic
- Semantic web
- RDF
- Introduction to Lisp language

Introduction to Go language

Go is an open-source programming language developed by Google.

It is a statically-typed compiled language.

This language support **concurrent programming** and also allows running multiple **processes simultaneously**.

Go allows developers to deal with pointers.

Go has **garbage collection**, which itself does the memory management and allows the deferred execution of functions.

Interpreter vs. Compiler

Interpreter	Compiler
Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.

A program such as Go,C, C++ or Java needs to be compiled before it is run and the machine code is stored in a **storage device** and **loaded to memory** when required to be executed.

In contrast, Python has no compilation step. Instead, an interpreter reads over the code and translat line by line to machine code.

Using compilers, the program codes are translated into machine code already and hence the time to execute the code is very less but it is not possible to change the program without going back to the source code.

Interpreters make working with the source code much easier.

Go language applications

- AI applications that needs real-time constrains.
- High efficiency in using memory
  (suitable for large size of data)
- Go language is the ideal platform to write concurrency-intensive applications, and networking applications.
- Go language is perfect when it comes to cloud compatibility

GoLearn

- GoLearn support implementation of many machine learning algorithms.
  - DBSCAN (Density Based Spatial Cluster Analysis),
  - Random forest (RF),
  - k-nearest neighbors (KNN),
  - Naive Bayes (NB),
  - Neural network (NN),

Gorgonia

- **Gorgonia:** is a low level library has an efficient ability to deal with multi-dimensional arrays.
- It has several usage

Structure of Go program

A Go program basically consists of the following parts –

- Package Declaration
- Import Packages
- Functions
- Variables
- Statements and Expressions
- Comments

Structure of Go program

- Go programs can be written in any text editor
- Declare a main package (a package is a way to group functions, and it's made up of all the files in the same directory). Main package is the entry point to execute your program.
- Import required packages such as fmt package, which contains functions for formatting text, including printing to the console.
- Implement a main function to do the required functionality.
- A main function executes by default when you run the main package.
- Run your code to see the greeting.
- $ go run .

Executing your Go program

- After installing Go

- Open a command prompt and cd to your home directory.

- Create a directory for your Go source code.
- For example, use the following commands:
  - mkdir app1
  - cd app1

- To enable tracking your code create a go.mod file, run the go mod init command, give it the name of the module your code will be in.
  - $ go mod init example/app1
  - go: creating new go.mod: module example/app1

- Run your code to see the output.
- go run .

Hello world example

```
package main

import "fmt"

func main() {
   /* This is my first sample program. */
   fmt.Println("Hello, World!")
}
```

Basic syntax (Line Separator)

- In a Go program, the line separator key is a statement terminator.
- That is, individual statements don't need a special separator like “;” in C.
- The Go compiler internally places “;” as the statement terminator to indicate the end of one logical entity.

Go Identifiers

A Go identifier is a name used to identify a variable, function, or any other user-defined item.

- An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

- identifier = letter { letter | unicode_digit }.

Data Types

- 1. Boolean types

Boolean type consists of the two predefined constants: (a) true (b) false

- 2. Numeric types

They are arithmetic types and they represents a) integer types or b) floating point values.

- 3. String types

A string type represents the set of string values. Its value is a sequence of bytes.

Strings are immutable types that is once created, it is not possible to change the contents of a string.

- 4.Derived types

They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types f) Slice types g) Interface types h) Map types i) Channel Types

Variable Definition in Go

A variable definition tells the compiler where and how much storage to create for the variable.

- A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

- var variable_list optional_data_type
- var i, j, k int;
- var c, ch byte;
- var f, salary float32;

Variable initialization

- Variables can be initialized (assigned an initial value) in their declaration.
- The type of variable is automatically judged by the compiler based on the value passed to it.
- The initializer consists of an equal sign followed by a constant expression
- var d = 3 // declaration of d Here d is int

Dynamic Type Declaration
Type Inference in Go

- A dynamic type variable declaration requires the compiler to interpret the type of the variable based on the value passed to it.

- y:=20

Printing type of variable

```
package main
import "fmt“
func main() {
var x float64
x = 20.0
fmt.Println(x)
fmt.Printf("x is of type %T\n", x)
}
```

Println( )

- Println () function formats the string using the default formats for its operands. It add spaces and line breaker automatically

```
package main

import "fmt"

func main() {

    s := "Sam"
   age := 25
   fmt.Println("His name is", s)

   fmt.Println("His age is", age, "years")
}
```

```
His name is Sam
His age is 25 yearsh
```

Printf ()

- Means “Print Formatter”, it prints formatted strings.
-
- It does not insert a new line at the end like Println ().
- For that, you’ll have to add “\n” in the end.
- It formats the string according to a format specifier.
- %s : string
- %d : decimal
- %f :floating point
- %T : type

- package main
- import "fmt“
- func main() {
- var x float64 = 20.0
- y := 42
- fmt.Println(x)
- fmt.Println(y)
- fmt.Printf(" %f\n", x)
- fmt.Printf("%d\n", y)
- fmt.Printf("x is of type %T\n", x)
- fmt.Printf("y is of type %T\n", y)
- }

Python - Variable Types

- Variables are nothing but reserved memory locations to store values.

- This means that when you create a variable you reserve some space in memory.

- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

Pointers in Go language

- Go programming tasks are performed more easily with pointers, and other tasks, such as call by reference, cannot be performed without using pointers.
-
- Every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

Pointers in Go language

```
package main
import "fmt"
func main() {
   var a int = 10
   fmt.Printf("Address of a variable: %x\n", &a  )
}
```

```
Address of a variable: 10328000
```

Pointers for integers

var x int

If px is a pointer to x

Its value will be 1000
Px=1000

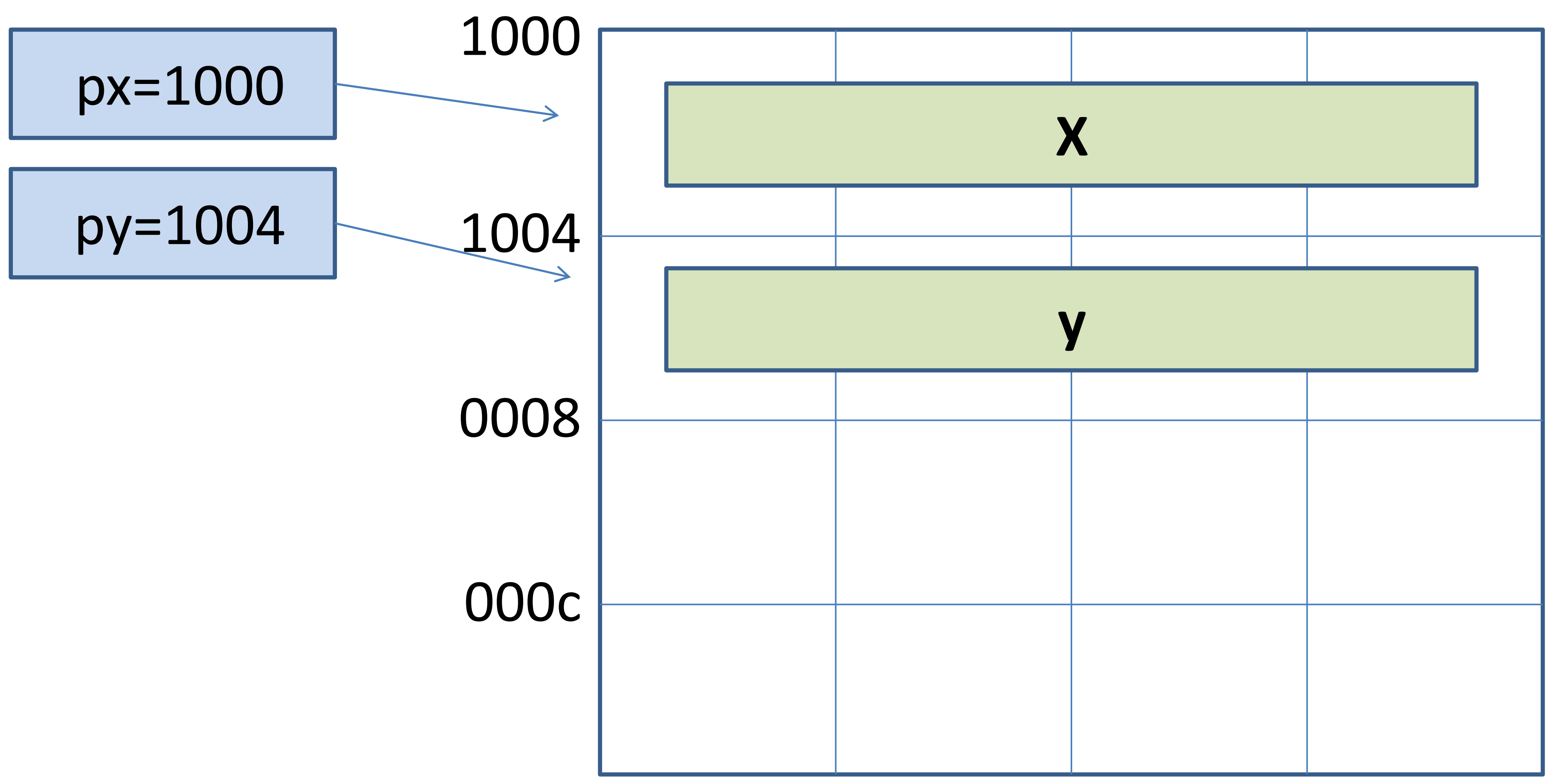
var y int

If py is a pointer to x

Its value will be 1004

Px=1004

Memory



Pointers in GO

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.

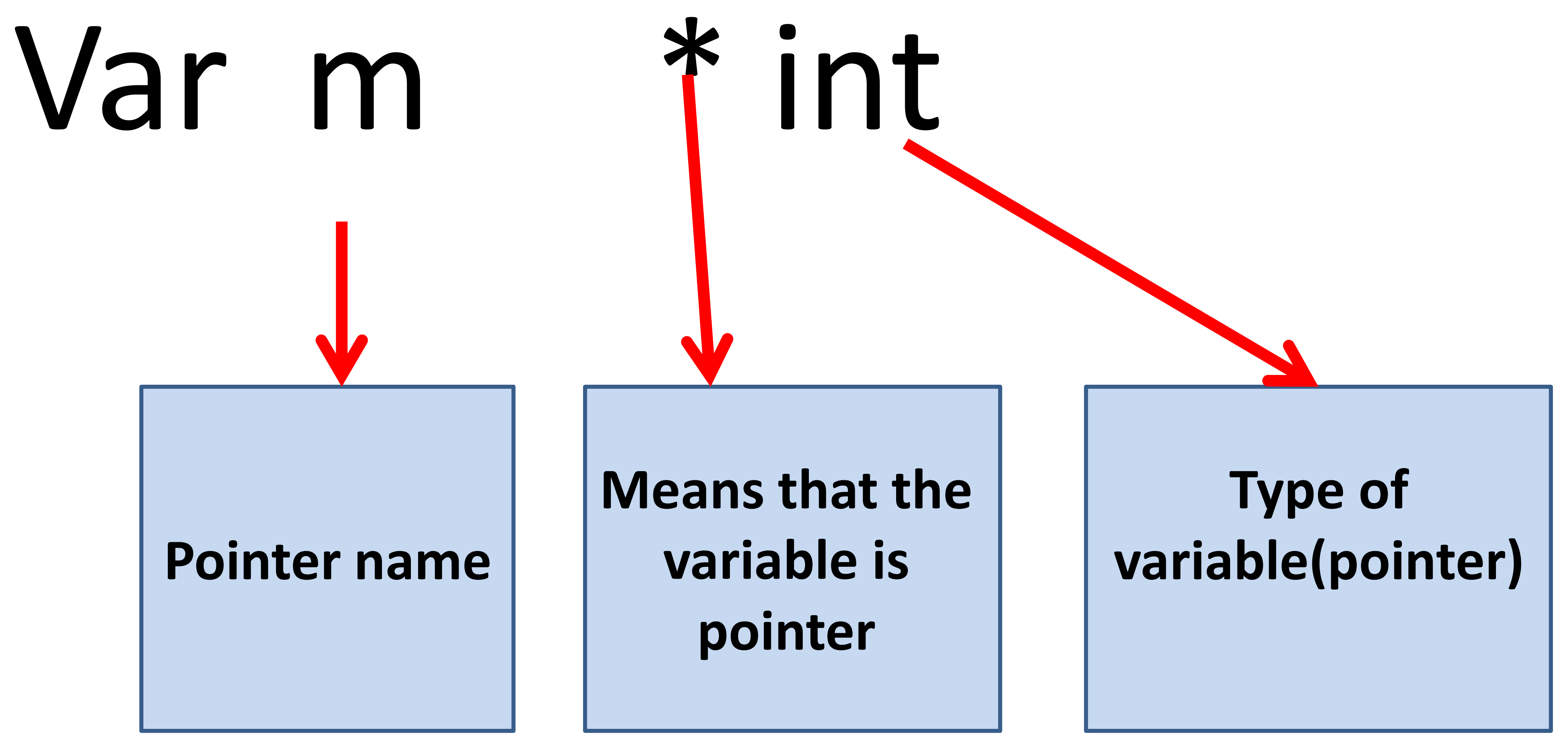
- Like any variable, you must declare a pointer before you can use it to store any variable address.

- The general form of a pointer variable declaration is –
- var var_name *var-type

Pointers declaration

Declaration

var pointer name   *type of variable;



Pointers Declaration

***int** x ; (x is a pointer to an integer)

***float** z; (z is a pointer to a float)

Here, **type** is the pointer's base type; it must be a valid Go data type

var-name is the name of the pointer variable.

The **asterisk *** used to declare a pointer.

- var ip *int   /* pointer to an integer */
- var fp *float32   /* pointer to a float */