Intelligent programming

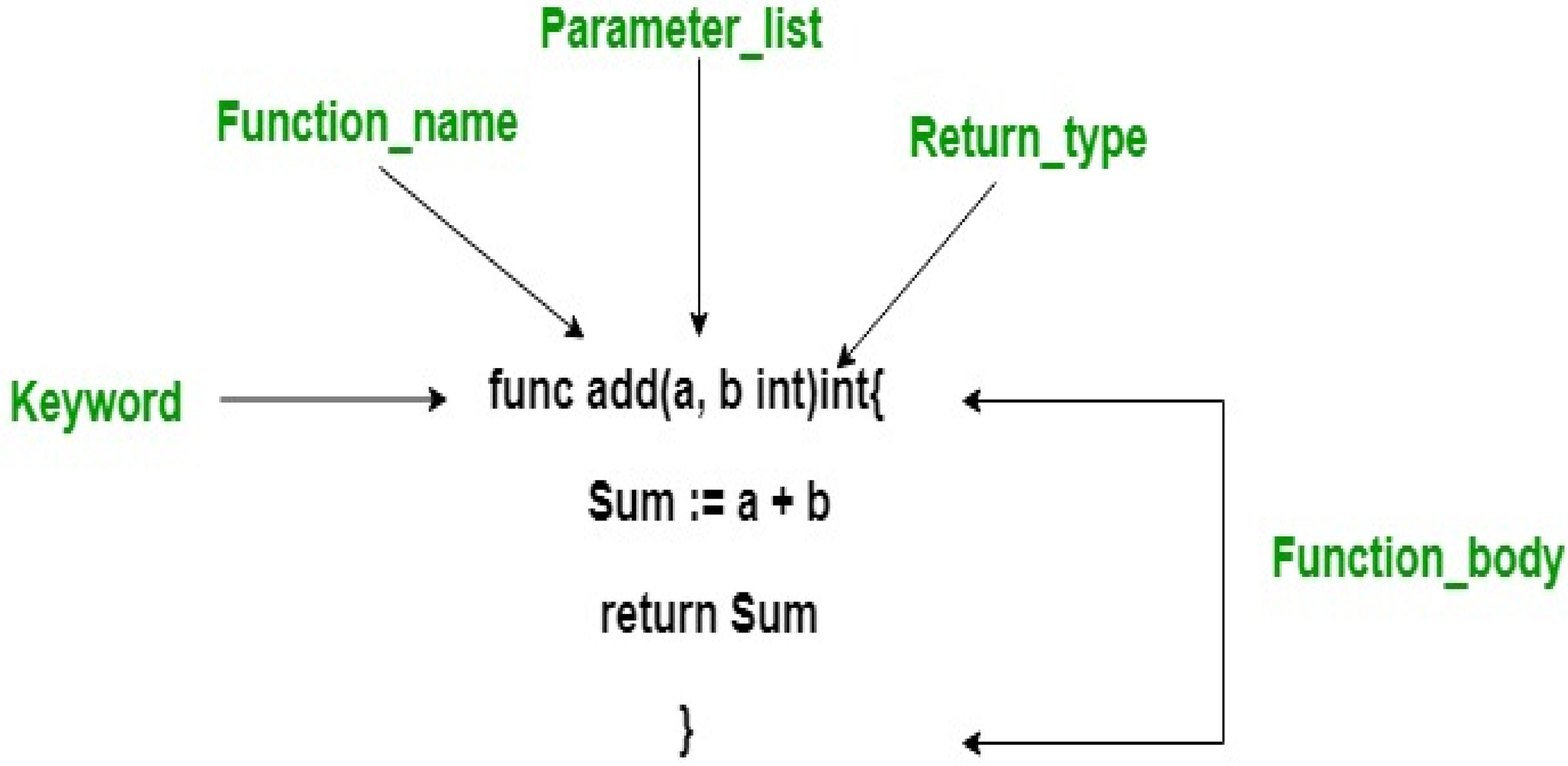
Lecture Three

- Functions
- Pointers in functions (call by reference)
- Structure
- Pointer to structure

Function Declaration

Syntax:

```
func  function_name  (Parameter-list)  (Return_type) {
// function body.....
}
```


Parameter_list
Function_name
Return_type
Keyword
func add(a, b int)int{
Sum := a + b
return Sum
}
Function_body

Function to calculate area of rectangle

```
package main
import "fmt"

func  area (length, width int) int{
    Ar := length* width
    return Ar
}

func main() {

fmt.Printf("Area of rectangle is : %d", area(12, 10))
}
```

Function Arguments

- In Go language, the parameters passed to a function are called actual parameters whereas the parameters received by a function are called formal parameters.

- Note: By default Go language use call by value method to pass arguments in function.

- Call by value: : In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations.

- So any changes made inside functions are not reflected in actual parameters of the caller.

Call by value

```
package main
import "fmt"
func swap(a, b int ) int {
    var o int
    o= a
    a=b
    b=o
   return o
}
func main( ) {
 var p int = 10
 var q int = 20
  fmt.Printf("p = %d and q = %d", p, q)
 swap(p, q)
   fmt.Printf("\np = %d and q = %d",p, q)
}
```

Output

p = 10 and q = 20

p = 10 and q = 20

Formal parameters

Actual parameters

Call by reference

Both the actual and formal parameters refer to the same locations.

Any changes made inside the function are actually reflected in actual parameters of the caller.

Call by value and Call by reference

Call by value

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Call by reference

- This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Call by reference

```
package main
import "fmt"
func swap(a, b *int) int{
   var o int
   o = *a
   *a = *b
   *b = o
  return o
}
func main() {
 var p int = 10
 var q int = 20
 fmt.Printf("p = %d and q = %d", p, q)
 swap(&p, &q)
  fmt.Printf("\np = %d and q = %d",p, q)
}
```

```
p = 10 and q = 20
p = 20 and q = 10
```

Call by reference

Variadic Functions

- The function that is called with the varying number of arguments is known as variadic function.
- A user is allowed to pass zero or more arguments in the variadic function.

Variadic Functions

```
function  function_name  (para1, para2...type) type {
// code...
}
```

Only the last parameter of a function can be variadic.

Example

```
func  hello (a int, b ...int)  {
 // code...
}
```

the parameter b is variadic since it's prefixed by ellipsis and it can accept any number of arguments.

This function can be called by using the syntax.

```
hello(1, 2)                 //passing one argument "2" to b

hello(5, 6, 7, 8, 9)   //passing arguments "6, 7, 8 and 9" to b
```

It is also possible to pass zero arguments to a variadic function.

```
hello(1) // passing 1 to a and no parameter to b
```

Variadic Functions

```
package main

import (
    "fmt"
    "strings"
)

func  joinstr  (elements ...string)  string {
   return  strings.Join  (elements, "-")
}
func main() {
    fmt.Println (joinstr( ))
    fmt.Println (joinstr(“A", “B"))
    fmt.Println (joinstr(“A", “and", “B"))
    fmt.Println (joinstr(“A", “and", “B", “and", “c"))

}
```

```
A-B
A-and-B
A-and-B-and-C
```

Anonymous function in Go Language

- Go language provides a special feature known as an anonymous function.

- An anonymous function is a function which doesn't contain any name.

- An anonymous function is also known as ***function literal***.

Anonymous function

```
package main

import "fmt"

func main() {

     func(){

    fmt.Println(“FCDS")
  }

(  )

}
```

```
Output

FCDS
```

Structure in Go

- A structure or struct in Golang is a user-defined type that allows to group/combine items of possibly different types into a single type.

- Any real-world entity which has some set of properties/fields can be represented as a struct.

- This concept is generally compared with the classes in object-oriented programming.

- It can be termed as a lightweight class that does not support inheritance but supports composition.

type **struct_variable_type** struct {

member definition;

member definition;

…

member definition;

}

Structure in Go

First define the type

```
type  Address  struct {
      name string
      street string
      city string
      country string
      Pincode int
}
```

type Address struct { name, street, city, state string Pincode int }

To Define a structure:

var a Address

- initialize a variable of a struct type using a struct

- var a = Address{“Ahmed", “Foad", "Alexandria", “Egypt", 252636}

- Once a structure type is defined, it can be used to declare variables of that type using the following syntax.

- variable_name := structure_variable_type {value1, value2...valuen}
- a := Address{“Ahmed", “Foad", "Alexandria", “Egypt", 252636}

- Similar to
- y:=20

```
package main
 import "fmt"
type Books struct {
   title string
   author string
   subject string
   book_id int
}
func main() {
   var Book1 Books
   var Book2 Books
  Book1.title = “Intelligent  Programming"
   Book1.author = " Kumar"
   Book1.subject = "Go Programming "
   Book1.book_id = 634507
  Book2.title = "Telecom Billing"
   Book2.author = "Zara "
   Book2.subject = "Telecom Billing Tutorial"
   Book2.book_id = 6495700
```

```
fmt.Printf( "Book 1 title : %s\n", Book1.title)
   fmt.Printf( "Book 1 author : %s\n", Book1.author)
   fmt.Printf( "Book 1 subject : %s\n", Book1.subject)
   fmt.Printf( "Book 1 book_id : %d\n", Book1.book_id)

   fmt.Printf( "Book 2 title : %s\n", Book2.title)
   fmt.Printf( "Book 2 author : %s\n", Book2.author)
   fmt.Printf( "Book 2 subject : %s\n", Book2.subject)
   fmt.Printf( "Book 2 book_id : %d\n", Book2.book_id)
}
```

Structures as Function Arguments

- You can pass a structure as a function argument in very similar way as you pass any other variable or pointer.

```
type Books struct {
   title string
   author string
   subject string
   book_id int
}
func main() {
   var Book1 Books
   var Book2 Books
    Book1.title = “Intelligent Programming"
   Book1.author = " Kumar"
   Book1.subject = "Go Programming "
   Book1.book_id = 6452307
   Book2.title = "Telecom Billing"
   Book2.author = "Zara Ali"
   Book2.subject = "Telecom Billing Tutorial"
   Book2.book_id = 6495700
   printBook(Book1)
    printBook(Book2)
}
```

```
func printBook( book Books ) {

   fmt.Printf( "Book title : %s\n", book.title);

   fmt.Printf( "Book author : %s\n", book.author);

   fmt.Printf( "Book subject : %s\n", book.subject);

   fmt.Printf( "Book book_id : %d\n", book.book_id);

}
```

- Define a structure of a product including a name, id, manufacture year, then input three products and define a function to print data of the product and use function to print products data

Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable

var   struct_pointer *Books

Now, you can store the address of a structure variable in the above defined pointer variable.
To find the address of a structure variable, place the & operator before the structure's name as follows

struct_pointer = &Book1;

To access an member (titel)
struct_pointer.title;

```
type Books struct {
   title string
   author string
   subject string
   book_id int
}
func main() {
   var Book1 Books
    var Book2 Books
 Book1.title = “Intelligent Programming"
   Book1.author = "Kumar"
   Book1.subject = "Go Programming "
   Book1.book_id = 6495407
  Book2.title = "Telecom Billing"
   Book2.author = "Zara Ali"
   Book2.subject = "Telecom Billing Tutorial"
   Book2.book_id = 6495700
     printBook(&Book1)
      printBook(&Book2)
}
```

```
func printBook( book *Books ) {

   fmt.Printf( "Book title : %s\n", book.title);

   fmt.Printf( "Book author : %s\n", book.author);

   fmt.Printf( "Book subject : %s\n", book.subject);

   fmt.Printf( "Book book_id : %d\n", book.book_id);

}
```