PES
UNIVERSITY
CELEBRATING 50 YEARS

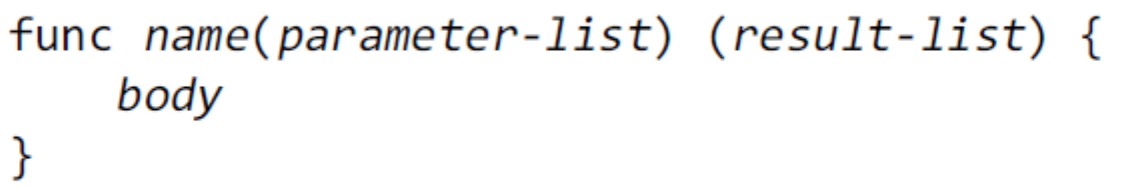
Go Programming Language

Functions and Methods

Department of Computer Science and Engineering

Go Programming Language
Functions

- A function lets us wrap up a sequence of statements as a unit that can be called from elsewhere in a program, perhaps multiple times.
- Functions make it possible to break a big job into smaller pieces that might well be written by different people separated by both time and space.
- A function hides its implementation details from its users.
- A function declaration has a name, a list of parameters, an optional list of results, and a body:



```
func name(parameter-list) (result-list) {
    body
}
```

- The unction’s parameters, which are the local variables whose values or arguments are supplied by the caller.
- The result list specifies the types of the values that the function returns

Go Programming Language

Functions

- If the function returns one unnamed result or no results at all, parentheses are optional and usually omitted.
- Leaving off the result list entirely declares a function that does not return any value and is called only for its effects.

Example:

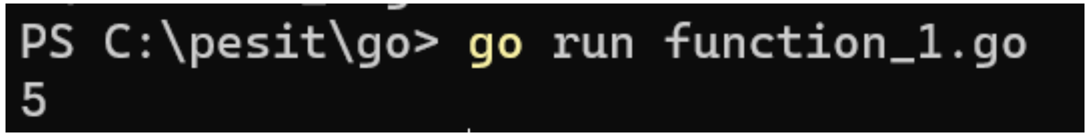
```go
package main

import ("fmt"
        "math"
       )

func main() {

              fmt.Println(hypot(3, 4))
}


func hypot(x, y float64) float64 {
                                   return math.Sqrt(x*x + y*y)
}
```



```
PS C:\pesit\go> go run function_1.go
5
```

Go Programming Language

Functions

- Like parameters, results may be named. In that case, each name declares a local variable initialized to the zero value for its type.
- A function that has a result list must end with a return statement
- The blank identifier can be used to emphasize that a parameter is unused

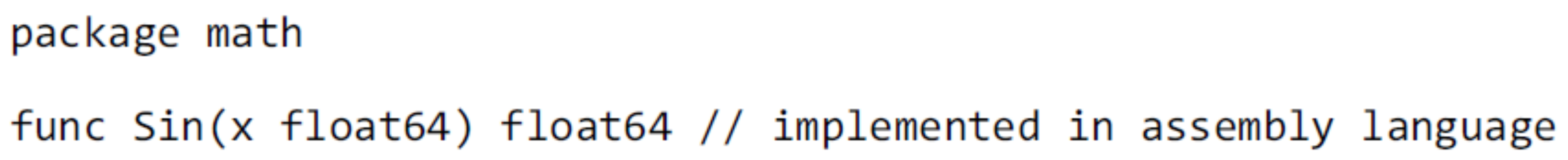
```
func first(x int, _ int) int { return x }
```

- Every function call must provide an argument for each parameter, in the order in which the parameters were declared.
- Go has no concept of default parameter values, nor any way to specify arguments by name
- Parameters are local variables within the body of the function, with their initial values set to the arguments supplied by the caller.
- Go supports recursion

Go Programming Language

Functions

- Arguments are passed by value, so the function receives a copy of each argument; modifications to the copy do not affect the caller.
- If the argument contains some kind of reference, like a pointer, slice, map, function, or channel, then the caller may be affected by any modifications the function makes to variables indirectly referred to by the argument
- If a function is declared without a body, it indicates that the function is implemented in a language other than Go.



```go
package math

func Sin(x float64) float64 // implemented in assembly language
```

Go Programming Language
Functions – return values

- A function can return more than one result.
- A bare return is a shorthand way to return each of the named result variables in order
- Bare returns are best used sparingly.

Go Programming Language

Anonymous Functions

- Named functions can be declared only at the package level, but we can use a function literal to denote a function value within any expression.
- A function literal is written like a function declaration, but without a name following the func keyword.
- It is an expression, and its value is called an anonymous function.
- Function literals let us define a function at its point of use
- The anonymous inner function can access and update the local variables of the enclosing function

Go Programming Language

Variadic Functions

- Variadic function is one that can be called with varying numbers of arguments.
- The most familiar examples are **fmt.Printf** and its variants.
- To declare a variadic function, the type of the final parameter is preceded by an ellipsis, ‘‘...’’, which indicates that the function may be called with any number of arguments of this type.

```go
func sum(vals ...int) int {
    total := 0
    for _, val := range vals {
        total += val
    }
    return total
}
```

Go Programming Language

Deferred Function calls

- Syntactically, a defer statement is an ordinary function call prefixed by the keyword **defer**.
- The function and argument expressions are evaluated when the statement is executed, but the actual call is deferred until the function that contains the defer statement has finished, whether normally, by executing a return statement or abnormally, by panicking.
- Any number of calls may be deferred;
- They are executed in the reverse of the order in which they were deferred.
- A defer statement is often used with paired operations like open and close, connect and disconnect, or lock and unlock to ensure that resources are released in all cases, no matter how complex the control flow.
- The right place for a defer statement that releases a resource is immediately after the resource has been successfully acquired.

Go Programming Language
Built-in function - panic

- Go’s type system catches many mistakes at compile time, but others, like an out-of-bounds array access or nil pointer dereference, require checks at run time.
- When the Go runtime detects these mistakes, it panics.
- During a typical panic, normal execution stops, all deferred function calls in that go routine are executed, and the program crashes with a log message.
- This log message includes the panic value, which is usually an error message of some sort, and, for each goroutine, a stack trace showing the stack of function calls that were active at the time of the panic.
- This log message often has enough information to diagnose the root cause of the problem without running the program again, so it should always be included in a bug report about a panicking program.

Go Programming Language

Built-in function - recover

- In response to a panic, it might be possible to recover in some way, or at least clean up the mess before quitting
- If the built-in recover function is called within a deferred function and the function containing the defer statement is panicking, recover ends the current state of panic and returns the panic value.
- The function that was panicking does not continue where it left off but returns normally.
- If recover is called at any other time, it has no effect and returns nil
- Recovering indiscriminately from panics is a dubious practice because the state of a package's variables after a panic is rarely well defined or documented
- Recovering from a panic within the same package can help simplify the handling of complex or unexpected errors, but as a general rule, you should not attempt to recover from another package's panic.

Go Programming Language

Object Oriented Programming - Methods

- Since the early 1990s, object-oriented programming (OOP) has been the dominant programming paradigm in industry
- An object-oriented program is one that uses methods to express the properties and operations of each data structure so that clients need not access the object’s representation directly.
- An object is simply a value or variable that has methods, and a method is a function associated with a particular type

Go Programming Language
Method declaration

- A method is declared with a variant of the ordinary function declaration in which an extra parameter appears before the function name.
- The parameter attaches the function to the type of that parameter.

```
type Point struct{ X, Y float64 }

func (p Point) Distance(q Point) float64 {
        return math.Hypot(q.X - p.X, q.Y - p.Y)
}
```

- The extra parameter p is called the method’s receiver
- In Go, we don’t use a special name like **this** or **self** for the receiver
- In a method call, the receiver argument appears before the method name

PES UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering