PES UNIVERSITY
CELEBRATING 50 YEARS

Go Programming Language

Introduction

Department of Computer Science and Engineering

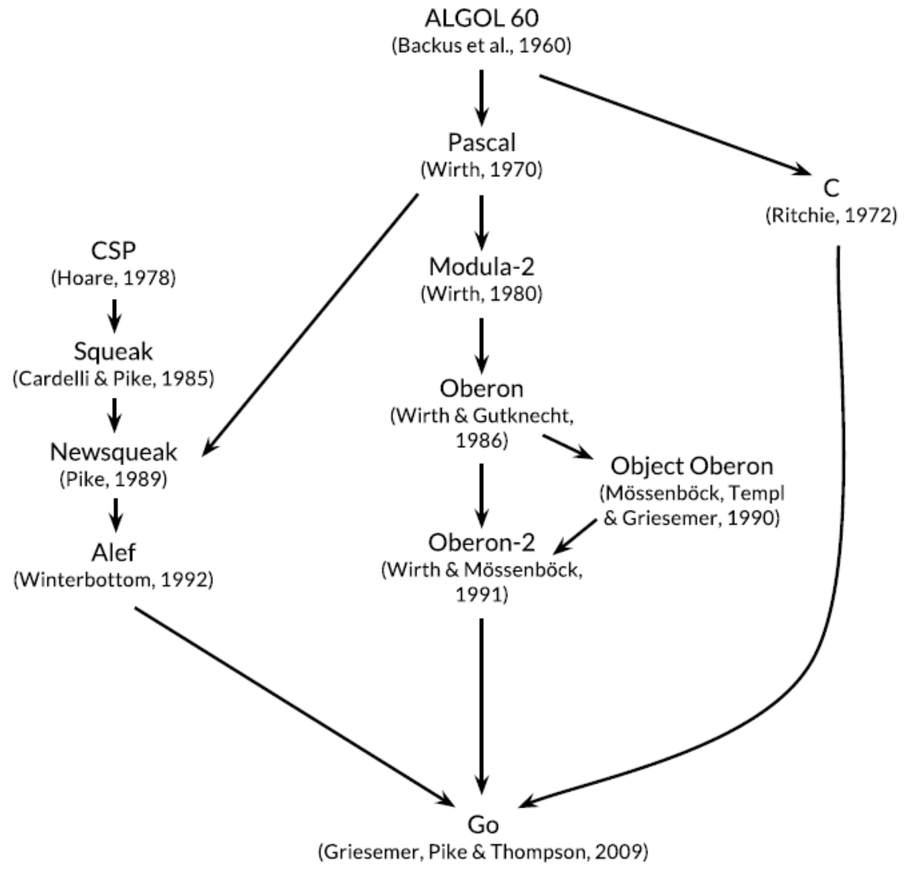
Go Programming Language

Introduction

- Go is an open source programming language that makes it easy to build simple, reliable, and efficient software
- Go was conceived in September 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, while working at Google, and was announced in November 2009.
- Go is especially well suited for building infrastructure like networked servers, and tools and systems for programmers
- It has become popular as a replacement for untyped scripting languages because it balances expressiveness with safety.
- Untyped languages, also known as dynamically typed languages, are programming languages that do not make you define the type of a variable

Go Programming Language
Genesis

- Go inherited its expression syntax, control-flow statements, basic data types, call-by-value parameter passing, pointers, from C.
- In Communicating Sequential Processing, a program is a parallel composition of processes that have no shared state; the processes communicate and synchronize using channels.
- Go inherited the fundamental concepts of concurrency from CSP.
- Oberon-2 influenced the syntax for packages, imports, and declarations, and Object Oberon provided the syntax for method declarations

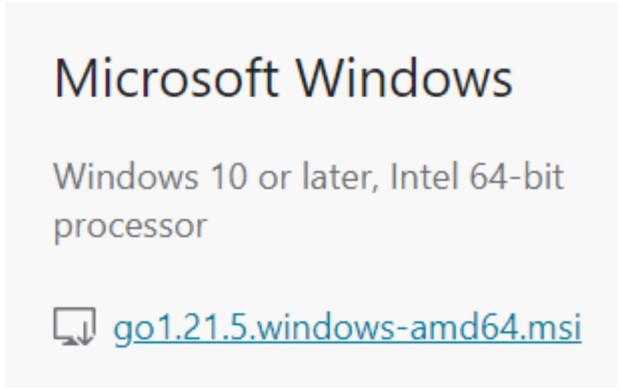


Go Programming Language
Introduction

- Go has no implicit numeric conversions, no constructors or destructors, no operator overloading, no default parameter values, no inheritance, no generics, no exceptions, no macros, no function annotations, and no thread-local storage
- Go’s built-in data types and most library data structures are crafted to work naturally without explicit initialization or implicit constructors, so relatively few memory allocations and memory writes are hidden in the code.
- Go’s aggregate types (structs and arrays) hold their elements directly, requiring less storage and fewer allocations and pointer indirections than languages that use indirect fields
- Go’s standard library, provides clean building blocks and APIs for I/O, text processing, graphics, cryptography, networking, and distributed applications, with support for many standard file formats and protocols.

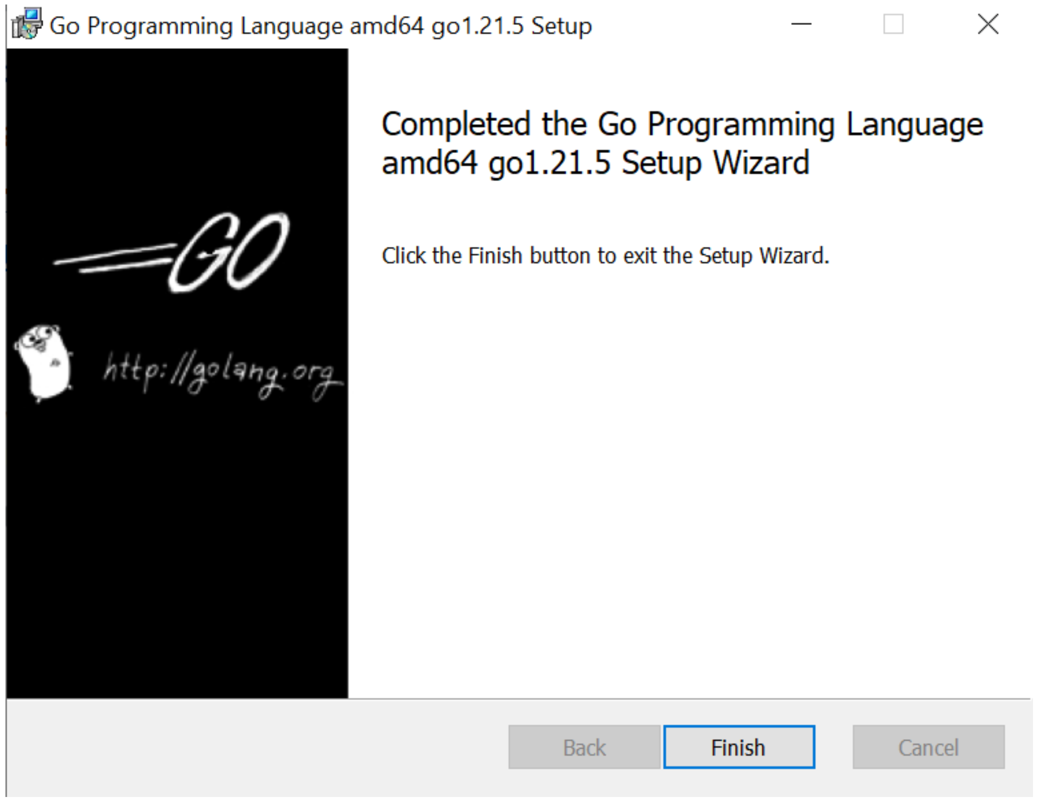
Go Programming Language
Installation

- https://go.dev/doc/install


Microsoft Windows

Windows 10 or later, Intel 64-bit processor

go1.21.5.windows-amd64.msi

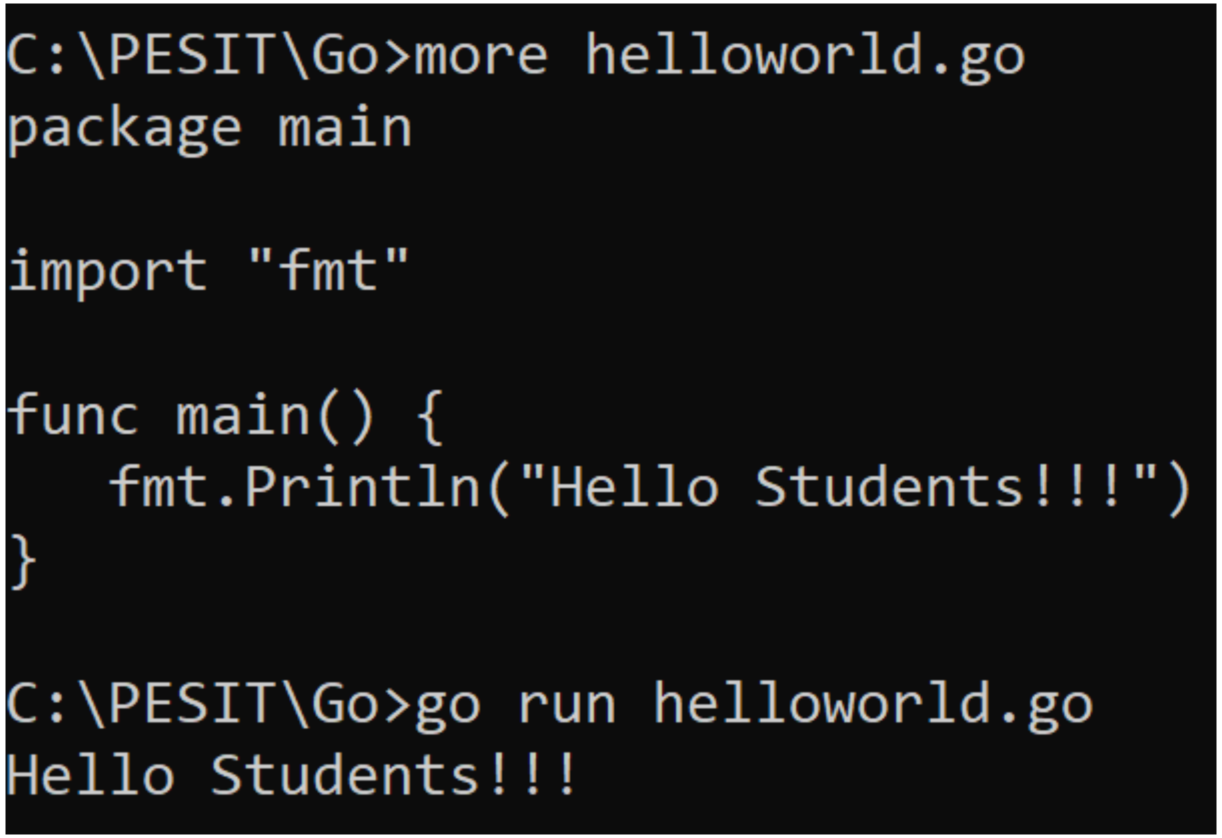

Go Programming Language amd64 go1.21.5 Setup

Completed the Go Programming Language amd64 go1.21.5 Setup Wizard

Click the Finish button to exit the Setup Wizard.

Go Programming Language

First program

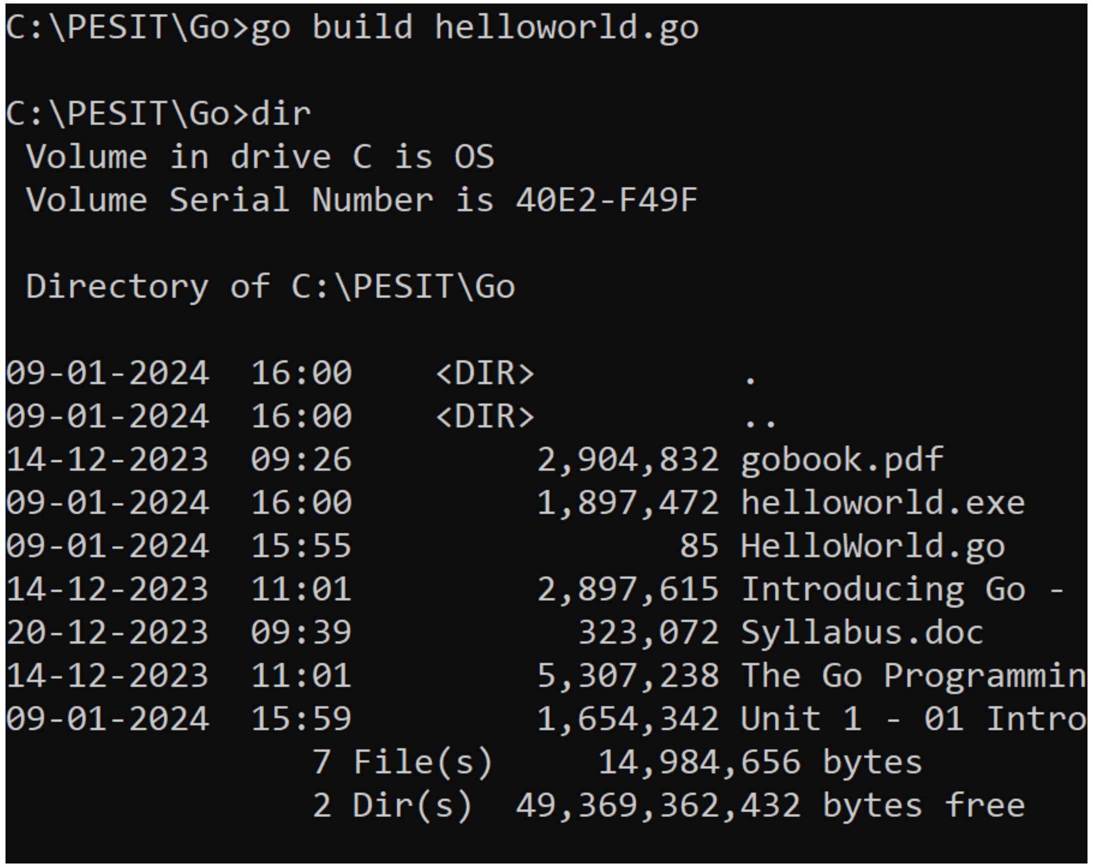


```
C:\PESIT\Go>more helloworld.go
package main

import "fmt"

func main() {
   fmt.Println("Hello Students!!!")
}

C:\PESIT\Go>go run helloworld.go
Hello Students!!!
```



```
C:\PESIT\Go>go build helloworld.go

C:\PESIT\Go>dir
 Volume in drive C is OS
 Volume Serial Number is 40E2-F49F

 Directory of C:\PESIT\Go

09-01-2024  16:00    <DIR>          .
09-01-2024  16:00    <DIR>          ..
14-12-2023  09:26         2,904,832 gobook.pdf
09-01-2024  16:00         1,897,472 helloworld.exe
09-01-2024  15:55                85 HelloWorld.go
14-12-2023  11:01         2,897,615 Introducing Go -
20-12-2023  09:39           323,072 Syllabus.doc
14-12-2023  11:01         5,307,238 The Go Programmin
09-01-2024  15:59         1,654,342 Unit 1 - 01 Intro
               7 File(s)     14,984,656 bytes
               2 Dir(s)  49,369,362,432 bytes free
```

Go Programming Language
First program

- Go code is organized into packages, which are similar to libraries or modules in other languages.
- A package consists of one or more **.go** source files in a single directory that define what the package does.
- Each source file begins with a package declaration, here package **main**, that states which package the file belongs to, followed by a list of other packages that it imports, and then the declarations of the program that are stored in that file.
- **fmt** package contains functions for printing formatted output and scanning input.
- **Println** is one of the basic output functions in fmt; it prints one or more values, separated by spaces, with a newline character at the end so that the values appear as a single line of output.

Go Programming Language

First program

- Package **main** is special. It defines a standalone executable program, not a library.
- Within package main the function main is also special - it’s where execution of the program begins.
- Whatever main does is what the program does. Of course, main will normally call upon functions in other packages to do much of the work, such as the function fmt.Println.
- A program will not compile if there are missing imports or if there are unnecessary ones. This strict requirement prevents references to unused packages from accumulating as programs evolve.
- The import declarations must follow the package declaration. After that, a program consists of the declarations of functions, variables, constants, and types (introduced by the keywords func, var, const, and type)
- The order of declarations does not matter

Go Programming Language
First program

- A function declaration consists of the key word func, the name of the function, a parameter list (empty for main), a result list (also empty here), and the body of the function—the statements that define what it does—enclosed in braces.
- Go does not require semicolons at the ends of statements or declarations, except where two or more appear on the same line.
- In effect, newlines following certain tokens are converted into semicolons, so where newlines are placed matters to proper parsing of Go code
- The opening brace **{** of the function must be on the same line as the end of the **func** declaration, not on a line by itself

Command line arguments

- The **os** package provides functions and other values for dealing with the operating system in a platform-independent fashion.
- Command-line arguments are available to a program in a variable named Args that is part of the os package; thus its name anywhere outside the os package is os.Args.
- The variable os.Args is a slice of strings.
- Slices are a fundamental notion in Go. Think of a slice as a dynamically sized sequence s of array elements where individual elements can be accessed as s[i] and a contiguous subsequence as s[m:n].
- The first element of os.Args, os.Args[0], is the name of the command itself
- The other elements are the arguments that were presented to the program when it started execution
- A slice expression of the form s[m:n] yields a slice that refers to elements m through n - 1

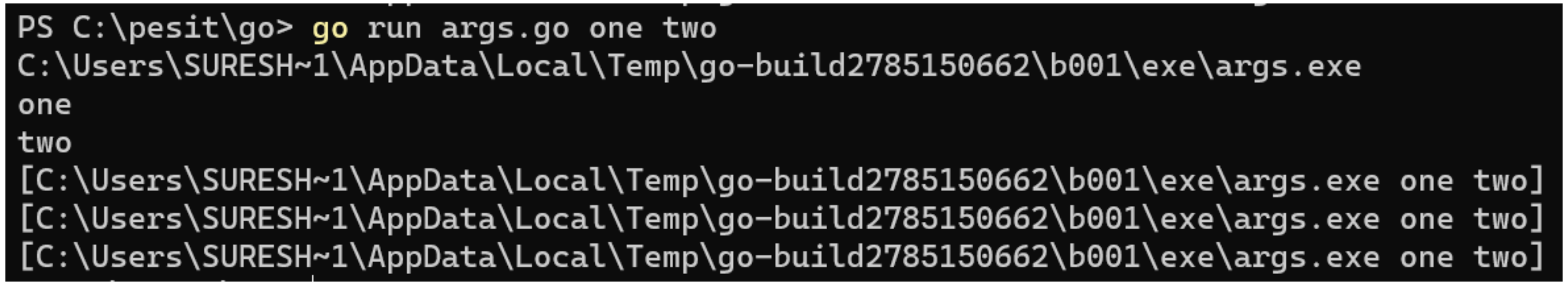
Go Programming Language

Command line arguments

```go
package main

import ("fmt"
        "os")

func main() {
   fmt.Println(os.Args[0])
   fmt.Println(os.Args[1])
   fmt.Println(os.Args[2])
   fmt.Println(os.Args[0:3])
   fmt.Println(os.Args[0:])
   fmt.Println(os.Args[:3])
}
```



```
PS C:\pesit\go> go run args.go one two
C:\Users\SURESH~1\AppData\Local\Temp\go-build2785150662\b001\exe\args.exe
one
two
[C:\Users\SURESH~1\AppData\Local\Temp\go-build2785150662\b001\exe\args.exe one two]
[C:\Users\SURESH~1\AppData\Local\Temp\go-build2785150662\b001\exe\args.exe one two]
[C:\Users\SURESH~1\AppData\Local\Temp\go-build2785150662\b001\exe\args.exe one two]
```

PES UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering

PES UNIVERSITY

CELEBRATING 50 YEARS

Go Programming Language

Program Structure

Department of Computer Science and Engineering

Go Programming Language

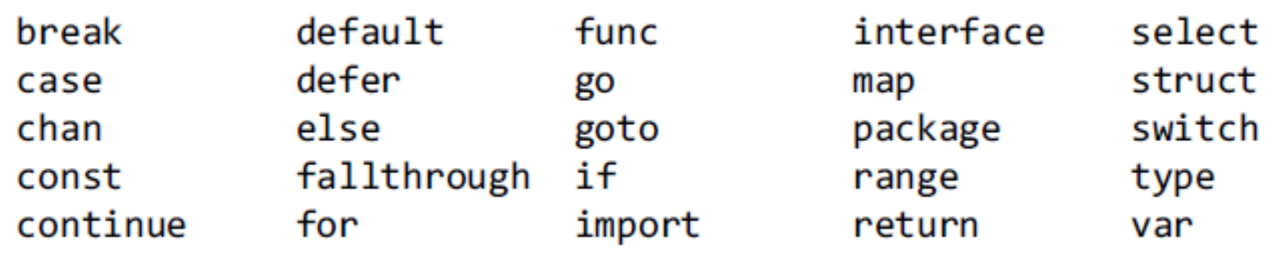
Program structure

- In Go, as in any other programming language, a program is built from a small set of basic constructs.
- **Variables** store values.
- Simple **expressions** are combined into larger ones with **operations** like addition and subtraction.
- Basic types are collected into aggregates like **arrays** and **structs**.
- Expressions are used in statements whose execution order is determined by control-flow statements like if and for.
- Statements are grouped into **functions** for isolation and reuse.
- Functions are gathered into source files and **packages**.

Go Programming Language

Names

- The names of Go - functions, variables, constants, types, statement labels, and packages follow a simple rule
- A name begins with a letter or an underscore and may have any number of additional letters, digits, and underscores.
- Names are case sensitive. heapSort and Heapsort are different names.
- Keywords:

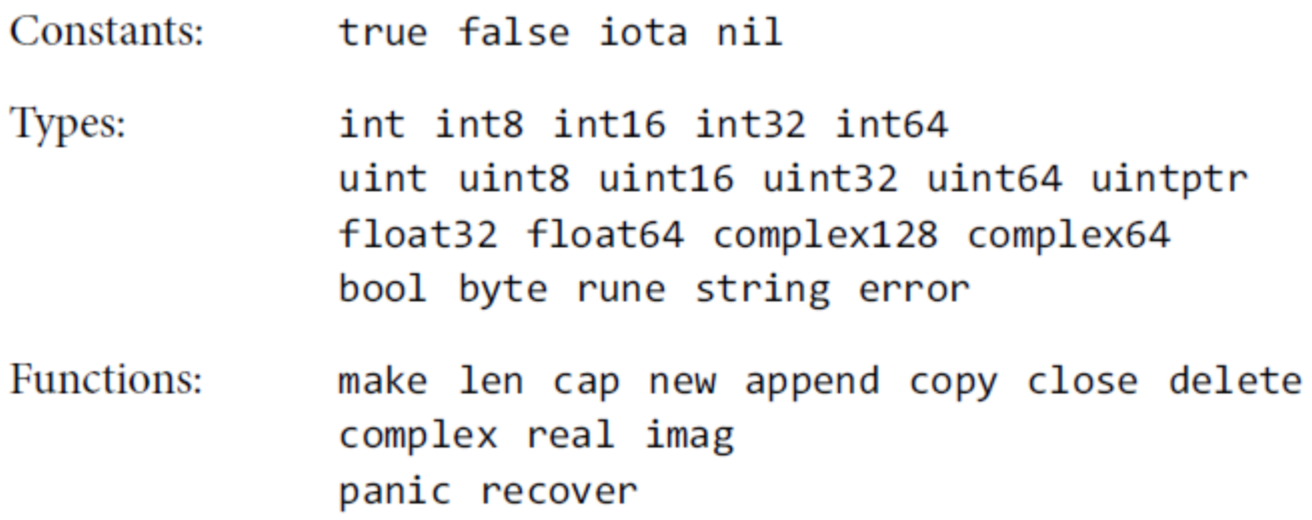


break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

- Keywords can't be used as names.

Go Programming Language

Predeclared Names



Constants:	`true false iota nil`

Types:	`int int8 int16 int32 int64`
	`uint uint8 uint16 uint32 uint64 uintptr`
	`float32 float64 complex128 complex64`
	`bool byte rune string error`

Functions:	`make len cap new append copy close delete`
	`complex real imag`
	`panic recover`

- If an entity is declared within a function, it is local to that function.
- If an entity is declared outside of a function, it is visible in all files of the package to which it belongs.
- The case of the first letter of a name determines its visibility across package boundaries.
- If the name begins with an upper-case letter, it is exported, which means that it is visible and accessible outside of its own package and may be referred to by other parts of the program
- Package names themselves are always in lower case

Go Programming Language
Variables

- A **var** declaration creates a variable of a particular type, attaches a name to it, and sets its initial value.
- Each declaration has the general form **var name type = expression**
- Either the **type** or the **= expression** part may be omitted, but not both
- If the type is omitted, it is determined by the initializer expression.
- If the expression is omitted, the initial value is the zero value for the type, which is 0 for numbers, false for booleans, "" for strings, and nil for interfaces and reference types (slice, pointer, map, channel, function).
- The **zero value** of an aggregate type like an **array** or a **struct** has the zero value of all of its elements or fields.
- The zero-value mechanism ensures that a variable always holds a well-defined value of its type
- In Go there is no such thing as an uninitialized variable
- Within a function, a variable may be declared using the form **name := expression**
- Note that **:=** is a declaration, whereas **=** is an assignment

Go Programming Language

Pointers

- A pointer value is the address of a variable.
- A pointer is thus the location at which a value of the variable is stored.
- Not every value has an address, but every variable does.
- With a pointer, we can read or update the value of a variable indirectly, without using or even knowing the name of the variable.
- If a variable is declared **var x int**, the expression &x (‘‘address of x’’) yields a pointer to an
- integer variable x
- The zero value for a pointer of any type is nil. The test p != nil is true if p points to a variable.
- Pointers are comparable. Two pointers are equal if and only if they point to the same variable or both are nil.
- Because a pointer contains the address of a variable, passing a pointer argument to a function makes it possible for the function to update the variable that was indirectly passed.

Go Programming Language

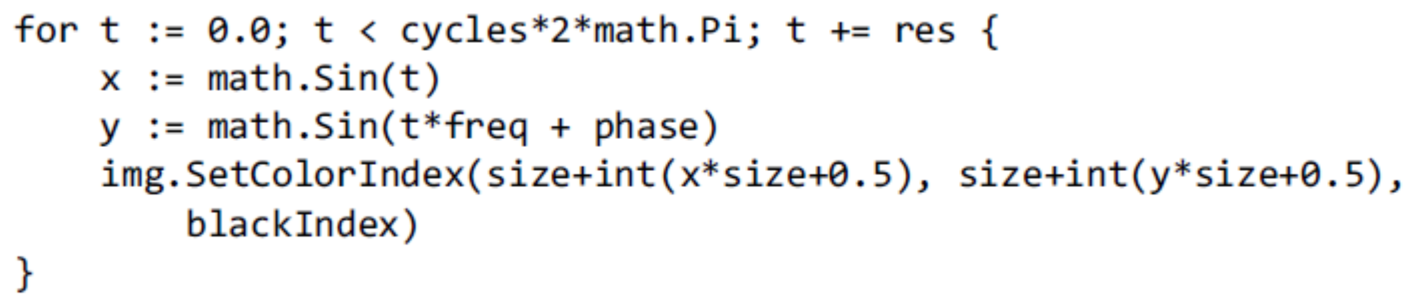
new Function

- Another way to create a variable is to use the built-in function new.
- The expression **new(T)** creates an unnamed variable of type T, initializes it to the zero value of T, and returns its address, which is a value of type *T.
- A variable created with new is no different from an ordinary local variable whose address is taken, except that there’s no need to invent (and declare) a dummy name, and we can use new(T) in an expression.
- Thus new is only a syntactic convenience, not a fundamental notion

Go Programming Language

Lifetime of variables

- The lifetime of a variable is the interval of time during which it exists as the program executes.
- The lifetime of a package-level variable is the entire execution of the program.
- Local variables have dynamic lifetimes: a new instance is created each time the declaration statement is executed, and the variable lives on until it becomes unreachable, at which point its storage may be recycled.
- Function parameters and results are local variables too; they are created each time their enclosing function is called.



```
for t := 0.0; t < cycles*2*math.Pi; t += res {
    x := math.Sin(t)
    y := math.Sin(t*freq + phase)
    img.SetColorIndex(size+int(x*size+0.5), size+int(y*size+0.5),
        blackIndex)
}
```

- The va x and y are created on each iteration of the loop.

Assignment

- The value held by a variable is updated by an assignment statement, which in its simplest form
- has a variable on the left of the **=** sign and an expression on the right.
- An assignment, explicit or implicit, is always legal if the left-hand side (the variable) and the right-hand side (the value) have the same type.
- The assignment is legal only if the value is assignable to the type of the variable.
- Tuple assignment, allows several variables to be assigned at once.

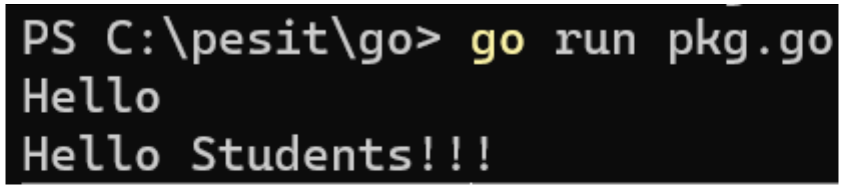
Type declaration

- The type of a variable or expression defines the characteristics of the values it may take on, such as their size, how they are represented internally, the intrinsic operations that can be performed on them, and the methods associated with them.
- A type declaration defines a new named type that has the same underlying type as an existing type.
- **type name underlying_type**
- Type declarations most often appear at package level, where the named type is visible throughout the package

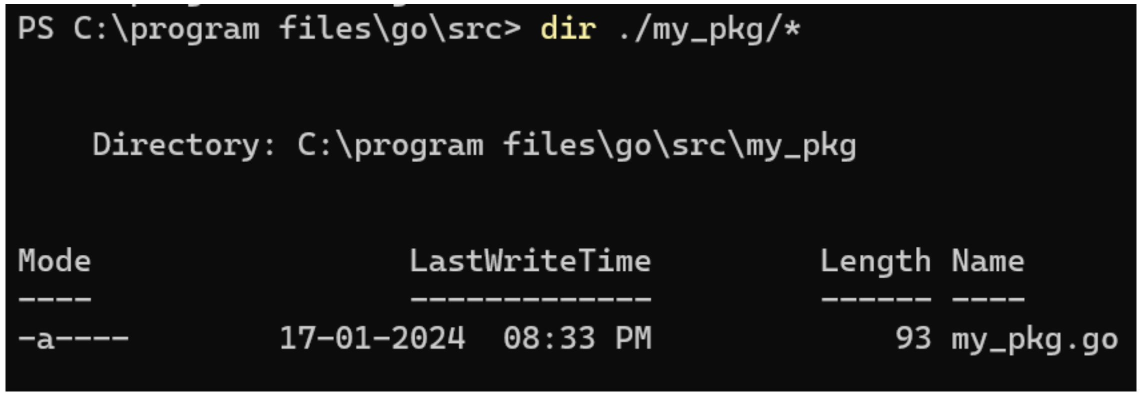
Go Programming Language

Packages and Files

- Packages in Go serve the same purposes as libraries or modules in other languages, supporting modularity, encapsulation, separate compilation, and reuse.
- The source code for a package resides in one or more .go files, usually in a directory whose name ends with the import path
- Each package serves as a separate name space for its declarations
- To refer to a function from outside its package, we must qualify the identifier to make explicit reference.



```
PS C:\pesit\go> go run pkg.go
Hello
Hello Students!!!
```



```
PS C:\program files\go\src> dir ./my_pkg/*


    Directory: C:\program files\go\src\my_pkg


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        17-01-2024   08:33 PM             93 my_pkg.go
```

Go Programming Language

Scope

- The scope of a declaration is a region of the program text
- It is a compile-time property
- A **syntactic block** is a sequence of statements enclosed in braces like those that surround the body of a function or loop.
- A name declared inside a syntactic block is not visible outside that block.
- Other groupings of declarations that are not explicitly surrounded by braces in the source code are called **lexical blocks**
- Lexical block for the entire source code is called the **universe block**
- The declarations of built-in types, functions, and constants like int, len, and true are in the universe
- block and can be referred to throughout the entire program.

Go Programming Language
Scope

- A program may contain multiple declarations of the same name so long as each declaration is in a different lexical block.
- When the compiler encounters a reference to a name, it looks for a declaration, starting with the inner most enclosing lexical block and working up to the universe block.
- If the compiler finds no declaration, it reports an ‘‘undeclared name’’ error.
- If a name is declared in both an outer block and an inner block, the inner declaration will be found first.

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering

Go Programming Language

Data Types

Department of Computer Science and Engineering

Go Programming Language

Basic data types - integers

Integers

- Go provides both signed and unsigned integers.
- There are four distinct sizes of signed integers - 8, 16, 32, and 64 bits - represented by the types int8, int16, int32, and int64, and corresponding unsigned versions uint8, uint16, uint32, and uint64.
- Unsigned integer type **uintptr**, whose width is not specified but is sufficient to hold all the bits of a pointer value. The uintptr type is used only for low-level programming
- Regardless of their size, int, uint, and uintptr are different types from their explicitly sized siblings. Thus int is not the same type as int32, even if the natural size of integers is 32 bits, and an explicit conversion is required to use an int value where an int32 is needed, and vice versa.
- Float to integer conversion discards any fractional part, truncating toward zero.

Go Programming Language

Operations on integers

- Binary operators for arithmetic, logic, and comparison in order of decreasing precedence

```
*    /    %    <<    >>    &    &^
+    -    |    ^
==   !=   <    <=    >     >=
&&
||
```

- Comparison operators

`==`	equal to
`!=`	not equal to
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to

- Bitwise binary operators

`&`	bitwise AND
`|`	bitwise OR
`^`	bitwise XOR
`&^`	bit clear (AND NOT)
`<<`	left shift
`>>`	right shift

Basic data types – floating point numbers

Floating-point numbers

- Go provides two sizes of floating-point numbers, float32 and float64.
- A float32 provides approximately six decimal digits of precision, whereas a float64 provides about 15 digits
- Very small or very large numbers are written in scientific notation, with the letter e or E preceding the decimal exponent: **const Avogadro = 6.02214129e23**

Complex numbers

- Go provides two sizes of complex numbers, complex64 and complex128, whose components are float32 and float64 respectively.
- The built-in function complex creates a complex number from its real and imaginary components, and the built-in real and imag functions extract those components

Go Programming Language

Basic data types – boolean

Boolean

- A value of type bool, or boolean, has only two possible values, true and false.
- The conditions in if and for statements are booleans
- Comparison operators like == and < produce a boolean result.
- Boolean values can be combined with the && (AND) and || (OR) operators
- There is no implicit conversion from a boolean value to a numeric value like 0 or 1, or vice versa.

Go Programming Language

Basic data types – strings

Strings

- A string is an immutable sequence of bytes.
- Strings may contain arbitrary data, including bytes with value 0, but usually they contain human-readable text.
- Text strings are conventionally interpreted as UTF-8 encoded sequences
- The built-in **len** function returns the number of bytes in a string, and the index operation s[i] retrieves the i-th byte of string s, where 0 <= i < len(s)
- The substring operation s[i:j] yields a new string consisting of the bytes of the original string starting at index i and continuing up to, but not including, the byte at index j.
- The + operator makes a new string by concatenating two strings

Go Programming Language

Basic data types – strings

Conversion between strings and numbers

- To convert an integer to a string, use the function strconv.Itoa ()
- To convert a string to an integer, use the function strconv.Atoi ()
- strconv.ParseFloat converts the string s to a floating-point number with the precision specified by bitSize: 32 for float32, or 64 for float64

Go Programming Language

Complex data types – arrays

Arrays

- An array is a fixed-length sequence of zero or more elements of a particular type.
- Because of their fixed length, arrays are rarely used directly in Go.
- Slices, which can grow and shrink, are much more versatile
- Individual array elements are accessed with the conventional subscript notation, where subscripts run from zero to one less than the array length.
- The built-in function **len** returns the number of elements in the array.
- r := [...]int{99: -1} defines an array r with 100 elements, all zero except for the last, which has value −1.
- Two arrays can be directly compared using the == operator, which reports whether all the corresponding elements are equal.

Go Programming Language

Complex data types – slices

Slices

- Slices represent variable-length sequences whose elements all have the same type.
- A slice type is written []T, where the elements have type T; it looks like an array type without a size.
- A slice is a lightweight data structure that gives access to a subsequence (or perhaps all) of the elements of an array, which is known as the slice's underlying array.
- A slice has three components: a pointer, a length, and a capacity.
- The pointer points to the first element of the array that is reachable through the slice, which is not necessarily the array's first element.
- The length is the number of slice elements; it can't exceed the capacity, which is usually the number of elements bet ween the start of the slice and the end of the underlying array.
- The built-in functions len and cap return length and capacity values.

Complex data types – slices

Slices

- Multiple slices can share the same underlying array and may refer to overlapping parts of that array.
- Since a slice contains a pointer to an element of an array, passing a slice to a function permits the function to modify the underlying array elements
- Unlike arrays, slices are not comparable, so we cannot use == to test whether two slices contain the same elements.
- The standard library provides the highly optimized **bytes.Equal** function for comparing two slices of bytes
- The built-in append function appends items to slices
- The zero value of a slice type is nil. A nil slice has no underlying array. The nil slice has length and capacity zero

Go Programming Language

Complex data types – maps

Maps

- Hash table is one of the most ingenious and versatile of all data structures
- In Go, a map is a reference to a hash table, and a map type is written map[K]V, where K and V are the types of its keys and values.
- All of the keys in a given map are of the same type, and all of the values are of the same type, but the keys need not be of the same type as the values.
- Map elements are accessed through the usual subscript notation

Go Programming Language

Complex data types – Struct

Struct

- A struct is an aggregate data type that groups together zero or more named values of arbitrary types as a single entity. Each value is called a field.
- The classic example of a struct from data processing is the employee record, whose fields are a unique ID, the employee’s name, address, date of birth, position, salary, manager, and the like.
- All of these fields are collected into a single entity that can be copied as a unit, passed to functions and returned by them, stored in arrays.
- The individual fields of a struct are accessed using the dot notation
- If all the fields of a struct are comparable, the struct its elf is comparable, so two expressions of that type may be compared using == or !=.

Go Programming Language

JSON

JSON

- Java Script Object Notation (JSON) is a standard notation for sending and receiving structured Information
- Go has excellent support for encoding and decoding these formats, provided by the standard library packages encoding/json
- Converting a Go data structure to JSON is called marshaling. Marshaling is done by json.Marshal

PES UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering

PES UNIVERSITY

CELEBRATING 50 YEARS

Go Programming Language

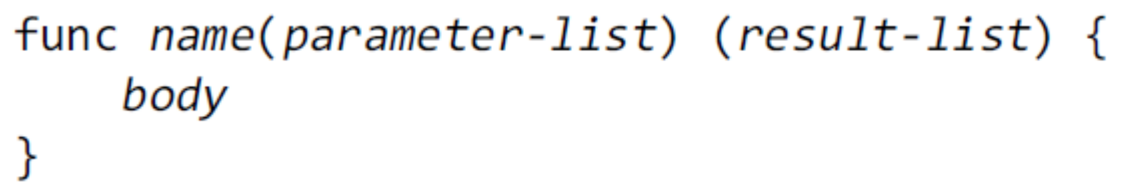
Functions and Methods

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:

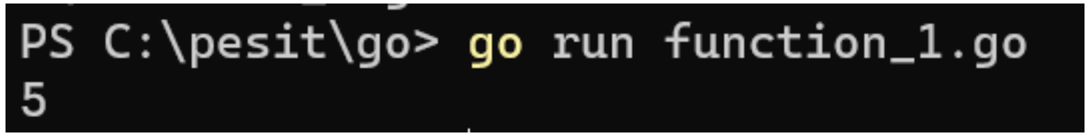
```
package main

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.

```
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.

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.

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

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.

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