GO PROGRAMMING

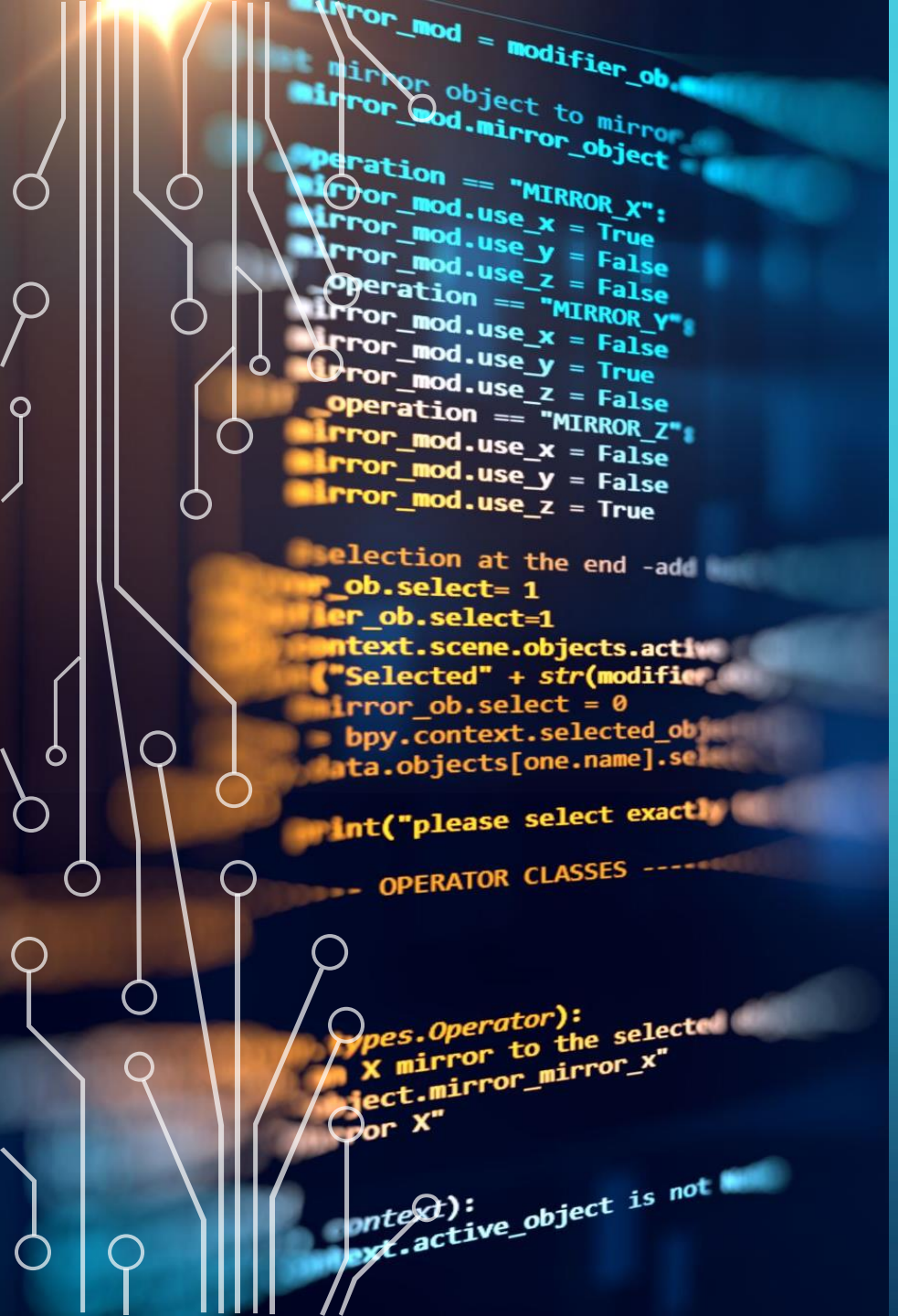
JOHN ROEDER

JACOB BORCHERT

EDWARD HOWELL

SLIDE DECK OVERVIEW

- Origins/The Why of Go
- Data Types and Simple Code Snippets
- Data Structures
- Built In Concurrency and Memory Allocation
- Pointers and Parameter Passing
- Real World Applications of Go



ORIGINS:

- Go was designed by Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson.
- Compiled scripting language.
- Go is statically typed
- Go uses static scoping.
- Released open source in 2012.

WHY GO WAS CREATED

- Designed to be simpler and more straightforward than C and C++
- Combine ease of programming from dynamic languages with efficiency and safety of static typing
- Provide native support for concurrency and parallelism
- Used for building large-scale systems and infrastructure at Google

CODE SNIPPETS AND SYNTAX EXPLAINED

HELLO WORLD IN GO

```
package main // Define the package name. Every Go program starts with a package declaration.

import "fmt"    // Import the "fmt" package, which contains functions for formatting text, including printing to the console.

func main() {    // The main function, where execution of the program begins.

        fmt.Println("Hello, world!")

}
```

DATA TYPES:

- **bool**: A boolean value, true or false.
- **string**: A sequence of characters.
- **Numeric Types:**
  - Integer Types: **int, int8, int16, int32, int64, uint, uint8, uint16, uint32, uint64**
  - Floating-Point Types: **float32, float64**.
  - Complex Number Types: **complex64, complex128**.

VARIABLES AND TYPES SYNTAX

```
package main import "fmt"

func main() {

	var age int = 30 // Declare an integer variable named 'age' and initialize it to 30.

	var name = "John Doe" // Declare a string variable 'name' and infer the type string.

	fmt.Println("Name:", name, "- Age:", age) // Print variables with labels.

	height := 178.5 // Short variable declaration with type inference for a float64.

	fmt.Println("Height:", height, "cm") }
```

CONDITIONALS AND LOOPS:

- Has the conditionals: If-Else Statements, Switch Statements.
- No while loops in the language in the sense that there is no ‘while’ keyword. Uses for loops to emulate the while where it checks for conditions and runs.
- Can iterate over the data structures: array, slice, map, and channel

LOOPS EXPLAINED AND SYNTAX

```
package main import "fmt"

func main() {

 // A simple for loop that prints numbers from 1 to 5

 for i := 1; i <= 5; i++ {

          fmt.Println("Number:", i) }

}
```

HOW TO DO WHILE LOOPS IN GO

```
import "fmt"
func main() {
    condition := true  // This is your boolean condition
    // This will act as a 'while' loop
    for condition {
        fmt.Println("This loop will run until 'condition' is false")
        // Some logic here that might change the condition
        // For demonstration, let's say we set condition to false
        condition = false
}
fmt.Println("Exited the loop") }
```

SYNTAX OVERVIEW

Static Typing: Go is statically typed, meaning variables have types that are checked at compile time.

Semicolon Inference: Semicolons are not required at the end of statements; they are inferred by the compiler.

Package System: Go uses a package-based system for organizing code, with the main package being the entry point for executable programs.

Functions: Functions are defined using the func keyword, followed by the function name and parameters.

GO'S PACKAGE SYSTEM

Go programs are organized into packages.

package main: Entry point package for executable programs.

import "fmt": Importing the fmt package for formatted I/O.

DATA STRUCTURES:

- Maps, Arrays, Slices..
- Structs: a type of composite data type that allows you to group together data. This is essentially how you would make a class in Go. Essentially C Struct.
- Channels: allow for communication between GoRoutines. Used to pass data between GoRoutines and coordinate their execution. Channels are typed.
- Does not offer linkedlists, heaps, or trees, but these can be implemented using structs

MAPS

structures_C.c

```c
//have to define a struct for key val pair
struct KeyValuePair {
    char key[50];
    int value;
};

int main() {
    //to declare map
    struct KeyValuePair myMap[MAX_SIZE];
    int mapSize = 0;

    //very chunky
    strcpy(myMap[mapSize].key, "apple");
    myMap[mapSize].value = 5;
    mapSize++;
}
```

structures.c++

```cpp
#include <iostream>
#include <map>
#include <string>

int main() {
    // make map
    std::map<std::string, int> myMap;

    // add to map
    myMap["apple"] = 5;
    myMap["banana"] = 7;
    myMap["orange"] = 3;

    //to access map vals
    std::cout << "Value of apple: " << myMap["apple"] << std::endl;
    std::cout << "Value of banana: " << myMap["banana"] << std::endl;
```

MAPS PT2

structures_GO.go

```go
//Maps in Go

package main
import "fmt"

func main(){
    //make map string key value int
    bigMap := make(map[string]int)

    //add to the map. VERY python esq
    bigMap["John"]= 1

    //accessing elements is very similar to python as well with...
    fmt.Println(bigMap["John"])
}
```

SLICES EX.

```go
func slice() {
    // Create the slice
    slice := []int{1, 2, 3, 4, 5}

    //Access slice
    fmt.Println("First element:", slice[0])
    fmt.Println("Second element:", slice[1])

    //create an array
    //difference is we must define size since arrs are not dynamic in GO.
    my_arr = [3]int(1, 2, 3)

    //to take a slice of an array
    slice1 := my_arr[1:3]
}
```

BUILT IN CONCURRENCY

```go
func nums() {
    for i := 1; i <= 5; i++ {
        fmt.Println("go:", i)
        time.Sleep(100 * time.Millisecond)
    }
}

func goRoutine() {
    //initiate the go routine
    go nums()

    // concurrency
    for i := 1; i <= 5; i++ {
        fmt.Println("Main Function:", i)
        time.Sleep(100 * time.Millisecond)
    }
}
```

CHANNELS

```go
func producer(ch chan<- int) {
    defer close(ch)
    for i := 0; i < 5; i++ {
        fmt.Println("Producing", i)
        ch <- i
        time.Sleep(time.Second)
    }
}

func consumer(ch <-chan int) {
    for val := range ch {
        fmt.Println("Consuming", val)
        time.Sleep(2 * time.Second)
    }
}

func main() {
    ch := make(chan int)

    go producer(ch)
    go consumer(ch)


    time.Sleep(10 * time.Second)
}
```

MEMORY ALLOCATION/DEALLOCATION

- Go uses Mark and Sweep for its garbage collection.
- GO marks the values it encounters as live and once the tracing is complete, it makes all the memory that is not marked available for allocation.
- Specifically: Tri-color mark and sweep.

POINTERS:

- Go does support pointer arithmetic and pointers to pointers.
- You can also pass pointers to functions.
- You declare pointers in Go by using the * symbol.
- Pointers are then initialized using the & operand:
  - Var point *int
  - Var num int = 2
  - point = &num

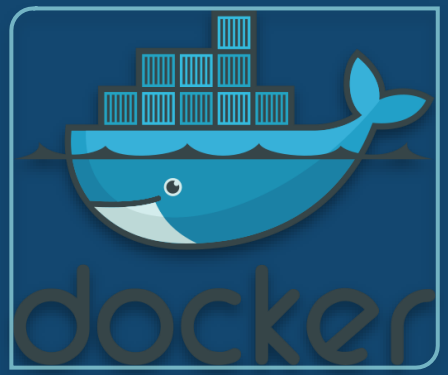
PARAMETER PASSING:

- Parameter passing is the mechanism used when arguments are passed to functions
- In Go, parameters are passed by value, or by reference.
- Go does not support pass-by-reference for primitive types.
  - If you need to pass a primitive type to a function by reference, you can use a pointer.

```
func addOne(val int) {
  val = val + 1
}
```

REAL WORLD APPLICATIONS

- Web Development
- Cloud Services (ex. Docker):
- Networking
- Real-Time Applications (ex. Instant messaging or video streaming sites)
- Companies like Google, Uber, Twitch, Dropbox all use Go.



INTERESTING THINGS ABOUT GO

- Go does not require the use of a semicolon to end a statement, although semicolons are valid to end statements.
- Go does not support inheritance.
- You can use Imaginary numbers in Go.
- No list data structure, uses Slices instead. These are very similar to python lists.

SOURCES/QUESTIONS

- https://go.dev/
- https://www.gopl.io/ch1.pdf
- https://www.geeksforgeeks.org/go-programming-language-introduction/