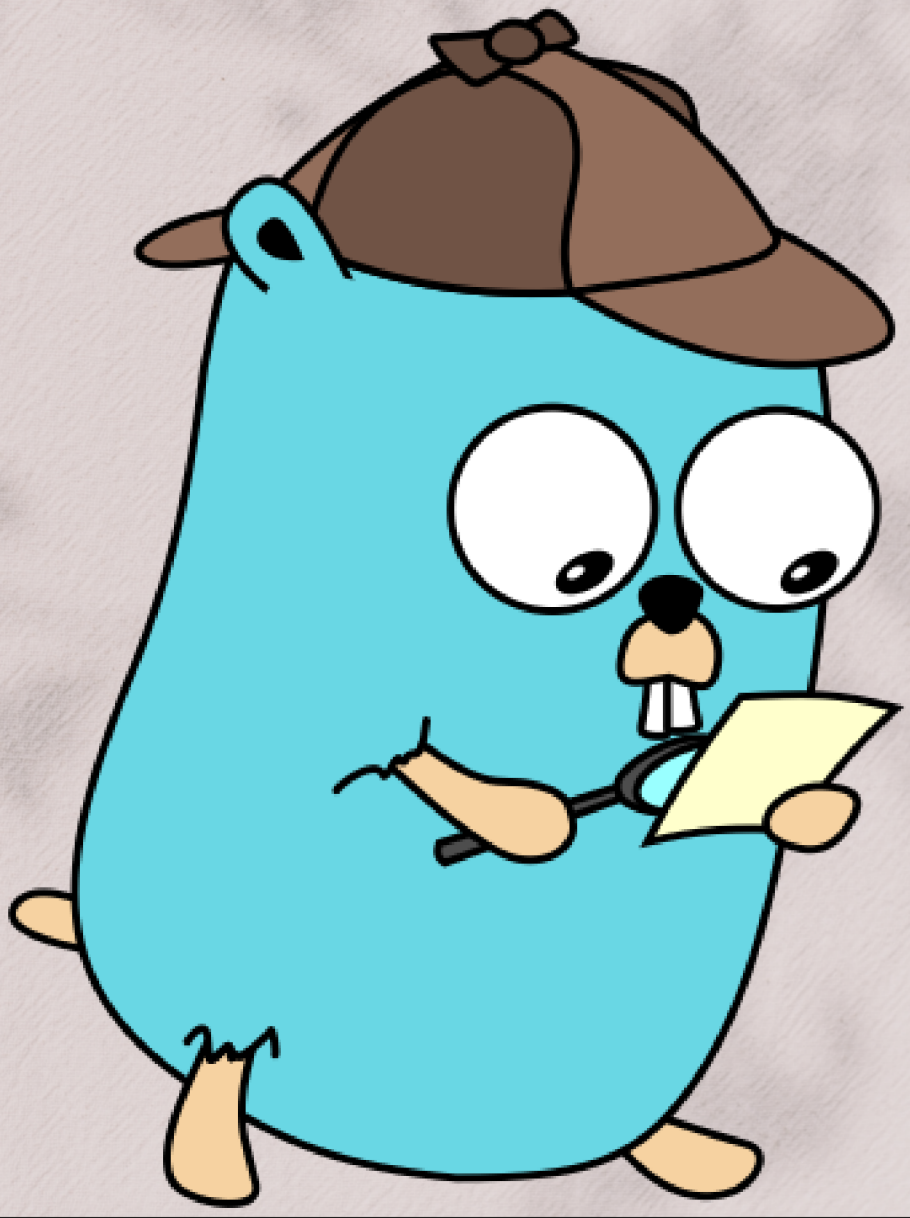
GOLANG CHEAT SHEET

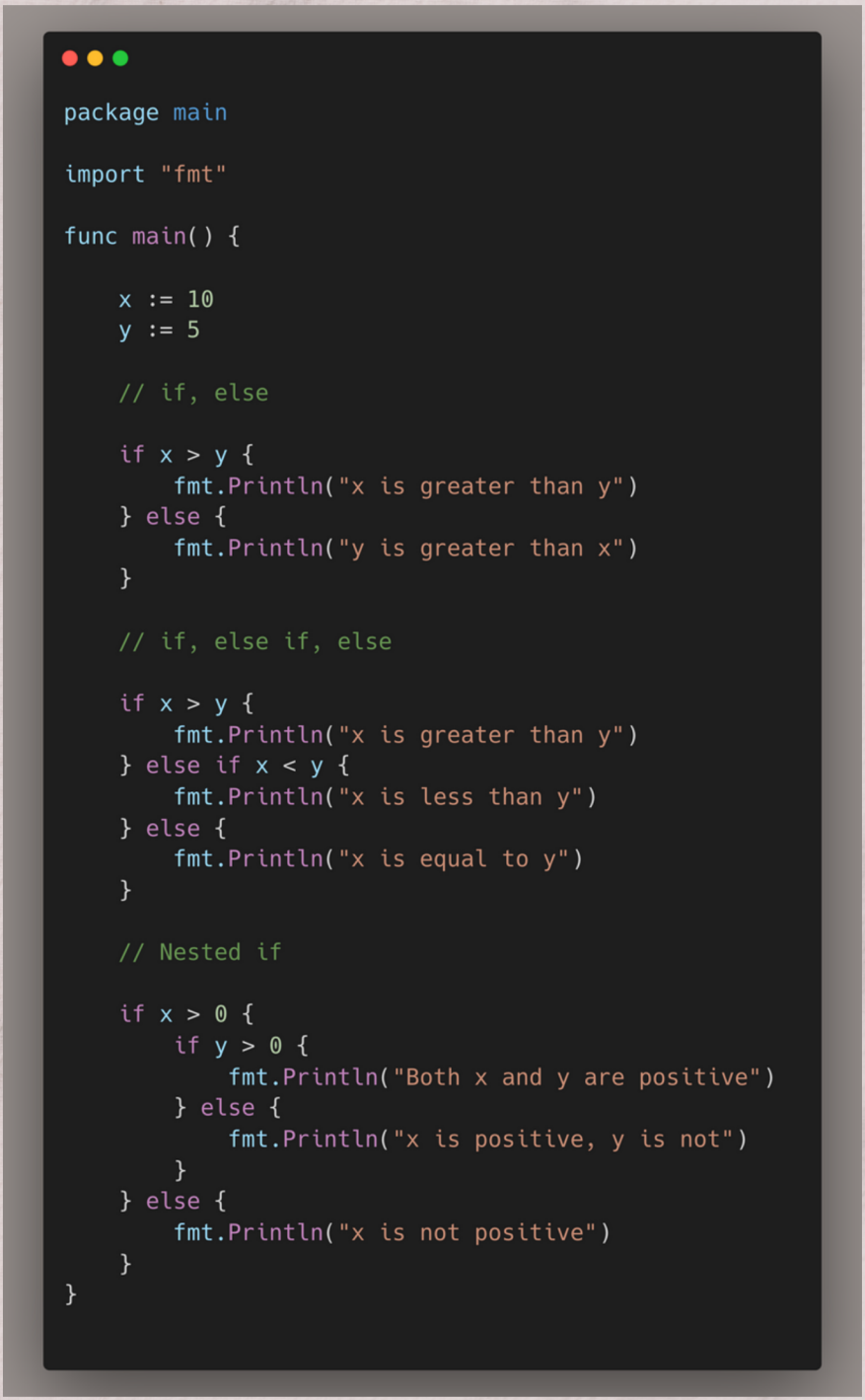
PART - 1



Amizhthan M

SWIPE

Conditional Statement



```go
package main

import "fmt"

func main() {

    x := 10
    y := 5

    // if, else

    if x > y {
        fmt.Println("x is greater than y")
    } else {
        fmt.Println("y is greater than x")
    }

    // if, else if, else

    if x > y {
        fmt.Println("x is greater than y")
    } else if x < y {
        fmt.Println("x is less than y")
    } else {
        fmt.Println("x is equal to y")
    }

    // Nested if

    if x > 0 {
        if y > 0 {
            fmt.Println("Both x and y are positive")
        } else {
            fmt.Println("x is positive, y is not")
        }
    } else {
        fmt.Println("x is not positive")
    }
}
```

Conditional statements in Go, such as `if`, `else`, and `else if`, allow code to be executed selectively based on conditions. They enable branching logic, executing different code blocks depending on whether certain conditions evaluate to true or false.

Amizhthan M

Go switch case

```go
package main

import "fmt"

func main() {
    // Example 1: Switch statement with string cases
    fruit := "banana"

    switch fruit {
    case "apple":
        fmt.Println("It's an apple.")
    case "banana":
        fmt.Println("It's a banana.")
    case "orange":
        fmt.Println("It's an orange.")
    default:
        fmt.Println("It's another fruit.")
    }

    // Example 2: Switch statement with boolean conditions
    number := 5

    switch {
    case number > 0 && number <= 3:
        fmt.Println("Number is between 1 and 3.")
    case number > 3 && number <= 6:
        fmt.Println("Number is between 4 and 6.")
    default:
        fmt.Println("Number is not within the specified ranges.")
    }

    // Example 3: Switch statement with variable initialization
    switch season := "summer"; season {
    case "spring":
        fmt.Println("It's spring.")
    case "summer":
        fmt.Println("It's summer.")
    case "autumn":
        fmt.Println("It's autumn.")
    case "winter":
        fmt.Println("It's winter.")
    }
}
```

Go's switch statement enables selective execution of code based on different cases, evaluating values or conditions, with an optional default case for unmatched scenarios.

Amizhthan M

Iterative statements

```go
package main

import "fmt"

func main() {

    // For loop

    for i := 0; i < 5; i++ {
        fmt.Println(i)
    }

    // For...range loop

    numbers := []int{1, 2, 3, 4, 5}
    for index, value := range numbers {
        fmt.Println("Index:", index, "Value:", value)
    }

    // While-like loop

    count := 0
    for count < 5 {
        fmt.Println("Count:", count)
        count++
    }

}
```

Iterative statements in Go, such as 'for' loops, enable repetitive execution of a block of code based on a specified condition or range. They allow iterating over collections, performing computations, and implementing various types of loops with precise control over loop variables and termination conditions.

Amizhthan M

Go break and continue

```go
package main

import "fmt"

func main() {

    // For loop with break

    for i := 1; i <= 5; i++ {
        if i == 3 {
            break
        }
        fmt.Println("For loop:", i)
    }

   // For loop with continue

    for i := 1; i <= 5; i++ {
        if i == 3 {
            continue
        }
        fmt.Println("For loop:", i)
    }

}
```

In Go, the break statement is used to exit the current loop prematurely, stopping further iterations. The continue statement, on the other hand, is used to skip the remaining code in the current iteration and move to the next iteration of the loop, allowing further iterations to continue.

Amizhthan M