Guida alla Programmazione in Go

1. Cos'è Go?

Go, o **Golang**, è un linguaggio di programmazione compilato, fortemente tipizzato e progettato per avere una sintassi semplice e leggibile. È ottimizzato per le prestazioni, con un'ottima gestione della concorrenza, il che lo rende una scelta popolare per lo sviluppo di applicazioni server e sistemi distribuiti.

2. Installazione e Ambiente di Lavoro

Per iniziare a programmare in Go:

1. **Installa Go**: Scarica il pacchetto di installazione dal sito ufficiale golang.org.
2. **Configura il PATH**: Aggiungi Go al tuo PATH di sistema per usare i comandi `go` dalla linea di comando.

Verifica l'installazione: Apri il terminale e digita:
bash
Copia codice
```
go version
```

3.
4. **Editor di testo**: Consigliato l'uso di Visual Studio Code o GoLand per il supporto Go integrato.

3. Struttura di un Progetto Go

Un progetto Go è organizzato con una struttura specifica per il codice e i pacchetti. Puoi creare la tua prima cartella di progetto ed esplorarne l'organizzazione.

Esempio di struttura:

bash
Copia codice
```
myproject/
├── go.mod          # File per la gestione delle dipendenze
└── main.go         # File principale del programma
```

Per creare un progetto, usa il comando `go mod init`:

bash

Copia codice
```
go mod init nome_progetto
```

4. Il Primo Programma in Go

Scrivi il tuo primo programma in un file chiamato `main.go`:

go
Copia codice
```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

Esegui il programma:

bash
Copia codice
```
go run main.go
```

5. Sintassi di Base

Dichiarazione di Variabili e Tipi di Dati

Go è un linguaggio tipizzato staticamente e supporta variabili di diversi tipi.

go
Copia codice
```
var nome string = "Alice"     // Stringa
var età int = 30              // Intero
var altezza float64 = 1.75     // Float
var isAdmin bool = true        // Booleano

// Dichiarazione e assegnazione implicita
città := "Roma"               // Go deduce il tipo di variabile
```

Tipi di Dati Principali

- **string**: Sequenza di caratteri, es. `"Ciao"`
- **int**, **float64**, **bool**: Integer, Float, Boolean
- **array** e **slice**: Collezioni di elementi (fixed-size per array, dinamica per slice)
- **map**: Collezione di coppie chiave-valore

Operatori

- **Aritmetici**: `+`, `-`, `*`, `/`, `%`
- **Confronto**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logici**: `&&`, `||`, `!`
- **Di assegnazione**: `=`, `+=`, `-=`, `*=`, `/=`

Condizioni

Go utilizza **if** senza parentesi, e le parentesi graffe `{}` sono obbligatorie.

go
Copia codice
```
if età >= 18 {
    fmt.Println("Sei maggiorenne")
} else {
    fmt.Println("Sei minorenne")
}
```

6. Cicli

Go supporta solo il ciclo `for`, che può essere usato in varie forme:

Ciclo `for` standard

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

Ciclo `for` con condizione

go
Copia codice
```
i := 0
for i < 5 {
    fmt.Println(i)
```

```
    i++
}
```

Ciclo `for` infinito

go
Copia codice
```
for {
    fmt.Println("Loop infinito")
}
```

7. Array e Slice

Gli **array** in Go hanno dimensione fissa, mentre i **slice** sono collezioni di dimensione variabile.

Array

go
Copia codice
```
var numeri [5]int
numeri[0] = 1
fmt.Println(numeri)  // Output: [1 0 0 0 0]
```

Slice

go
Copia codice
```
numeri := []int{1, 2, 3, 4, 5}
fmt.Println(numeri)
```

Puoi aggiungere elementi a un slice con `append`:

go
Copia codice
```
numeri = append(numeri, 6)
fmt.Println(numeri)  // Output: [1 2 3 4 5 6]
```

8. Mappe

Le mappe in Go sono collezioni di coppie chiave-valore, simili a dizionari in altri linguaggi.

go
Copia codice
```go
età := map[string]int{
    "Alice": 25,
    "Bob":   30,
}
fmt.Println(età["Alice"])
```

9. Funzioni

Le funzioni in Go sono definite con la parola chiave `func`:

go
Copia codice
```go
func saluta(nome string) string {
    return "Ciao, " + nome
}

fmt.Println(saluta("Marco"))
```

Le funzioni possono avere più valori di ritorno:

go
Copia codice
```go
func sommaEProdotto(a int, b int) (int, int) {
    return a + b, a * b
}

somma, prodotto := sommaEProdotto(3, 4)
fmt.Println(somma, prodotto)  // Output: 7 12
```

10. Strutture (Struct)

In Go, le **struct** sono simili alle classi in altri linguaggi e vengono utilizzate per creare tipi di dati complessi.

go
Copia codice
```go
type Studente struct {
    nome string
    età  int
```

```
}

studente := Studente{nome: "Alice", età: 20}
fmt.Println(studente.nome)
```

Metodi

In Go, puoi definire metodi associati a una struct:

go
Copia codice

```
func (s Studente) saluta() string {
    return "Ciao, sono " + s.nome
}

fmt.Println(studente.saluta())
```

11. Puntatori

Go utilizza i puntatori per fare riferimento alla memoria. Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile.

go
Copia codice

```
x := 5
p := &x  // Puntatore a x
fmt.Println(*p)  // Dereferenziazione: Output 5
```

12. Concorrenza con Goroutine

Una delle funzionalità principali di Go è la gestione della concorrenza con le **goroutine**. Le goroutine sono funzioni che possono essere eseguite in modo concorrente.

go
Copia codice

```
func saluta() {
    fmt.Println("Ciao dal goroutine")
}

func main() {
    go saluta()           // Lancia saluta() come goroutine
```

```
    fmt.Println("Ciao dal main")
    time.Sleep(time.Second)  // Attende per permettere alla
goroutine di completarsi
}
```

13. Canali (Channels)

I **channels** permettono la comunicazione tra goroutine.

go
Copia codice

```
func saluta(ch chan string) {
    ch <- "Ciao dal goroutine"
}

func main() {
    ch := make(chan string)
    go saluta(ch)
    fmt.Println(<-ch)  // Riceve il messaggio dal channel
}
```

14. Gestione degli Errori

In Go, gli errori sono valori e vengono gestiti con la parola chiave `error`.

go
Copia codice

```
func dividi(a, b int) (int, error) {
    if b == 0 {
        return 0, fmt.Errorf("divisione per zero")
    }
    return a / b, nil
}

risultato, err := dividi(10, 0)
if err != nil {
    fmt.Println("Errore:", err)
} else {
    fmt.Println("Risultato:", risultato)
}
```

15. Modulo e Gestione delle Dipendenze

Go gestisce le dipendenze con il file `go.mod`:

Crea un nuovo modulo:
bash
Copia codice
```
go mod init nome_progetto
```

1.

Per installare una libreria, usa il comando:
bash
Copia codice
```
go get nome_pacchetto
```

2.

16. Esecuzione e Compilazione di un Programma Go

Puoi eseguire un programma Go con:

bash
Copia codice
```
go run main.go
```

Per compilare il programma in un file eseguibile:

bash
Copia codice
```
go build main.go
```

17. Conclusione

Go è un linguaggio potente, moderno e ideale per sviluppatori che cercano prestazioni