Custom Errors: You can define custom error types by implementing the Error() method.

```
package main

import (
 "fmt"
)

type MyError struct {
 Message string
}

func (e *MyError) Error() string {
 return e.Message
}

func doSomething(value int) error {
 if value < 0 {
 return &MyError{Message: "value cannot be negative"}
 }
 return nil
}

func main() {
 err := doSomething(-1)
 if err != nil {
 fmt.Println("Custom Error:", err)
 }
}
// Custom Error: value cannot be negative
```

File Handling

Go provides support for reading and writing files using the os and io/ioutil packages.

Reading & Writing Files

Writing to a File

```go
package main

import (
 "fmt"
 "os"
)

func main() {
 content := []byte("Hello, File Handling in Go!")
 err := os.WriteFile("example.txt", content, 0644)
 if err != nil {
 fmt.Println("Error writing file:", err)
 return
 }
 fmt.Println("File written successfully")
}
// File written successfully
```

Reading from a File

```go
package main

import (
 "fmt"
 "os"
)

func main() {
 data, err := os.ReadFile("example.txt")
 if err != nil {
 fmt.Println("Error reading file:", err)
 return
 }
 fmt.Println("File content:", string(data))
}
// File content: Hello, File Handling in Go!
```

Working with JSON: GO provides the encoding/json package to work with JSON data.

Encoding (Struct to JSON)

```
package main

import (
 "encoding/json"
 "fmt"
)

type Person struct {
 Name string `json:"name"`
 Age int `json:"age"`
}

func main() {
 person := Person{Name: "Alice", Age: 25}
 jsonData, _ := json.Marshal(person)
 fmt.Println(string(jsonData))
}
// {"name":"Alice","age":25}
```

Decoding (JSON to Struct)

```
package main

import (
 "encoding/json"
 "fmt"
)

type Person struct {
 Name string `json:"name"`
 Age int `json:"age"`
}

func main() {
 jsonData := `{"name":"Bob","age":30}`
 var person Person
 json.Unmarshal([]byte(jsonData), &person)
 fmt.Printf("Name: %s, Age: %d\n", person.Name, person.Age)
}
// Name: Bob, Age: 30
```

HTTP & Web Development

Go provides built-in support for HTTP and web development using the net/http package.

Creating an HTTP Server

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World!")
}

func main() {
    http.HandleFunc("/", handler)
    fmt.Println("Server is running on port 8080...")
    http.ListenAndServe(":8080", nil)
}
// Server is running on port 8080...
```

Visit http://localhost:8080/ in a web browser, and it will display:

Hello, World!

Handling Requests & Responses

```
package main

import (
    "encoding/json"
    "fmt"
    "net/http"
)

type Response struct {
    Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
    response := Response{Message: "Hello, JSON!"}
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(response)
}

func main() {
    http.HandleFunc("/json", handler)
    fmt.Println("Server is running on port 8080...")
    http.ListenAndServe(":8080", nil)
}
// Server is running on port 8080...
```

Visit **http://localhost:8080/json** in a web browser or API tool, and it will return:

```
{"message":"Hello, JSON!"}
```

Making HTTP Requests

GET Request

```
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    resp, err :=
http.Get("https://jsonplaceholder.typicode.com/todos/1")
    if err != nil {
        fmt.Println("Error:", err)
        return
    }
    defer resp.Body.Close()

    body, _ := ioutil.ReadAll(resp.Body)
    fmt.Println(string(body))
}
```

Visit **http://localhost:8080/json** in a web browser or API tool, and it will return:

```
{
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
}
```