1. What is the purpose of the http.HandleFunc function?

http.HandleFunc registers a handler function for the given pattern. Whenever an HTTP request is received for that pattern, the registered function is called to handle the request.

1. Why do we check the request method in the handler function using r.Method?

We check the request method to ensure that the client is using the correct HTTP method for the request. For example, if the client is trying to use a POST request to perform a read operation, we would return a "Method Not Allowed" response.

1. What is the purpose of json.Marshal?

json.Marshal serializes a Go value to JSON format. In this case, we are serializing a Go value (returned by data.getAll()) to JSON so that we can return it to the client.

1. What is the purpose of json.NewDecoder and Decode?

json.NewDecoder creates a new JSON decoder that reads from an io.Reader (in this case, r.Body). Decode decodes the JSON-encoded data and stores it in the value pointed to by its argument (&req in this case).

1. Why do we use w.WriteHeader and w.Write in the handler function?

w.WriteHeader is used to set the HTTP response status code (e.g. 200 OK, 404 Not Found). w.Write writes the response body (in this case, the serialized JSON data) to the HTTP response.

1. Why do we set the "Content-Type" header to "application/json"?

We set the "Content-Type" header to "application/json" to indicate to the client that the response body is in JSON format.

1. What is the purpose of panic(err) at the end of the code?

panic(err) is used to terminate the program if an error occurs while starting the HTTP server. This ensures that the program does not continue running with an invalid server configuration.

1. What is the purpose of the http.HandleFunc function?

http.HandleFunc sets up a request handler for the specified path pattern, and invokes the given handler function when a matching request is received.

1. What does http.ListenAndServe do?

http.ListenAndServe starts an HTTP server that listens for incoming requests on the specified network address (in this case, ":8080") and calls the given handler for each request.

1. What is the purpose of the http.ResponseWriter interface?

http.ResponseWriter is used to write the response that the server sends back to the client, including headers and body.

1. What is the purpose of the http.Request struct?

http.Request represents an incoming HTTP request, and contains information such as the HTTP method, headers, URL path, and request body.

1. What is the purpose of w.WriteHeader()?

w.WriteHeader() sets the HTTP response status code that the server will send back to the client. It must be called before any data is written to the response body.

1. What is the purpose of json.NewEncoder()?

json.NewEncoder() returns a new json.Encoder instance that can be used to encode Go values into JSON and write them to an output stream.

1. What is the purpose of json.NewDecoder()?

json.NewDecoder() returns a new json.Decoder instance that can be used to decode JSON data from an input stream into Go values.

1. What is the purpose of the json.Marshal() function?

json.Marshal() encodes a Go value into a JSON-encoded byte array.

1. What is the purpose of the json.Unmarshal() function?

json.Unmarshal() decodes a JSON-encoded byte array into a Go value.

1. What is the purpose of the fmt.Fprintf() function?

fmt.Fprintf() formats and writes data to the specified io.Writer, using a format string and a variadic list of arguments. In this case, it is used to write JSON-formatted error messages to the response body.

1. What are some advantages of using Golang over other programming languages? Answer: Golang has several advantages over other programming languages, including fast compilation times, efficient memory usage, built-in concurrency support, and a strong focus on simplicity and readability.
2. What is a Goroutine in Golang? Answer: A Goroutine is a lightweight thread managed by the Go runtime. Goroutines allow for concurrent execution of multiple functions within the same process.
3. What is a Channel in Golang? Answer: A Channel in Golang is a typed conduit through which Goroutines communicate with one another. Channels provide a safe and efficient way to pass messages between Goroutines.
4. What is a Pointer in Golang? Answer: A Pointer in Golang is a variable that holds the memory address of another variable. Pointers are often used in Golang to pass large data structures between functions without incurring the overhead of copying the data.
5. What is a Struct in Golang? Answer: A Struct in Golang is a composite data type that groups together zero or more values of different types. Structs are often used in Golang to represent complex data structures or to define custom data types.
6. What is a Slice in Golang? Answer: A Slice in Golang is a variable-length sequence of elements of the same type. Slices are often used in Golang to represent collections of data that can be dynamically resized at runtime.
7. What is the defer keyword in Golang? Answer: The defer keyword in Golang is used to schedule a function call to be executed at the end of the current function or method. Deferring a function call can be useful for resource cleanup, error handling, or other tasks.
8. What is a Panic in Golang? Answer: A Panic in Golang is a run-time error that indicates that the program has entered an unrecoverable state. When a Panic occurs, the program will stop execution and print a stack trace of the error.
9. What is an Interface in Golang? Answer: An Interface in Golang is a type that defines a set of methods that a concrete type must implement in order to satisfy the interface. Interfaces are often used in Golang to enable polymorphism and abstraction.
10. What is the difference between a Map and a Slice in Golang? Answer: A Map in Golang is a built-in data type that represents a collection of key-value pairs. A Slice in Golang is a variable-length sequence of elements of the same type. Maps are often used in Golang to represent associative arrays or lookup tables, while Slices are often used to represent sequences of data that can be dynamically resized.