**Context - Standard Library**

The package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes.

**Notes**

* Incoming requests to a server should create a Context.
* Outgoing calls to servers should accept a Context.
* The chain of function calls between them must propagate the Context.
* Replace a Context using WithCancel, WithDeadline, WithTimeout, or WithValue.
* When a Context is canceled, all Contexts derived from it are also canceled.
* Do not store Contexts inside a struct type; instead, pass a Context explicitly to each function that needs it.
* Do not pass a nil Context, even if a function permits it. Pass context.TODO if you are unsure about which Context to use.
* Use context Values only for request-scoped data that transits processes and APIs, not for passing optional parameters to functions.
* The same Context may be passed to functions running in different goroutines; Contexts are safe for simultaneous use by multiple goroutines.

**Encoding - Standard Library**

Encoding is the process or marshaling or unmarshaling data into different forms. Taking JSON string documents and convert them to values of our user defined types is a very common practice in many go programs today. Go's support for encoding is amazing and improves and gets faster with every release.

**Notes**

* Support for Decoding and Encoding JSON and XML are provided by the standard library.
* This package gets better and better with every release.

**io - Standard Library**

The ability to stream and pass data around is incredibility important. Data is constantly coming at our programs whether over a socket, file, device, etc. Many times this data needs to just be moved from one stream. Sometimes it needs to be encrypted, hashed or stored for safe keeping. The Writer and Reader interfaces may be the most heavily used and supported interfaces in both the standard library and the community.

**Notes**

* The standard library provides all the infrastructure we need to stream and work with data.
* If we implement the Reader and Writer interfaces in our own types, we get this functionality for free.
* Implementing interfaces to existing functionality saves us from both development time and bugs.

**Logging - Standard Library**

Logging is an important part of every program. Logs are our eyes and history to what is happening inside the program as it runs. The standard library provides the log package to support the basic features of logging and the ability for you to extend and customize logging to fit your needs.

**Notes**

* Support for logging is already built into the standard library.
* The log package can be easily extended to meet your logging needs.

**Reflection - Standard Library**

Reflection is the ability to inspect a value to derive type or other meta-data. Reflection can give our program incredible flexibility to work with data of different types or create values on the fly. Reflection is critical for the encoding and decoding of data.

**Notes**

* The reflection package allows us to inspect our types.
* We can add "tags" to our struct fields to store and use meta-data.
* Encoding package leverages reflection and we can as well.