**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.

**Concurrency Patterns**

There are lots of different patterns we can create with goroutines and channels. Two interesting patterns are resource pooling and concurrent searching.

**Notes**

* The work code provides a pattern for giving work to a set number of goroutines without losing the guarantee.
* The resource pooling code provides a pattern for managing resources that goroutines may need to acquire and release.
* The search code provides a pattern for using multiple goroutines to perform concurrent work.