Golang Error Handling Cheat Sheet

1. What is an Error?

An error in Go is any value that implements the `error` interface:

```
type error interface {
    Error() string
}
```

Errors are usually returned as the last return value from functions. A nil error means success.

2. Creating Errors

- errors.New("message") -> creates a basic error
- fmt.Errorf("formatted message %v", value) -> creates formatted errors

Example:

```
return errors.New("cannot divide by zero")
return fmt.Errorf("user %s not found", user)
```

3. Best Practices

- Always check errors explicitly.
- Provide meaningful messages with context.
- Use wrapping (%w) to preserve the original error.
- Avoid ignoring errors (_ = func()).

4. Custom Error Types

We can define structs that implement Error() string to create structured errors.

Example:

```
type MyError struct {
    Code int
    Message string
}
func (e MyError) Error() string {
    return fmt.Sprintf("Code %d: %s", e.Code, e.Message)
}
```

5. Wrapping & Unwrapping (Go 1.13+)

- fmt.Errorf("context: %w", err) -> wrap error
- errors.Is(err, target) -> check specific error

- errors.As(err, &target) -> check type of error

6. Panic vs Error

- Error -> expected, recoverable problem (e.g., file not found)
- Panic -> unexpected, unrecoverable problem (e.g., nil pointer dereference)
- Use panic only for truly exceptional cases