# Overview of the Standard Library

This lesson emphasizes the importance of packages and provides one-line descriptions of some famous packages from the standard Go library.

WE'LL COVER THE FOLLOWING



## Introduction

The Go-distribution contains over 250 standard built-in packages for common functionality, like `fmt`, `os`, ... , designated as the standard library written in the Go language itself (besides some low-level routines). See the documentation here. The API in all packages (included package `os`) is the same for all systems (Windows, Linux, ...); the only package that is different for each system is `syscall`. In the examples and exercises throughout the course, we use the packages of the standard library.

## Common packages

Here we will discuss the general purpose of a number of common packages grouped by function; we will not go into detail about their inner structure.

- `unsafe`: contains commands to step out of the Go type-safety which is not needed in normal programs. It can be useful when interfacing with C/C++.
- `os`: gives a platform-independent interface to operating-system functionality. Its design is Unix-like. It hides the differences between

various operating systems to give a consistent view of files and other OS-objects.

- `os/exec`: gives the possibility to run external OS commands and programs.
- `syscall`: this is the low-level, external package, which provides a primitive interface to the underlying OS's calls.
- `archive/tar and /zip – compress`: contains functionality for (de)compressing files.
- `fmt`: contains functionality for formatted input-output.
- `io`: provides basic input-output functionality, mostly as a wrapper around os-functions.
- `bufio`: wraps around io to give buffered input-output functionality.
- `path/filepath`: contains routines for manipulating filename paths targeted at the OS used.
- `flag`: contains functionality to work with command-line arguments.
- `strings`: contains functions for manipulating and processing strings.
- `strconv`: converts strings to basic data types.
- `unicode`: contains special functions for Unicode characters.
- `regexp`: provides sophisticated pattern-searching functionalities for strings.
- `bytes`: contains functions for the manipulation of byte slices.
- `index/suffixarray`: contains methods for very fast searching in strings.
- `math`: contains the basic mathematical constants and functions.
- `math/cmplx`: methods for manipulating complex numbers.
- `math/rand`: contains pseudo-random number generators.
- `sort`: contains functionality for sorting arrays and user-defined collections.
- `math/big`: contains multi-precision arithmetic methods for working with arbitrarily large integers and rational numbers.
- `container`: contains sub-packages that implement containers for manipulating collections, for example:
  - `list`: for working with doubly-linked lists.
  - `ring`: for working with circular lists.
- `time`: contains basic functionalities for working with times and dates.

- `log`: contains functionalities for logging information in a running program. We'll use it throughout examples in the following chapters.
- `encoding/json`: implements the functions for reading/decoding as well as writing/encoding data in JSON format.
- `encoding/xml`: this is a simple XML 1.0 parser for examples of JSON and XML.
- `text/template`: use this package to make data-driven templates that can generate textual output mixed with data, like HTML.
- `net`: contains basic functions for working with network data.
- `http`: contains functionality for parsing HTTP requests/replies, provides an extensible HTTP server and a basic client.
- `html`: this is a parser for HTML5.
- `crypto – encoding – hash - ...`: these form a multitude of packages for encrypting and decrypting data.
- `runtime`: contains operations for interacting with the Go-runtime, such as the garbage collection and goroutines.
- `reflect`: implements runtime introspection, allowing a program to manipulate variables with arbitrary types.

---

These were some common packages used in a Go program. In the next lesson, you have to write a program to solve a problem.