Erlang Cheatsheet

A quick reference guide to the Erlang programming language, covering syntax, data types, concurrency, and OTP principles.

Erlang Basics

Syntax Fundamentals

Variable Assignment	Erlang uses single assignment. Variables start with an uppercase letter. `X = 10.`
Atoms	Atoms are literal constants, starting with a lowercase letter. `status = ok.`
Comments	Single-line comments start with `%`. `% This is a comment`
Tuples	Tuples are compound data types. `Point = {10, 20}.`
Lists	Lists are dynamic arrays. `Numbers = [1, 2, 3].`
Strings	Strings are lists of character codes. `Name = "Erlang".`

Basic Operators

Arithmetic	`+`, `-`, `*`, `/`, `div`, `rem`
Comparison	`==`, `/=`, `<`, `>`, `=<`, `=>`
Boolean	`and`, `or`, `xor`, `not`
List Operators	`++`, `--` (append and subtract lists)

Concurrency

Processes

Spawning Processes	Use `spawn` to create a new process. `spawn(Module, Function, Args).`
Sending Messages	Use `!` to send messages to a process. `ReceiverPid ! {self(), Message}.`
Receiving Messages	Use `receive` to handle incoming messages. `receive {Sender, Message} -> io:format("Received ~p from ~p~n", [Message, Sender]) end.`
Process Identifiers (PIDs)	Returned by `spawn`, used to identify processes.

Message Handling

Messages are the primary means of communication between Erlang processes. They are asynchronous and can be any Erlang term.

The `receive` block selectively receives messages based on pattern matching. Messages that don't match remain in the mailbox.

Use `after` to specify a timeout for the `receive` block.

```
receive
  Message ->
    ...
after 5000 ->
    io:format("Timeout~n")
end.
```

OTP Principles

Supervisors

Supervisors are processes that monitor and restart other processes (children) in case of failure. They ensure the system's fault tolerance.

Common supervision strategies include `one_for_one`, `rest_for_one`, and `one_for_all`.

Example:

```
{simple_one_for_one, {local,
my_supervisor},
    [{my_worker, {my_worker, start_link,
[]}, permanent, brutal_kill, worker,
[my_worker]}]}.
```

Behaviours

`gen_server`	Generic server behaviour for stateful processes.
`gen_statem`	Generic state machine behaviour.
`gen_event`	Generic event handler behaviour.
`supervisor`	Behaviour for creating supervisor processes.

Applications

Applications are a collection of modules, processes, and other resources that form a reusable component. They provide a way to package and manage Erlang code.

An application resource file (`.app`) defines the application's metadata, such as its name, description, and dependencies.

Common Built-in Functions (BIFs)

Process Related

`self()`	Returns the PID of the current process.
`spawn(Module, Function, Args)`	Spawns a new process.
`exit(Reason)`	Terminates the current process with the given reason.
`erlang:monitor(process, Pid)`	Sets up a monitor for the specified process.

Data Type Conversion

`list_to_atom(List)`	Converts a list to an atom.
`atom_to_list(Atom)`	Converts an atom to a list.
`list_to_integer(List)`	Converts a list to an integer.
`integer_to_list(Integer)`	Converts an integer to a list.

I/O

`io:format(Format, Args)`	Prints formatted output.
`file:read_file(Filename)`	Reads the contents of a file.