[i] Passing a single tuple rather than a pile of arguments gives Erlang much of its flexibility, especially when you get to passing messages between different processes

[i] When you create and deliver Erlang programs, you will be distributing them as a set of compiled BEAM files. You don't need to compile each one from the shell

*[i] Most Erlang programming (beyond tinkering in the shell) is creating functions in modules and connecting them into larger programs.*

[i] In Erlang, greater-than-or-equal-to is written >=, and less-than-orequal-to is written =<. Don't make them look like arrows.

## MODULES

**CHAPTER 5*

### Recursion. Factorial example

```erlang
-module(rec_factorial).
-export([factorial/1]).

factorial(N) when N > 1 ->
   N * factorial(N - 1);
factorial(N) when N =< 1 ->
   1.
```

### Recursion. Counting up

```erlang
-module(rec_counting_up).
-export([countup/1]).

% public
```

```
countup(Limit) ->
   countup(1, Limit).

% private
countup(Count, Limit) when Count =< Limit ->
   io:format("~w~n", [Count]),
   countup(Count + 1, Limit);
countup(Count, Limit) when Count >= Limit ->
   io:format("Finished.~n").
```

**Recursion. Counting down**

```
-module(rec_counting_down).
-export([countdown/1]).

countdown(From) when From > 0 ->
   io:format("~w~n", [From]),
   countdown(From-1); % clauses

countdown(From) when From == 0 ->
   io:format("blastoff!~n").
```

**stdout print**

```
io:format("message")
```

**If operator**

```
-module(meet_if).
```

```
-export([fall_velocity/2]).

fall_velocity(Planemo, Distance) when Distance >= 0 ->
   Gravity =
       case Planemo of
           earth -> 9.8;
           moon -> 1.6;
           mars -> 3.71
       end,

   Velocity = math:sqrt(2 * Gravity * Distance),

   Description =
       if
           Velocity == 0 -> 'stable';
           Velocity < 5 -> 'slow';
           % ','
           Velocity >= 5, Velocity < 10 -> 'moving';
           % 'and
           (Velocity >= 10) and (Velocity < 20) -> 'fast';
           Velocity >= 20 -> 'speedy'
       end,

   if
       % send message to stdout
       (Velocity > 40) -> io:format("Look out below!")
   end,

   Description.
```

**Guards in case statements**

```erlang
-module(case_guards).
-export([fall_velocity/2]).

fall_velocity(Planemo, Distance) ->
   Gravity =
       case Planemo of
           earth when Distance >= 0 -> 9.8;
           moon when Distance >= 0 -> 1.6;
           mars when Distance >= 0 -> 3.71
       end,

   math:sqrt(2 * Gravity * Distance).
```

**Case operator**

```erlang
-module(meet_case).
-export([fall_velocity/2]).

% guard
fall_velocity(Planemo, Distance) when Distance >= 0 ->
   Gravity =
       case Planemo of
           earth -> 9.8;
           moon -> 1.6;
           mars -> 3.71;
           % default case
           _ -> fail
       end,

   Result =
       case Gravity of
           fail -> 'First argument - Planemo - is invalid';
           _ -> math:sqrt(2 * Gravity * Distance)
       end,

   Result.
```

**Element. Extract from tuple by index**

```
-module(extract_from_tuple).
-export([display/1]).

display(Data) -> element(1,Data) + element(3,Data).
```

**Encapsulating arguments in a tuple and passing them to a private function**

```
-module(public_private).
-export([fall_velocity/1]).

% public /1 function
fall_velocity({Planet, Distance}) -> fall_velocity(Planet,
Distance).

% private /2 function
fall_velocity(earth, Distance) -> math:sqrt(2 * 9.8 * Distance);
fall_velocity(moon, Distance) -> math:sqrt(2 * 1.6 * Distance);
fall_velocity(mars, Distance) -> math:sqrt(2 * 9.8 * Distance).
```

**Clauses with Tuples (‘Match’ pattern)**

```
-module(matching_with_tuples).
-export([fall_velocity/1]).

fall_velocity({earth, Distance}) -> math:sqrt(2 * 9.8 *
Distance);

fall_velocity({moon, Distance}) -> math:sqrt(2 * 1.6 *
Distance);

fall_velocity({mars, Distance}) -> math:sqrt(2 * 9.8 *
Distance).
```

**Guards**

*drop_space_guards.erl*

```
-module(drop_space_guards).
-export([fall_velocity/2]).

%% Clauses + Guards
fall_velocity(earth, Distance) when Distance >= 0 -> math:sqrt(2
* 9.8 * Distance);
fall_velocity(moon, Distance) when Distance >= 0 -> math:sqrt(2
* 1.6 * Distance);
fall_velocity(mars, Distance) when Distance >= 0 -> math:sqrt(2
* 3.71 * Distance).
```

**Clauses (‘Match’ pattern)**

*drop_space.erl*

```
-module(drop_space).
-export([fall_velocity/2]).

%% Clauses
fall_velocity(earth, Distance) -> math:sqrt(2 * 9.8 * Distance);
fall_velocity(moon, Distance) -> math:sqrt(2 * 1.6 * Distance);
fall_velocity(mars, Distance) -> math:sqrt(2 * 3.71 * Distance).
```

**Specify the input-output format for EDoc**

*useless.erl*

```
-spec(tripled_sound(number()) -> number()).

Will be shown in Docs as followed:
tripled_sound(Sound::number()) -> number()
```

**Documenting Modules with EDoc**

*useless.erl*

```
%% @author Ruslan Bakanov <ruslan.bakanov@ooma.com>
[https://google.com]
%% @doc Some useless functions to practice Erlang Modules
%% @reference from <a href="https://google.com">The Link</a>
%% Ruslan Bakanov, 2024
%% @copyright 2024 by Ruslan Bakanov
%% @version 0.1

-module(useless).
```

```
-export([clean_sound/0, clean_sound/1, doubled_sound/1,
tripled_sound/1]).

%% @doc Calculates the velocity of an object falling on Earth
%% as if it were in a vacuum (no air resistance). The distance
is
%% the height from which the object falls, specified in meters,
%% and the function returns a velocity in meters per second.
clean_sound() -> 0.
clean_sound(Sound) -> Sound.
doubled_sound(Sound) -> Sound * 2.
tripled_sound(Sound) -> Sound * 3.
```

*Generate docs, in the shell:*

```
edoc:files(["useless.erl","drop.erl"], [{dir, "doc"}]).
```

**Comments. Syntax and variety**

```
-module(useless_public). % comment in the end of string

%%% - flush left

%% - intended with surrounded code
```

**All-public module. Export all functions using one string (not recommended, Warning triggered)**

*useless_public.erl*

```
-module(useless_public).
```

```
% export_all flag enabled - all functions will be exported
-compile(export_all).

clean_sound() -> 0.
clean_sound(Sound) -> Sound.
doubled_sound(Sound) -> Sound * 2.
tripled_sound(Sound) -> Sound * 3.
```

**Import modules inside a module (not recommended)**

*combined_import.erl*

```
-module(combined_import).
-import(useless,[tripled_sound/1]).
-import(drop, [fall_velocity/1]).
-export([calculate/1]).
calculate(Value) -> tripled_sound(fall_velocity(Value)).
```

**Use other modules inside a module**

combined.erl

```
-module(combined).
-export([calculate/1]).
calculate(Value) ->
useless:tripled_sound(drop:fall_velocity(Value)).
```

**Another example**

useless.erl

```
-module(useless).
```

```erlang
-export([clean_sound/0, clean_sound/1, doubled_sound/1,
tripled_sound/1]).

clean_sound() -> 0.
clean_sound(Sound) -> Sound.
doubled_sound(Sound) -> Sound * 2.
tripled_sound(Sound) -> Sound * 3.
```

**Module example**

drop.erl

```erlang
-module(drop).
-export([fall_velocity/1, mps_to_mph/1, mps_to_kph/1]).

fall_velocity(Distance) -> math:sqrt(2 * 9.8 * Distance).

mps_to_mph(Mps) -> 2.23693629 * Mps.

mps_to_kph(Mps) -> 3.6 * Mps.
```

https://github.com/simonstl/introducing-erlang-2nd/blob/master/ch02/ex1-drop/drop.erl

**SHELL**

**Tuples**

```
7> Tuple1 = {coordinates,1,2}.
{coordinates,1,2}
8> element(1,Tuple1).
coordinates
```

```
9> tuple_size(Tuple1).
3
10>
```

**Combine modules**

```
useless:tripled_sound(drop:fall_velocity(1)).
```

**Call the module**

```
drop:mps_to_kph(10).
```

**Compile the module**

From the folder which collects the module:

```
c({filename_without_ext}).
c(drop)

ls().
drop.beam     drop.erl
```

**Call the function**

```
FallVelocity(3).
```

**Multiple statement function**

```
FallVelocity2 = fun(Distance) -> X = (2 * 9.8 * Distance), math:sqrt(X) end.
```

**Define function**

```
FallVelocity = fun(Distance) -> math:sqrt(2 * 9.8 * Distance) end.
```

**Unbind all variables**

```
f().
```

**Unbind variable**

```
f(X).
```

**Show variables list**

```
b().
```

**Define variable (Erlang variable are in fact constants)**

```
X=1.
```

**Other commands**

```
cd().
```

```
ls().
```

**Display current folder**

```
pwd().
```

**Run Erlang**

```
erl
```

**Installation**

I’ve installed Erlang 26 using homebrew as follows

Erlang Quick Install