Erlang CheatSheet v1.0

Variable **% comments (starts with upper case)**

Str = “John Doe”. % strings are stored as integers.

% Writing the value on output
Io:format(“Name is: ~s~n”, [Str]).
Io:fwrite(“Name is: ~s~n”, [Str]).
~n => new line
~s => string
~f => float
~w=>standard output. Like Object.toString().
~p=>like ~w but breaks after each line

List = [1,2,3,4]. % lists
NewList = [6, 7, List] returns [6,7, [1,2,3,4]] % appends to a new list
29> [H|T] = AList.
["a","b",{1,2,3}] % returns the **H**ead and **T**ail. H and T are Unbound variables.

Tuple = {1.0, 2.0, 3.0}
element(2, Tuple) returns 2.0 % tuple index
{_, Second, _} = Tuple stored 2.0 in **Second** variable % pattern matching to retrieve a value

List and Tuple can contain any type.
Atuple = {1,2,3}.
{1,2,3}
AList = ["a", "b", Atuple].
["a","b",{1,2,3}]
Anewtuple = {atom1, atom2, AList}.
{atom1,atom2,["a","b",{1,2,3}]

Functions:
Anonymous:
F = fun(X) -> X end. % F(10) prints 10
Named:
```
method_name(Arg) ->
    Arg.
```

% Perform action on each element on list
L = [1,2,3,4,5].
[1,2,3,4,5]
lists:map(fun(X) -> 2*X end, L). **% using map method of lists module**
[2,4,6,8,10]
45> [2*X || X <- L]. **% or using list comprehensions**
[2,4,6,8,10]

atom or ‘ATOM’

Macros:
-define(macro1, Replacement)
?macro1 % to use the macro

Erlang Shell:
c(ModuleName) % compile module on erlang shell
cd(“dirname”) % change directory on shell
f() to clear all existing bindings
rr("records.hrl") to read a records file
rf(record_name) to forget a record

Records:
-record(todo, {status=reminder,who=john,text}).
- to create an instance of record: X = #todo{status=urgent}.
- extracting values from record is similar to pattern matching
- X#todo.status %% to get a single value.

- ++ is the infix append operator
- [1] ++ [2] ++ [3] = [1,2,3]
- X--Y is the list subtraction operator. It subtracts the elements of Y from X.

Use pattern matching/recursion to replace iteration.
total([{What,N|T}]) -> cost(What) * N + total(T);
total([]) -> 0.

File attributes % -import, -export, -module
-module(ModuleName)
-export([Func_a/0, Func_b/1]).
- import to import the module and methods % -import(lists, [map/2]).
- Includes File: -include(Filename). % -include_lib(Name).

case Expression of
    Pattern1 [when Guard1] -> Expr_seq1;
    Pattern2 [when Guard2] -> Expr_seq2
end.