WHAT THE IF

`If`s act like guards and share guards' syntax, but outside of a function clause's head. In fact, the `if` clauses are called *Guard Patterns*. Erlang's `if`s are different from the `if`s you'll ever encounter in most other languages; compared to them they're weird creatures that might have been more accepted had they had a different name. When entering Erlang's country, you should leave all you know about `if`s at the door. Take a seat because we're going for a ride.

To see how similar to guards the if expression is, look at the following examples:

```
-module(what_the_if).
-export([heh_fine/0]).


heh_fine() ->
if 1 =:= 1 ->
works
end,
if 1 =:= 2; 1 =:= 1 ->
works
end,
if 1 =:= 2, 1 =:= 1 ->
fails
end.
```

Save this as `what_the_if.erl` and let's try it:

```
1> c(what_the_if).
./what_the_if.erl:12: Warning: no clause will ever
match
./what_the_if.erl:12: Warning: the guard for this
clause evaluates to 'false'
{ok,what_the_if}
2> what_the_if:heh_fine().
** exception error: no true branch found when
evaluating an if expression
in function  what_the_if:heh_fine/0
```

Uh oh! the compiler is warning us that no clause from the if on line 12 (`1 =:= 2, 1 =:= 1`) will ever match because its only guard evaluates to `false`. Remember, in Erlang, everything has to return something, and `if` expressions are no exception to the rule. As such, when Erlang can't find a way to have a guard succeed, it will crash: it cannot *not* return something. As such, we need to add a catch-all branch that will always succeed no matter what. In most languages, this would be called an 'else'. In Erlang, we use 'true' (this explains why the VM has thrown "no true branch found" when it got mad):

```
oh_god(N) ->
if N =:= 2 -> might_succeed;
true -> always_does %% this is Erlang's if's 'else!'
end.
```

And now if we test this new function (the old one will keep spitting warnings, ignore them or take them as a reminder of what not to do):

```
3> c(what_the_if).
./what_the_if.erl:12: Warning: no clause will ever
match
./what_the_if.erl:12: Warning: the guard for this
clause evaluates to 'false'
{ok,what_the_if}
4> what_the_if:oh_god(2).
might_succeed
5> what_the_if:oh_god(3).
always_does
```

Here's another function showing how to use many guards in an `if` expression. The function also illustrates how any expression must return something: *Talk* has the result of the `if` expression bound to it, and is then concatenated in a string, inside a tuple. When reading the code, it's easy to see how the lack of a `true` branch would mess things up, considering Erlang has no such thing as a null value (ie.: lisp's nil, C's NULL, Python's None, etc):

```
%% note, this one would be better as a pattern match in
function heads!
%% I'm doing it this way for the sake of the example.
```

```
help_me(Animal) ->
Talk = if Animal == cat -> "meow";
Animal == beef -> "mooo";
Animal == dog -> "bark";
Animal == tree -> "bark";
true -> "fgdadfgna"
end,
{Animal, "says " ++ Talk ++ "!"}.
```

And now we try it:

```
6> c(what_the_if).
./what_the_if.erl:12: Warning: no clause will ever
match
./what_the_if.erl:12: Warning: the guard for this
clause evaluates to 'false'
{ok,what_the_if}
7> what_the_if:help_me(dog).
{dog,"says bark!"}
8> what_the_if:help_me("it hurts!").
{"it hurts!","says fgdadfgna!"}
```

You might be one of the many Erlang programmers wondering why 'true' was taken over 'else' as an atom to control flow; after all, it's much more familiar. Richard O'Keefe gave the following answer on the Erlang mailing lists. I'm quoting it directly because I couldn't have put it better:

It may be more FAMILIAR, but that doesn't mean 'else' is a good thing. I know that writing '; true ->' is a very easy way to get 'else' in Erlang, but we have a couple of decades of psychology-of-programming results to show that it's a bad idea. I have started to replace:

```
                                by

      if X > Y -> a()          if X > Y  -> a()

       ; true  -> b()           ; X =< Y -> b()

      end                      end
```

```
if X > Y -> a()       if X > Y -> a()
 ; X < Y -> b()        ; X < Y -> b()
 ; true  -> c()        ; X ==Y -> c()
end                 end
```

which I find mildly annoying when _writing_ the code but enormously helpful when _reading_ it.

'Else' or 'true' branches should be "avoided" altogether: `if`s are usually easier to read when you cover all logical ends rather than relying on a *"catch all"* clause.
As mentioned before, there are only a limited set of functions that can be used in guard expressions (we'll see more of them in Types (or lack thereof)). This is where the real conditional powers of Erlang must be conjured. I present to you: the `case` expression!

Note: All this horror expressed by the function names in `what_the_if.erl` is expressed in regards to the `if` language construct when seen from the perspective of any other languages'`if`. In Erlang's context, it turns out to be a perfectly logical construct with a confusing name.

Source :http://learnyousomeerlang.com/syntax-in-functions