Explicando Erlang

Roberto Alves Neto

```
-module(echo).
-export([go/0, loop/0]).


go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.


loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

-> **função**

```
-module(echo).
-export([go/0, l

go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.


loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

chama a func loop no modulo echo passando nenhum parâmetro e salva seu pid

```
-module(echo).
-export([go/0, loop/0]).


go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.


loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

envia a mensagem 'hello' para 'loop'

```
-module(echo).
-export([go/0, loop/0]).


go() ->
    Pid2 = spawn(
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.


loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

Analogicamente igual a condição switch case

```erlang
-module(echo).
-export([go/0, loop/0]).


go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.

loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

-> envia 'stop' para 'loop'

```erlang
-module(echo).
-export([go/0, loop/0]).


go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        {Pid2, Msg} ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.


loop() ->
    receive
        {From, Msg} ->
            From ! {self(), Msg},
            loop();
        stop ->
    end.
```

depois de ser chamado uma vez fica em loop infinito empilhando mensagens para 'go'

quando receber essa instrução, para