Efene Programming Language

Mariano Guerra - Erlang Factory - London 2010

Me

C, ASM, C++

Java, Python

PHP, JS, C#

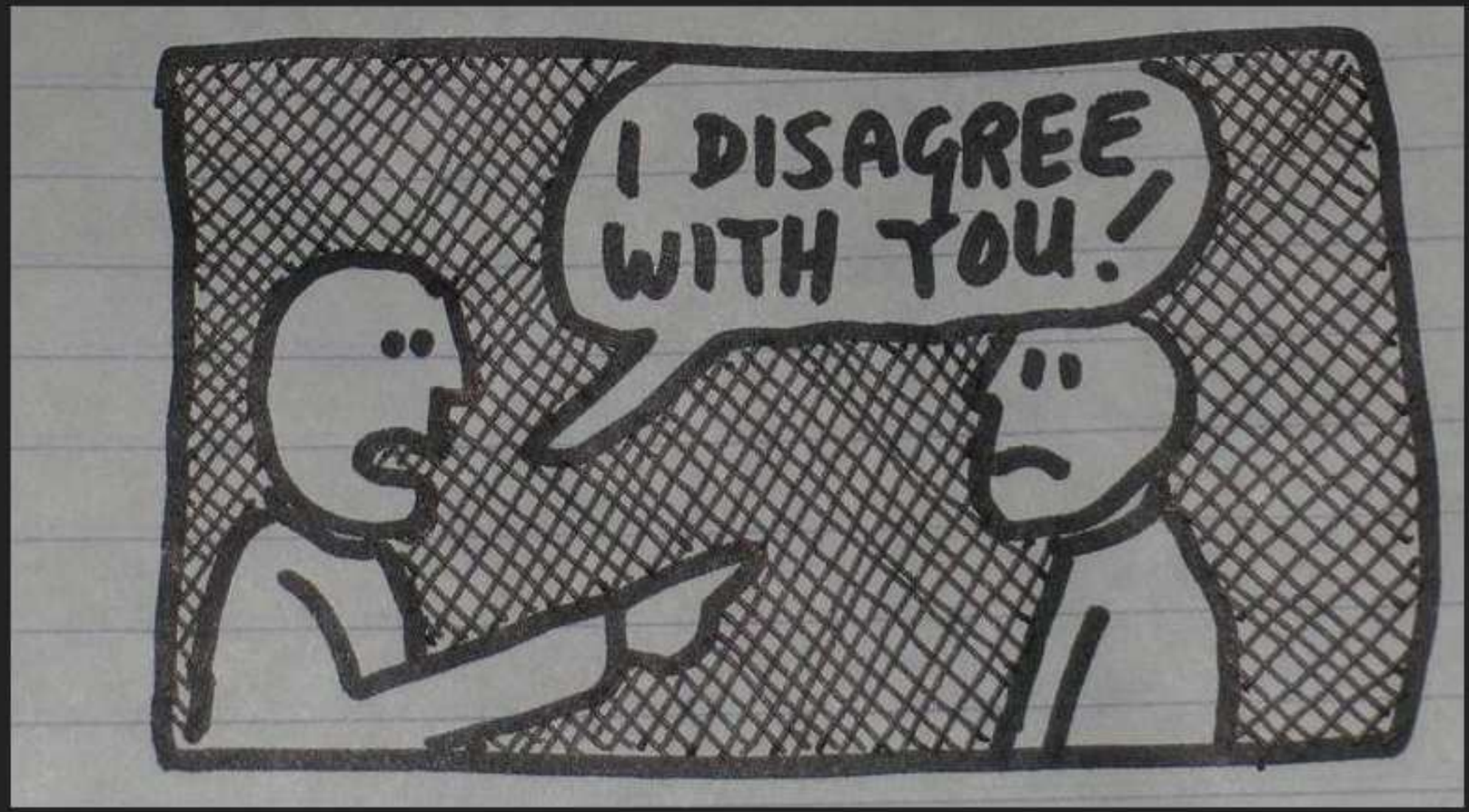
Erlang

I ♥ Python & JS

f(x)?

Why?

Disclaimer


I DISAGREE WITH YOU!

Erlang **rules**

New technologies aren't adopted because they are great, new, and disruptive; they are adopted only if the user's crisis solved by the technology is greater than the perceived pain of adoption

http://www.arcfn.com/2008/07/why-your-favorite-language-is-unpopular.html

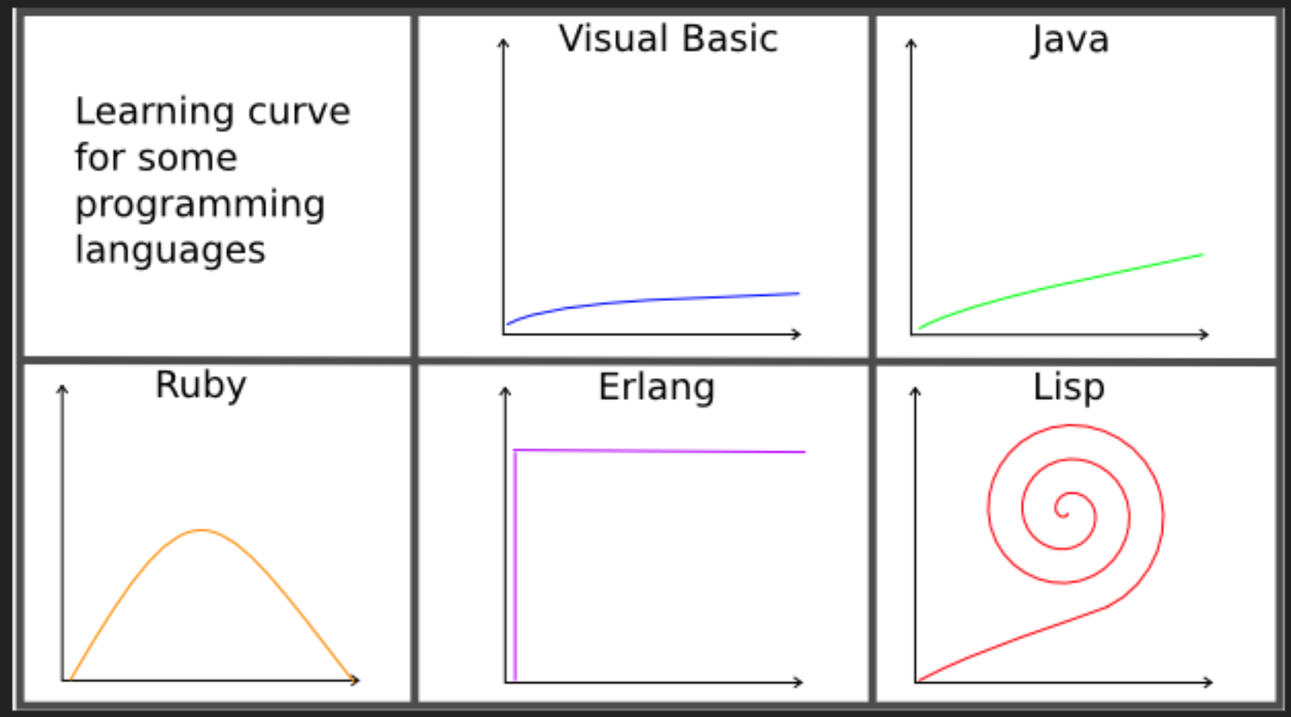
Crisis

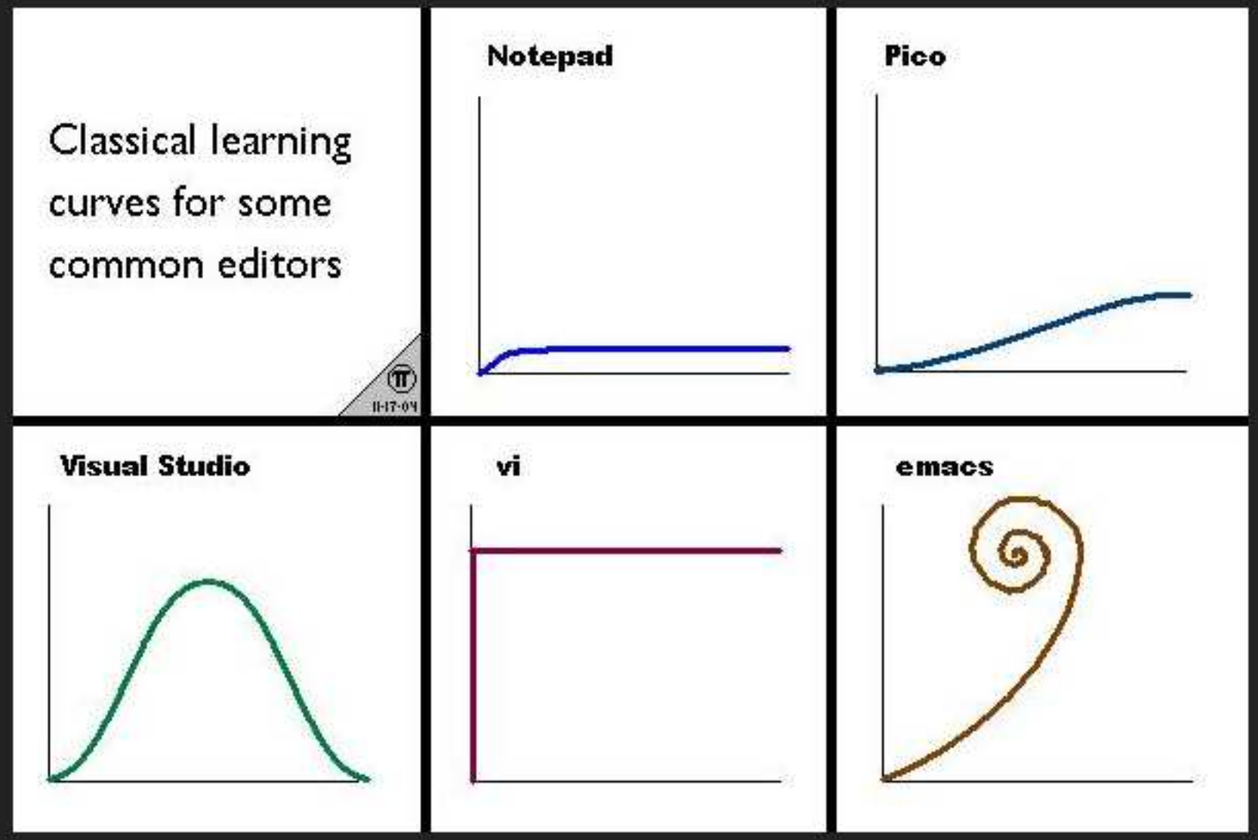
Manycore

Pain of adoption

*A lot of people complain about the Erlang's syntax when they first start using it -- **deal with it**. It is complicated and seemingly convoluted, but the more you write it the more natural it becomes. It **will take a while** for any of it **to make sense***

Chris Williams

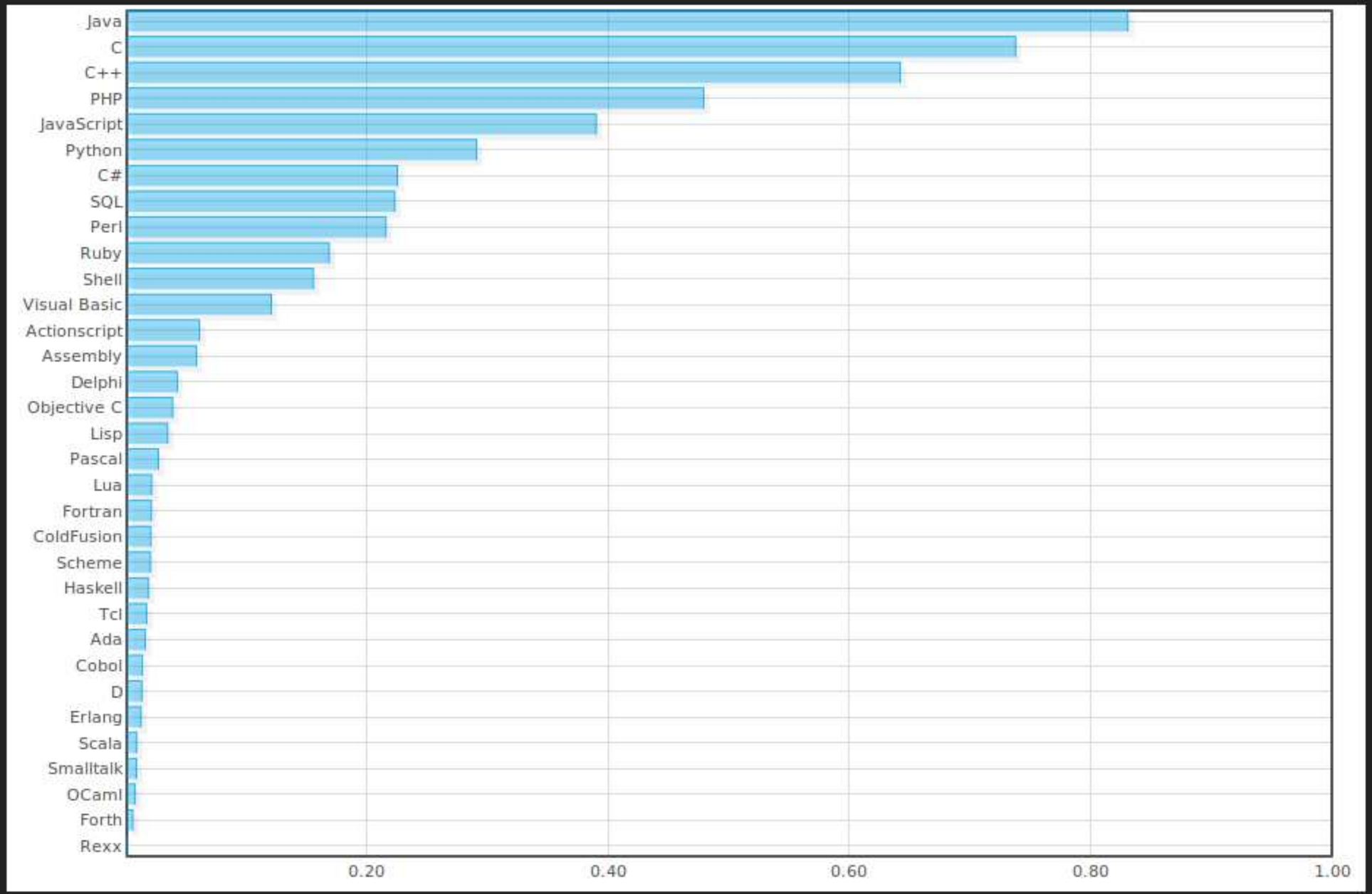

Learning curve for some programming languages
Visual Basic
Java
Ruby
Erlang
Lisp



http://unix.rulez.org/~calver/pictures/curves.jpg


come to the dark side
we have cookies


C is for Cookie



Position May 2010	Position May 2009	Delta in Position	Programming Language	Ratings May 2010	Delta May 2009	Status
1	2	↑	C	18.186%	+2.06%	A
2	1	↓	Java	17.957%	-1.58%	A
3	3	=	C++	10.378%	-0.69%	A
4	4	=	PHP	9.073%	-0.85%	A
5	5	=	(Visual) Basic	5.656%	-2.97%	A
6	7	↑	C#	4.779%	+0.51%	A
7	6	↓	Python	4.097%	-1.45%	A
8	9	↑	Perl	3.286%	-0.24%	A
9	11	↑↑	Delphi	2.566%	+0.24%	A
10	39	↑↑↑↑↑↑↑↑↑↑	Objective-C	2.363%	+2.23%	A
11	10	↓	Ruby	2.094%	-0.60%	A
12	8	↓↓↓↓	JavaScript	2.084%	-1.46%	A
13	12	↓	PL/SQL	0.859%	-0.24%	A
14	13	↓	SAS	0.732%	-0.07%	A
15	14	↓	Pascal	0.728%	-0.05%	A--
16	22	↑↑↑↑↑↑	Lisp/Scheme/Clojure	0.651%	+0.19%	B
17	16	↓	ABAP	0.650%	-0.02%	B
18	-	↑↑↑↑↑↑↑↑↑↑	Go	0.640%	+0.64%	A-
19	18	↓	MATLAB	0.612%	+0.09%	B
20	20	=	Lua	0.493%	+0.01%	B

C, C++, C#, Java, JavaScript

JavaScript was designed with Java's syntax and standard library in mind

All Java keywords are reserved in JavaScript

JavaScript's standard library follows Java's naming conventions

JavaScript's Math and Date objects are based on classes from Java 1.0

C → C++ → C#

Basic → Visual Basic → Visual Basic.NET

.?

A language with friendly syntax for people coming from mainstream languages

Code!

```
# when statement
compare_when = fn (A, B) {
    when A < B {
        lt
    }
    else when A > B {
        gt
    }
    else {
        eq
    }
}
```

```erlang
% when statement
compare_when(A, B) ->
    if
        A < B ->
            lt;
        A > B ->
            gt;
        true ->
            eq
end.
```

```
# when statement
compare_when = fn (A, B)
    when A < B
        lt

    else when A > B
        gt

    else
        eq
```

ifene?

I ♥ Python & JS

```
# if statement
compare_if = fn (A, B) {
    if A < B {
        lt
    }
    else if A > B {
        gt
    }
    else {
        eq
    }
}
```

```
# if statement
compare_if = fn (A, B)
    if A < B
        lt

    else if A > B
        gt

    else
        eq
```

```erlang
% if statement
compare_if(A, B) ->
    case A < B of
        true ->
            lt;
        false ->
            case A > B of
                true ->
                    gt;
                false ->
                    eq
            end
end.
```

```
# switch statement and multiline expressions
compare_to_string = fn (Result) {
    switch Result {
        case lt {
            "lower than"
        }
        case gt {
            "greater than"
        }
        case eq {
            "equal to"
        }
        else {
            "invalid value '" ++
                atom_to_list(Result) ++
                "'"
        }
    }
}
```

```
# switch statement and multiline expressions
compare_to_string = fn (Result)
    switch Result
        case lt
            "lower than"

        case gt
            "greater than"

        case eq
            "equal to"

        else
            "invalid value '" ++
                atom_to_list(Result) ++
                "'"
```

```erlang
% switch statement and multiline expressions
compare_to_string(Result) ->
    case Result of
        lt ->
            "lower than";

        gt ->
            "greater than";

        eq ->
            "equal to";

        _ ->
            "invalid value '" ++
                atom_to_list(Result) ++
                "'"
    end.
```

```
# try/catch expression and tuples
fail = fn (Fun) {
    try {
        Fun()
    }
    catch error Error {
        ("error", Error)
    }
    catch throw Throw {
        ("throw", Throw)
    }
    catch Type Desc {
        (atom_to_list(Type), Desc)
    }
}
```

```
# try/catch expression and tuples
fail = fn (Fun)
    try
        Fun()

    catch error Error
        ("error", Error)

    catch throw Throw
        ("throw", Throw)

    catch Type Desc
        (atom_to_list(Type), Desc)
```

```erlang
% try/catch expression and tuples
fail(Fun) ->
    try
        Fun()
    catch
        error:Error ->
            ("error", Error);
        throw:Throw ->
            ("throw", Throw);

        Type:Desc ->
            (atom_to_list(Type), Desc)
end.
```

```
# multiple function definition and guards
compare_to_string_guards = fn (Result) when Result == lt {
    "lower than"
}
fn (Result) when Result == gt {
    "greater than"
}
fn (Result) when Result == eq {
    "equal to"
}
fn (Result) {
    "invalid value '" ++
        atom_to_list(Result) ++
        "'"
}
```

```
# multiple function definition and guards
compare_to_string_guards = fn (Result) when Result == lt
    "lower than"

fn (Result) when Result == gt
    "greater than"

fn (Result) when Result == eq
    "equal to"

fn (Result)
    "invalid value '" ++
        atom_to_list(Result) ++
        "'"
```

```erlang
% multiple function definition and guards
compare_to_string_guards(Result) when Result == lt ->
    "lower than";

compare_to_string_guards(Result) when Result == gt ->
    "greater than";

compare_to_string_guards (Result) when Result == eq ->
    "equal to";

compare_to_string_guards(Result) ->
    "invalid value '" ++
        atom_to_list(Result) ++
        "'".
```

Extra

```erlang
@public
run = fn ()
    R0 = for X in lists.seq(1, 10)
        A = X + 1
        A

    R1 = for X in lists.seq(1, 10) if X % 2 == 0
        A = X + 1
        A

    R2 = for X in lists.seq(1, 5)
        for Y in lists.seq(6, 10)
            (X, Y)

    R2A = [(X, Y) for X in lists.seq(1, 5) for Y in lists.seq(6, 10

    R3 = for (X, Y) in lists.zip(lists.seq(1, 3), lists.seq(4, 6))
        (Y, X)
```

```
person = object(firstname, lastname, mail)

@public
run = fn ()
    # helper function
    Print = fn (X) { io.format("~p~n", [X]) }
    # create an "object"
    P = person("mariano", "guerra", "mail")
    # get firstname
    Print(P(get, firstname))
    # get lastname
    Print(P(get, lastname))
    # get the "object" as an erlang record
    Print(P(to, rec))
    # get the fields of the "object"
    Print(P(to, fields))
    # get the name of the "object"
    Print(P(to, name))

    # check if the "object" has an attr called firstname
    Print(P(has, firstname))
    # check if the "object" has an attr called address
    Print(P(has, address))

    # create a new "object" changing the firstname attribute
    P1 = P(setfirstname, "Mariano")
    # build a new person from the record of another one
    P2 = person(P1(to, rec))
    R = P2(to, rec)
    Print(person.R[firstname])
```

Ideas behind efene

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

DRY

```
-module(name).
-export([foo/1]).

% lot of code here ...

% case 1: is it public?
% case 2: you want to make it public to test it
% case 3: you want to make it private
% case 4: change the name of the function
```

```
% go to the top
% check if it's exported
% write the name (again) and write the arity
% to rename: change the name in multiple places
% come back here again

foo(<pattern1>) ->
    <body1>;
foo(<pattern2>) ->
    <body2>;
foo(<pattern3>) ->
    <body3>.
```

```
@public
foo = fn (<pattern1>)
    <body1>
fn (<pattern2>)
    <body2>
fn (<pattern3>)
    <body3>
```

Tests

Stability

Documentation

What sucks about Erlang

Damien Katz - http://damienkatz.net/2008/03/what_sucks_abou.html

Erlang is based originally on Prolog, a logic programming language that was briefly hot in the 80's. Surely you've seen other languages based on Prolog, right? No? Why not? Because Prolog sucks ass for building entire applications. But that hasn't deterred Erlang from stealing it's dynamite syntax.

Damien Katz

Erlang is amazing in ways it would take a whole book to describe properly. It's not a toy built to satisfy the urges of academics, it's used in successful, real world products.

Damien Katz

Erlang **rules**

```
f(X) ->
  X1 = foo(X),
  X2 = fab(X1),
  X3 = bar(X2),
  baz(X3).
```

```
f = fn(X)
  X->foo()->fab()->bar()->baz()
```

fnc program.fn
fn program function

Technical Stuff

leex, yecc

lexer → post lexer → parser →
post parser → compiler

lexer
→ leex
post lexer
→ normalize tokens

parser
→ yecc
post parser
→ attributes, @public, -module, -export
compiler
→ compile:forms

post lexer

→ fnc -t lex file.fn

parser

→ fnc -t tree file.fn

post parser

→ fnc -t ast file.fn

→ fnc -t mod file.fn

compiler

→ fnc -t beam file.fn

→ fnc file.fn

```
fnc -t erl file.fn
fnc -t fn file.fn
fnc -t ifn file.fn
fnc -c 'expression'
fnc -C 'expression'
fnc -s
```

Future

Spec

Help!

→ Download it
→ Test it
→ Report bugs
→ Spread the word

Thanks

Questions?