

# Things that Matter

*Decisions that Shape Languages*

The  
Pragmatic  
Programmers

# Seven Languages in Seven Weeks

A Pragmatic  
Guide to  
Learning  
Programming  
Languages

Bruce A. Tate

*Edited by Jacquelyn Carter*



The  
Pragmatic  
Programmers

# Seven Languages in Seven Weeks

A Pragmatic  
Guide to  
Learning  
Programming  
Languages

Bruce A. Tate

*Edited by Jacquelyn Carter*



The  
Pragmatic  
Programmers

# Seven More Languages in Seven Weeks

Languages That Are  
Shaping the Future



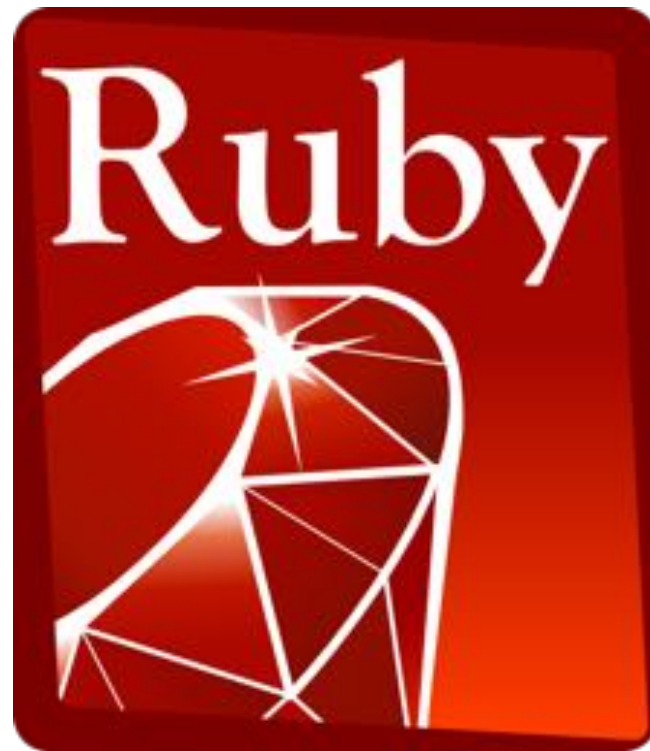
Bruce A. Tate, Fred Daoud,  
Ian Dees, and Jack Moffitt

Foreword by José Valim

*Edited by Jacquelyn Carter*



# the incubator







“The primary motivation was to amuse myself.”



“I like the way it makes programming enjoyable.”

– Matz









“So, we started Lua with the very specific goal of providing a language for problems

”

- *Roberto Ierusalimschy*



“So, we started Lua with the very specific goal of providing a language for problems that need a **good configuration language.**”

*- Roberto Ierusalimsky*



elm



The best of functional programming in your browser



“Many functional folks

”





“Many functional folks have a way of saying  
**extremely interesting** and useful things

”



“Many functional folks have a way of saying extremely interesting and useful things in a totally **inaccessible impractical** way,”



“Many functional folks have a way of saying extremely interesting and useful things in a totally inaccessible impractical way, and **I wanted to fix this.**”



elm



“Elm is not about being theoretically better. It is about being **demonstrably better.**”

– Evan Czaplicki









“On the lazy side (of FP), you had as many programming languages as there were researchers.”



“If he (David Turner, Miranda) had said yes (to making Miranda the single standard for research of lazy FP), Haskell would not exist.”

– John Hughes





Clojure



Clojure

“I wanted

”





Clojure

“I wanted a Lisp

”



# Clojure

“I wanted a Lisp **for Functional Programming,**

”



# Clojure

“I wanted a Lisp for Functional Programming, **symbiotic with an established Platform,**”



# Clojure

“I wanted a Lisp for Functional Programming, symbiotic with an established Platform, and **designed for Concurrency.**”

– Rich Hickey





“A simple,





“A simple, **object-oriented**,



“A simple, object-oriented, **distributed**,



“A simple, object-oriented, distributed, **interpreted**,



“A simple, object-oriented, distributed, interpreted,  
**robust**,



“A simple, object-oriented, distributed, interpreted, robust, **secure**,



“A simple, object-oriented, distributed, interpreted, robust, secure, **architecture neutral**,



“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, **portable**,



“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, **high-performance**,





“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, **multithreaded**,



“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and **dynamic** language.”




?





# the incubator





Your origins  
shape you





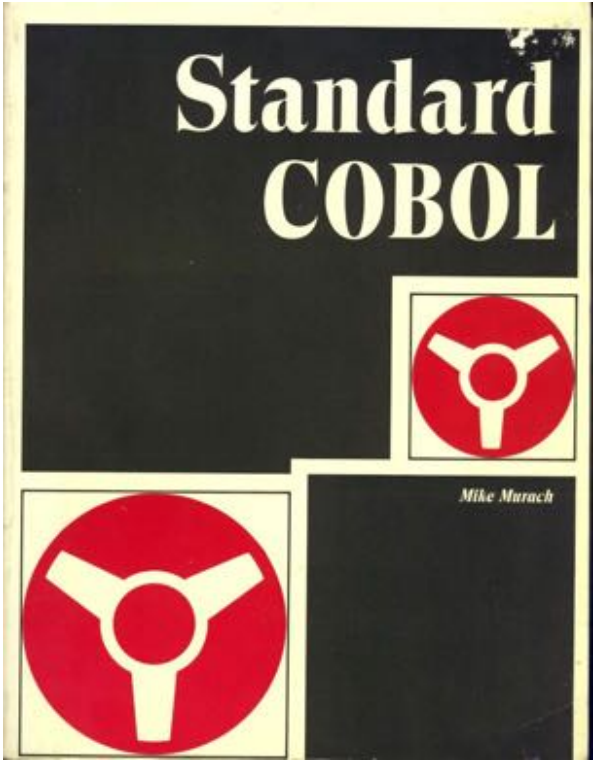
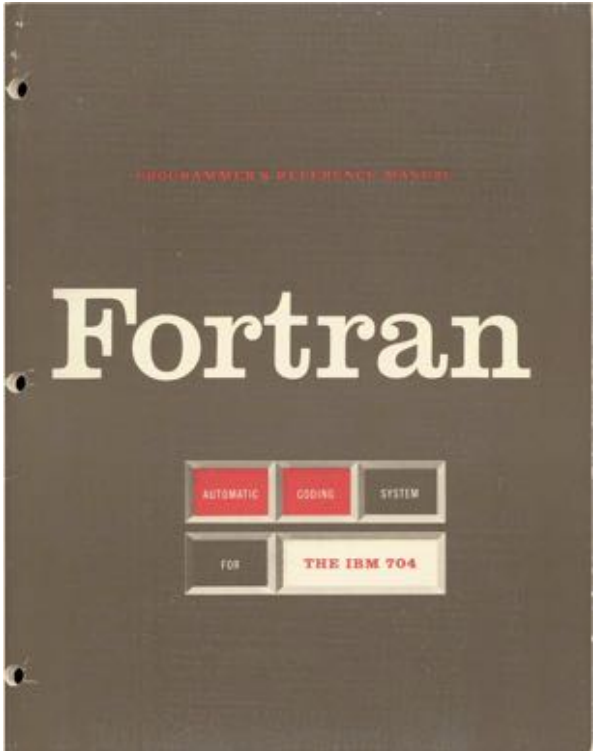
Know your  
community

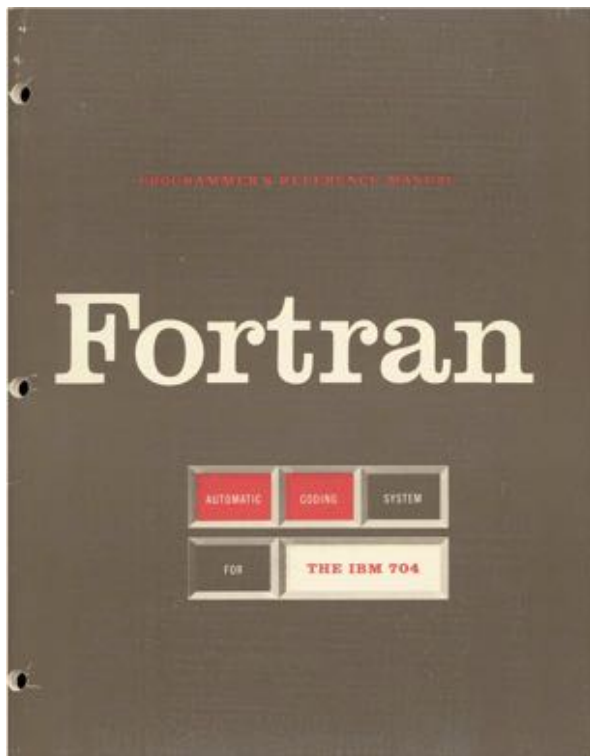




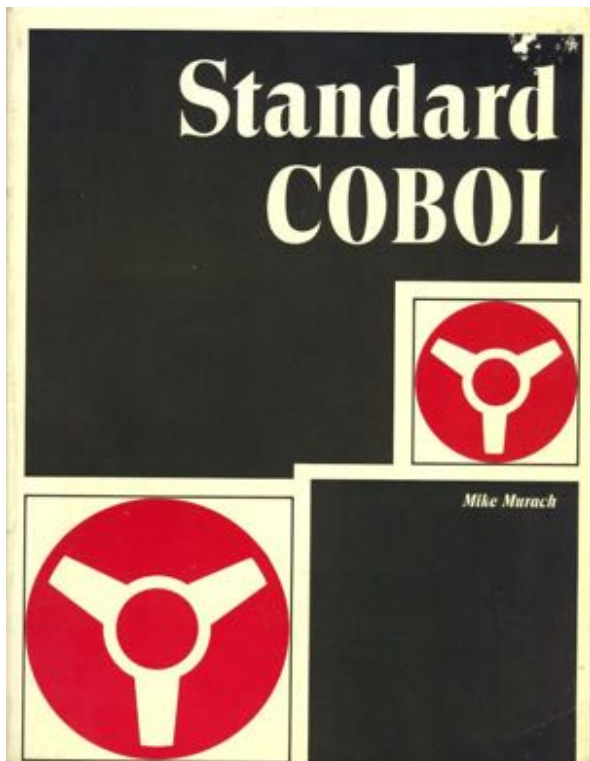








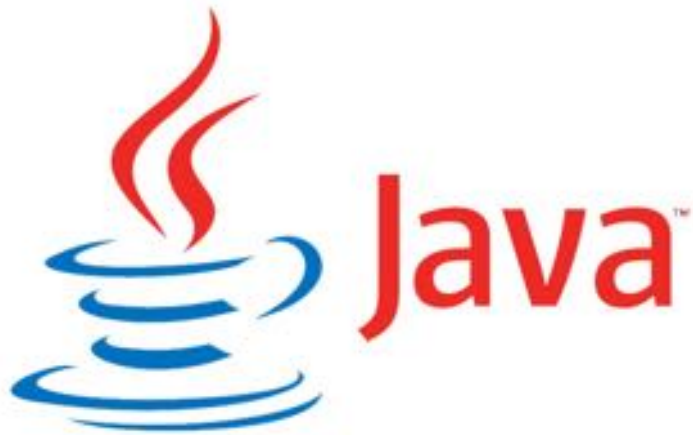
- *Population*
- *Support*
- *Investment*







- *Population*
- *Support*
- *Investment*



- *Population*
- *Community*
- *Investment*





Clojure



elixir



python™



elm



Scala



Idris



Clojure



elixir



elm



Scala



Idris

- *Population*
- *Support*
- *Investment*



julia



python™





elm

```

import Graphics.Gloss exposing (...)
import Mouse
import Window

main = Signal Element
main =
  Signal.map2 view Mouse.position Window.dimensions

view = (Int,Int) -> (Int,Int) -> Element
view (x,y) (w,h) =
  pieChart (List.map toFloat [x,y,w,h])

pieChart :: List Float -> Element
pieChart numbers =
  let fracs = normalize numbers
      offsets = List.scanl (-) 0 fracs
  in
    collage 300 300 <|
      List.concat (List.map (pieSlice 100) colors offsets fracs)
      ++ [ filled white (circle 70) ]

pieSlice :: Float -> Color -> Float -> Float -> List Path
pieSlice radius color offset angle =
  let makePoint t = fromPolar (radius, degrees (360 + offset + t))
  in
    [ filled color <| polygon [(0,0) :> List.map makePoint [ 0 .. 360 + angle ]],
      stroke (asPercent angle)
        |> move (fromPolar (radius*1.25, turn (offset + angle/2)))
    ]

asPercent :: Float -> Element
asPercent fraction =
  show (toString (toFloat (truncate (fraction + 100)))) ++ "%"

colors :: List Color
colors =
  [ lightBlue, lightGreen, lightYellow, lightRed
  , lightPurple, blue, green, yellow, red, purple
  ]

normalize :: List Float -> List Float
normalize xs =
  let total = List.sum xs
  in
    List.map (\x -> x/total) xs

```



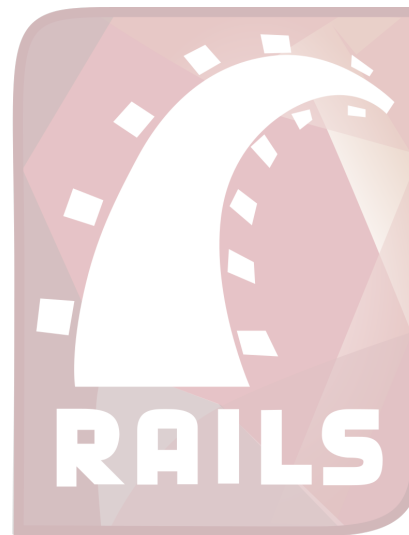
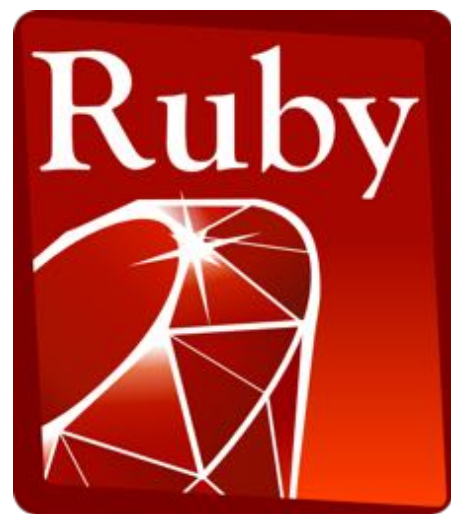
Hint: [syntax](#) for defining values, pronounced "syntax"

Hint: [Options](#)

Auto-update

Hot Swap

Compile





**MINSWAN**



Matz Is Nice So We Are Nice

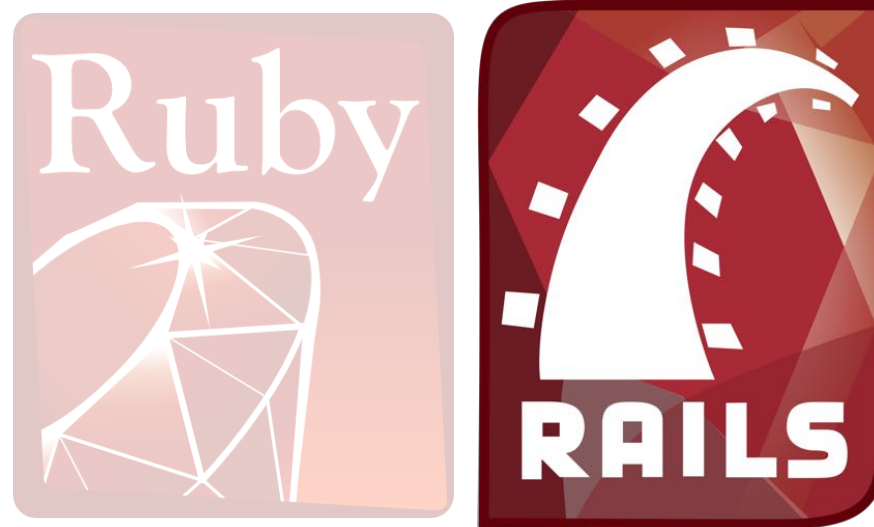




Matz Is Nice So We Are Nice



**MINSWAN**



Polish

*Erlang*





*Erlang*



elixir





Know your  
community





Make a Stand



SinTax

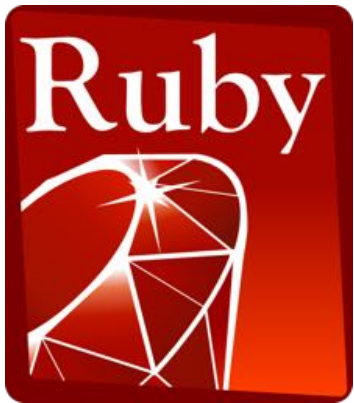
12 OZ.

Syntax has a profound  
impact on

Syntax has a profound  
impact on productivity



# Sugar makes programmers more productive



*“Languages are enhancers for  
your mind that shape the way  
you attack programming.”*

Syntax must be simple

# Syntax must be **simple**



[Smalltalk]



Clojure



Syntax has a profound  
impact on

Syntax has a profound  
impact on program design

Syntax must be profoundly  
simple and uniform

Syntax must be profoundly  
simple and uniform

Io



*The tree*  
Syntax must be profoundly  
simple and uniform

*The tree*  
Syntax must be profoundly  
simple and uniform



elixir

Syntax has a profound  
impact on

Syntax has a profound  
impact on **marketshare**



Syntax has a profound  
impact on **marketshare**







We will be lazy

Our functions will be pure

Our types are **strict and static**

*Erlang*

Make **concurrency** simple



Let it crash



elm

# Approachable Theory

Callbacks Stink



Make a Stand

Adapt or die.

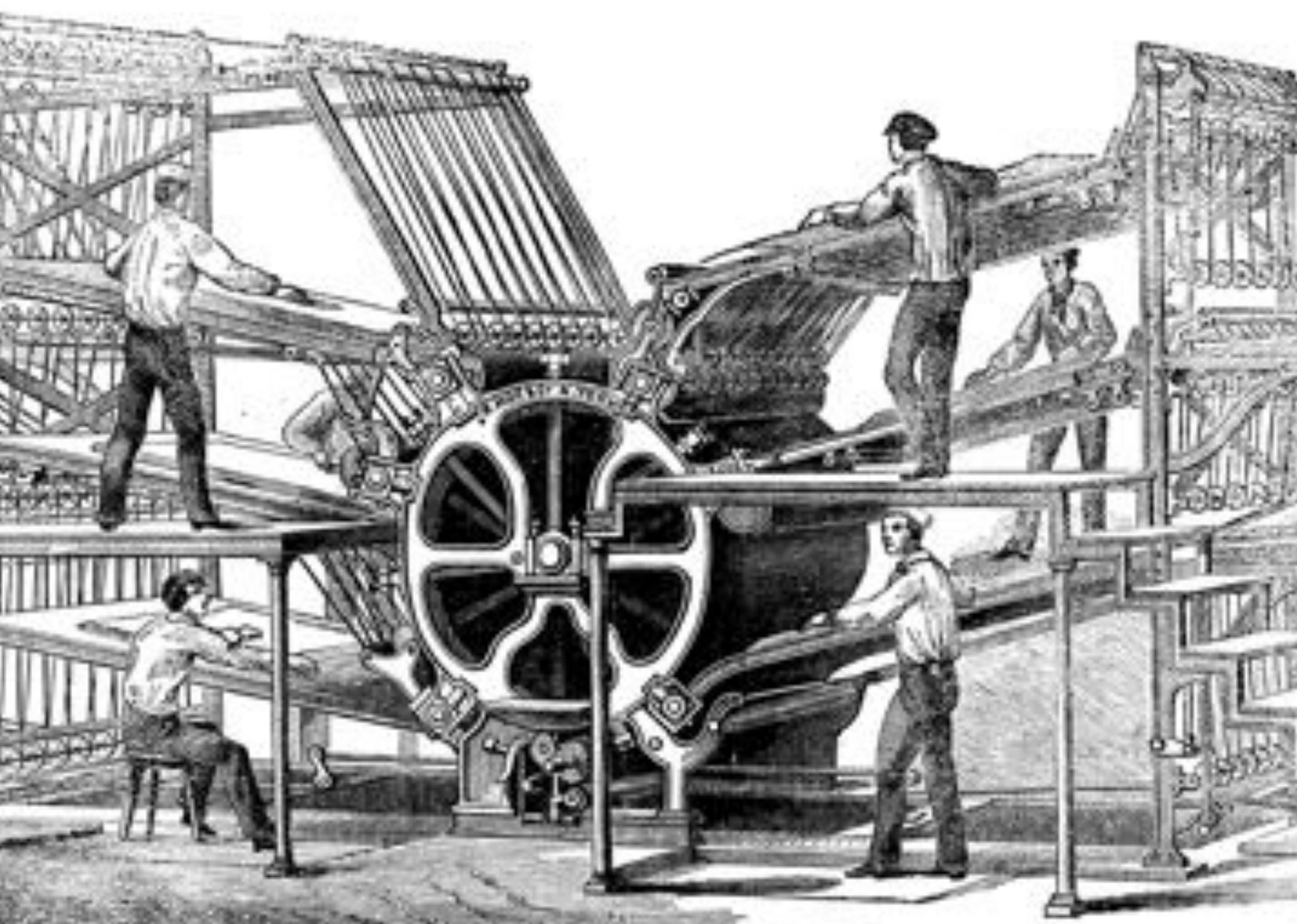




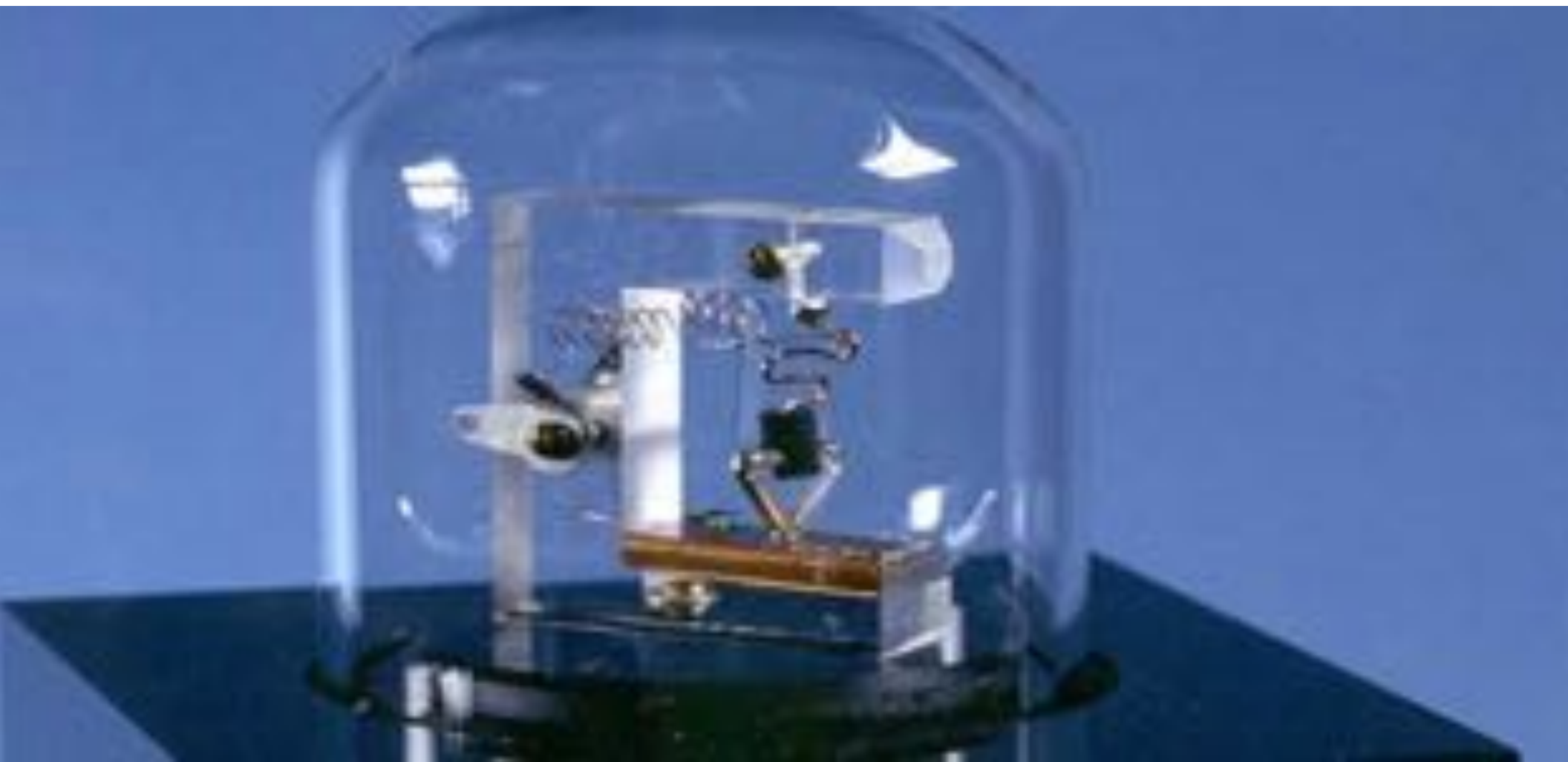












# Efficient Program Design

Idea

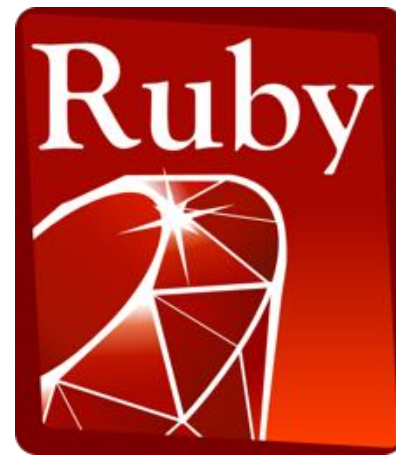
# Idioms

# Abstractions

*Erlang*







```
class String
  def blank?
    self == ""
  end
end
```

```
class NilClass
  def blank?
    true
  end
end
```

```
class Object
  def blank?
    false
  end
end
```

```
[nil, 4, ""].map do |item|
  item.blank?
end
```

# Efficient Program Design

# Efficient ~~Program~~ Design

# Efficient Language Design

# Adaptation

# Extension

16







# Macros



(+ 1 2)

( ' + | 2 )

*Erlang*

```
Top = self,  
Ref = make_ref,  
  
Pid = spawn_link(fun ->  
    Top ! { Ref, ... }  
),  
  
receive  
    { Ref, Value } -> Value  
end
```

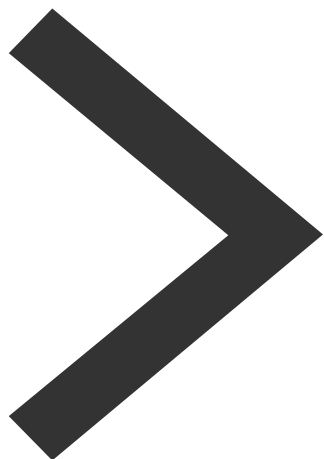


elixir



```
task = Task.async(&do_something/0)  
# do something concurrently  
result = Task.await(task)
```





defmacro

defprotocol

```
widgets  
|> Enum.filter...  
|> Enum.map...  
|> Enum.take(5)
```

```
widgets  
|> Stream.filter...  
|> Stream.map...  
|> Enum.take(5)
```

widgets

|> Stream.expensive1...

|> Stream.expensive2...

|> Enum.take(5)



widgets

|> Stream.expensive1...

|> ?

|> Stream.expensive2...

|> ?

|> Enum.take(5)

widgets

|> Stream.expensive1...

|> async\_process...

|> Stream.expensive2...

|> async\_process...

|> Enum.take(5)

widgets

|> Expensive.task1...

|> ?

|> Expensive.task2...

|> ?

|> Enum.take(5)

```
widgets
```

```
|> Expensive.task1...
```

```
|> process_farm(10)
```

```
|> Expensive.task2...
```

```
|> process_farm(20)
```

```
|> Enum.take(5)
```

```
widgets
```

```
|> Expensive.task1...
```

```
|> distribute(10)
```

```
|> Expensive.task2...
```

```
|> distribute(20)
```

```
|> Enum.take(5)
```

widgets

|> distribute(20)

|> Stream.expensive1...

|> Stream.expensive2...

|> Stream.expensive3...

|> Enum.take(n)

# Other Examples



# Other Examples





# Other Examples



elm



# Other Examples



elm



# Other Examples




elm



Adapt or die





Your origins  
shape you





Know your  
customer





Make a Stand

Adapt or die





?