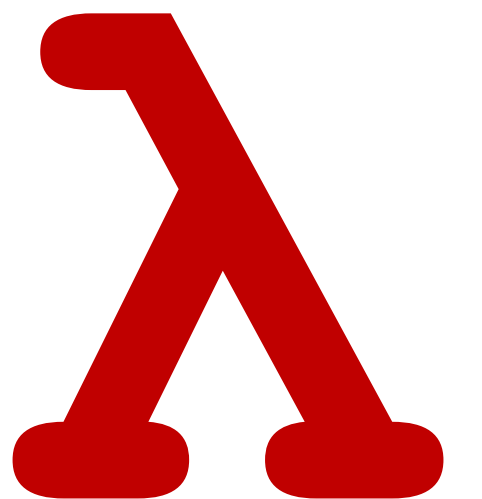# CSCI 275: Programming Abstractions

Lecture 35: Practical Concerns
Spring 2025

Stephen Checkoway
Slides from Molly Q Feldman

Questions?

Functional Language of the Week: Erlang

Started in the *1980s* by the Swedish telecom company Ericsson

Equipped with its own runtime system to help support telecom operations (i.e. fast transactions!)

"Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and **fault tolerance**." from the Erlang homepage

At the *language level* there is support to make sure that if one system component breaks, the code does not stop running

ERLANG

Functional Language of the Week: Erlang

```erlang
%% Immutable variables
> Fruits = ["banana","monkey","jungle"].
```

> Yes, Erlang ends all expressions with periods!

```erlang
%% Map values using stdlib functions
> lists:map(fun string:uppercase/1, Fruits).


%% Fold over lists using custom functions
> lists:foldl(fun(Str, Cnt) ->
string:length(Str) + Cnt end, 0, Fruits).
18
```

https://www.erlang.org/
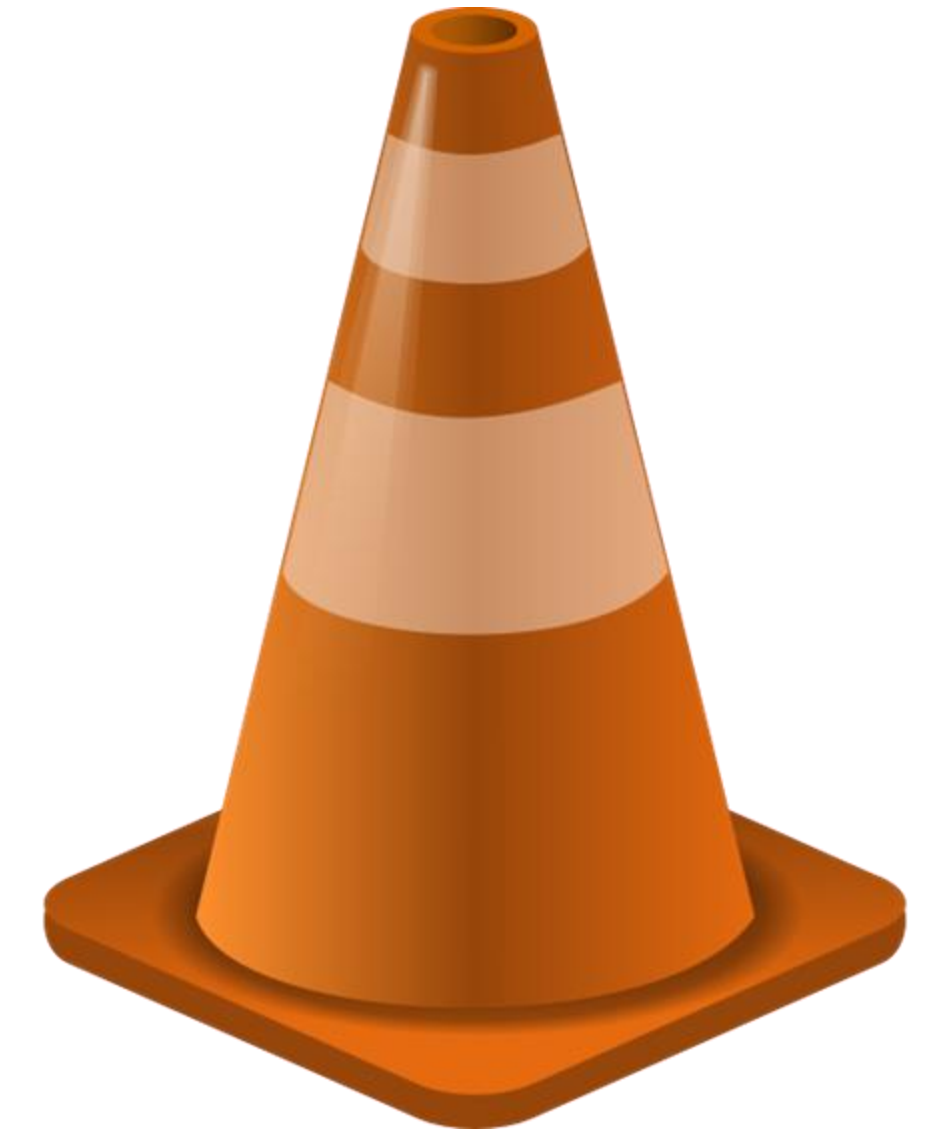
ERLANG

Practical Concerns

Partial answers to three big questions:

(1) What PL should I learn for what task?

(1) Why do we make new ones?

(1) Why do we have the languages that we have?

I'll give you my (and others') opinions on these! Not definitive, but hopefully fun/interesting. Also, I'll hopefully provide some helpful links.

Whirlwind Core Language Tour

Python

- The language we teach in CSCI 150
- #1 Language on the TIOBE Index
- Lots of use cases: scripting, objects, functions, dynamically typed
- OO but has functional parts

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"

    def __repr__(self):
        return f"Person({self.name}, {self.age})"
```

```java
//Basic class definition
public class void main(String[] args) {
    //Get the input from the user
    Scanner input = new Scanner(System.in);
    System.out.println("Enter the number of rows and columns in the array: ");
    int row = input.nextInt();
    int column = input.nextInt();
    //Create the array
    double[][] array = new double[row][column];
    //Get the values from the user
    System.out.println("Enter the array: ");
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array[i].length; j++) {
            array[i][j] = input.nextDouble();
        }
    }
}
```

Java

- The language we teach in CSCI 151
- #4. Language on the TIOBE Index
- Statically typed, object-oriented, *cross-platform JVM*

JavaScript

- Not part of our standing curriculum!
- #6 Language on the TIOBE Index
- Dynamically typed, event-based, powers most of the internet

```javascript
// Basic GET/POST request
var request = require('request');
function get(url, callback) {
    request.get(url, function (error, response, body) {
        if (!error && response.statusCode == 200) {
            callback(body);
        }
    });
}
                function post(url, data, callback) {
                    request.post(url, {form: data}, function (error, response, body) {
                        if (!error && response.statusCode == 200) {
                            callback(body);
                        }
                    });
                }
```

Which language for which task?

Which of these languages is your preferred language out of those we teach in the department?

A. Python

B. Java

C. Racket

D. Rust

E. Something else

Think about *why* you like to use those languages

Syntax?

Familiarity?

Use cases?

Libraries?

Ease of use?

Something else?

Prof. Molly's recommendation - based on conversations with other professors & professional developers!

Python

JavaScript (TypeScript)

Pick your OOP of Choice - Java mostly

But there is *no* "one size fits all" answer

Most people would suggest that JavaScript is ***THE*** web programming language to learn.

Yet what about types? **TypeScript.**

Shopify (powers a huge number of web transactions) and **Twitch** are all written in **Ruby on Rails**

Rust can be compiled to WebAssembly which interoperates with JavaScript

https://w3techs.com/technologies/overview/client_side_language

So, what options do you have when choosing a language?

Option #1: What you know!

Sometimes, for your own personal projects or for your own small applications, just **using the language you know** is probably the best idea!

Prof. Molly's Personal Anecdote: My first programming language was Python and it's the language I "reach for" the most

Steve's: Python was my 6th programming language and also the one I choose first: right up until it becomes complex and then I want types so I use Rust

Option #2: Application

You can use the general **type** of what you're building to help!

Writing a quick script?
Maybe use Python or Bash!

Building a website?
Maybe use JavaScript!

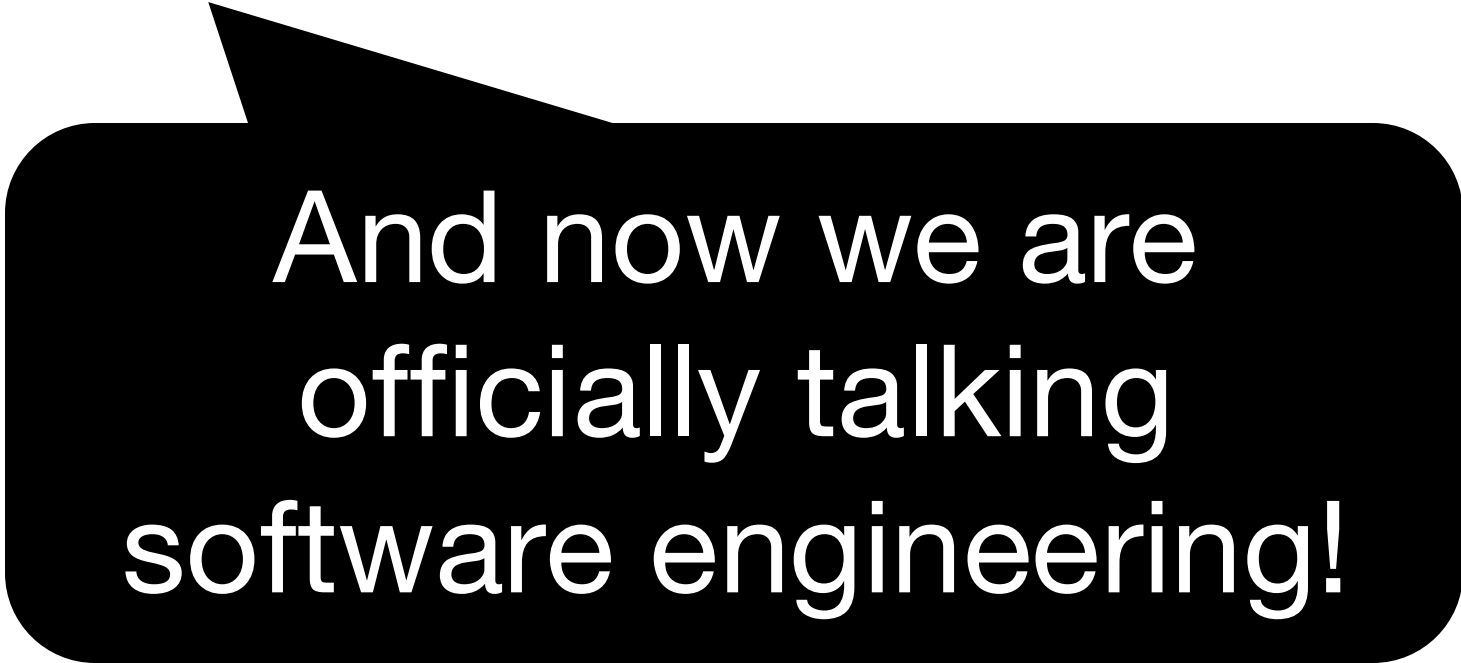Writing something that needs to be *fast*? Maybe use Rust or C!

Really specific application? Make your own domain-specific language!

Please don't use C if you can help it

Aside: Libraries Are Great, Use Them In the Real World

A lot of code you will/want to write *is not new,* in that someone has written it before!

Sometimes finding a good library / API / etc. can make the difference between picking Language A versus Language B

And now we are officially talking software engineering!

Aside to the aside: Standard library is key

Every language comes with a standard library of functions, some are better than others

C's standard library is small, barely more than wrappers around system calls
- E.g., doesn't have any standard data structures like growable arrays (vectors), maps (dictionaries), trees, or hashtables. You have to implement them yourself or use a 3rd party library

3rd party libraries can be quite difficult to use in some languages like C

Option #3: Paradigm

Programming paradigms are the (loose) categories that different languages can fall under. Some (common) options include:

- Functional
- Object-Oriented
- Logic
- Imperative

Racket

Java

Prolog

Python

Paradigms Can Cause Unproductive Boundaries

What paradigm is Python *really*?

"Languages do not organize into hierarchical taxonomies the way plants and animals do; **they are artificial entities that can freely be bred across supposed boundaries.** Language authors can pick from several different bins when creating languages, and indeed modern mainstream languages are **usually a mélange of many of these bins.**" – Krishnamurthi & Fisler

For an interesting perspective on the impact this can have in education:
https://cs.brown.edu/~sk/Publications/Papers/Published/kf-prog-paradigms-and-beyond/paper.pdf

Option #4: Look around you

- Look at what other companies, engineers, etc. are doing in your particular area!

- I see this with *frameworks* a lot and "X as a Service" technologies (https://github.blog/2023-11-08-the-state-of-open-source-and-ai/#the-most-popular-programming-languages)

- Job Ads, Github statistics, TIOBE, social, etc.

https://www.technologyreview.com/2015/04/02/168469/toolkits-for-the-mind/

Have you learned a programming language *outside* of the classroom?

A. Yes, as part of a job or internship

B. Yes, for fun!

C. Both A & B

D. Not yet!

Option #5: You don't get to choose!

Learning programming languages is a skill that gets easier over time

Many folks do not know a certain language (but *do know CS fundamentals*!) and learn a new language on the job

The more kinds of languages you have exposure to, the easier to learn a new language in my opinion

Why it's good to learn Racket!

Aside: Real World Applications of Racket?

The Scribble Documentation Tool: https://docs.racket-lang.org/scribble/

Racket Con features talks from a variety of speakers over the years! https://con.racket-lang.org/

Former topics are:

- Academic projects on Types

- Web programming

- Game programming

Why do we make more languages?

Why do **you** think we make more languages?

A lot of our "Functional Programming Languages of the Week" have been "new" languages

What narratives do they share in common?

- X + Y = Z and Z is therefore cooler/more usable/etc.

- A "new" version of an older language

- Simply adding functional features to existing languages

Note there is a difference between versions of a language and an entirely new language – this line is blurry though (e.g. Racket was a Scheme variant until it wasn't)

https://www2.ccs.neu.edu/racket/pubs/manifesto.pdf

BASIC – "Beginner's All Purpose Symbolic Instruction Code" (1964)

Initiative to help non-STEM students learn to program from Dartmouth College in 1964

- BASIC was designed in the context of time-sharing mainframes

"I tried, briefly, to develop simple subsets of Fortran and ALGOL, but found quickly that such could not be done" (Kurtz in Time)

```
10 PRINT "Hello, World!"
20 GOTO 10
```

Harry McCraken. "Fifty Years of BASIC, the Programming Language That Made Computers Personal". Time. April 29, 2014.

Pascal (1971)

"The development of the language Pascal is based on two principal aims. The first is to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language. The second is to develop **implementations of this language which are both reliable and efficient on presently available computers, dispelling the commonly accepted notion that useful languages must be either slow to compile or slow to execute**, and the belief that any nontrivial system is bound to contain mistakes forever."

Wirth, N. (1971). The programming language Pascal.
Acta informatica, 1(1), 35-63.

```
program Hello;
begin
    writeln ('Hello, World!')
end.
```

Rust

Started as a hobby project taking good features from older languages

Rust changed over time to simplify the language and became a great systems programming language (fast, strong type system, great memory safety without garbage collection)

Network Switch Languages

In the last decade, network switches – the devices which receive and forward internet packets – have moved from being *hardware configurable* to *software configurable*

This gave rise to programming language to control the switches

Why new languages?

Domain-specific languages let us write code focused on the problem domain (here: network packet routing)

Switches to make decisions about what to do with incoming network packets *quickly*

A language restricted to this task can produce software or hardware (e.g., programming FPGAs) to route packets efficiently

https://github.com/ethereum/solidity

P4 Open Source Programming Language

Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets.

Before P4, vendors had total control over the functionality supported in the network. And since networking silicon determined much of the possible behavior, silicon vendors controlled the rollout of new features (e.g., VXLAN), and rollouts took years.

P4 turns the traditional model on its head. Application developers and network engineers can now use P4 to implement specific behavior in the network, and changes can be made in minutes instead of years.

https://p4.org/

Language Design is like Puzzle Pieces

Language designers tend to be balancing many different factors:

- Facilitating adoption (keywords, curly braces, etc.)

- Paradigm or "underlying functionality"

- Their motivation

    - Application
    - Feature (Research or what not)
    - Other need
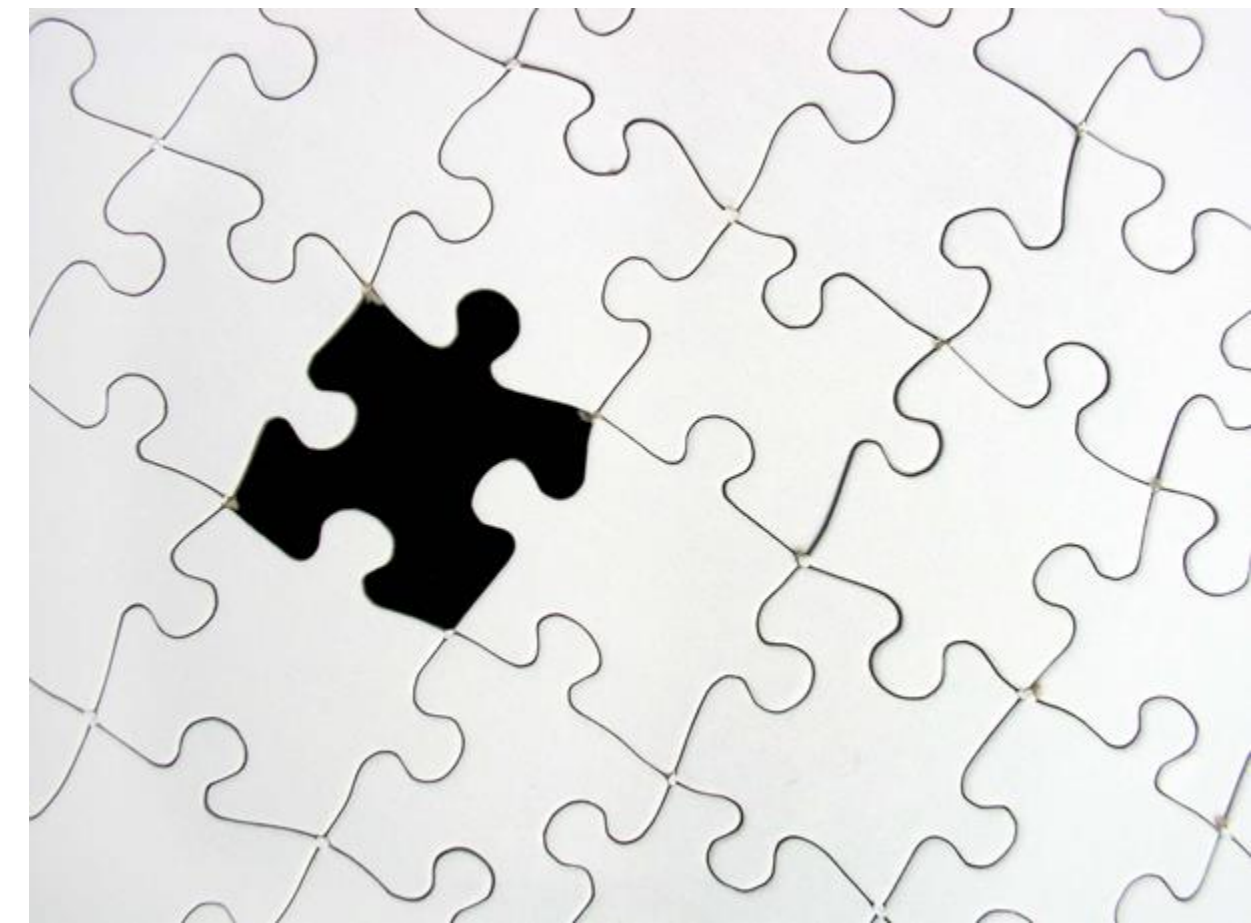
Esoteric programming languages

Odd languages not really designed for use

Some are silly, others impenetrable

Some resemble theoretical models of computation

Some aim for simplicity of the language (usually at the expense of making programming complex)

Shakespeare Programming Language

Do Not Adieu, a play in two acts.

Romeo, a young man with a remarkable patience.

Juliet, a likewise young woman of remarkable grace.

Ophelia, a remarkable woman much in dispute with Hamlet.

Hamlet, the flatterer of Andersen Insulting A/S.

Act I: Hamlet's insults and flattery.

Scene I: The insulting of Romeo.

[Enter Hamlet and Romeo]

Hamlet:
   You lying stupid fatherless big smelly half-witted coward! You
   are as stupid as the difference between a handsome rich brave
   hero and thyself! Speak your mind!

…

Declares variables Romeo, Juliet, Ophelia, and Hamlet. Descriptions are ignored

Labels for conditionals and gotos. E.g.,
Juliet:
   Am I better than you?
Hamlet:
   If so, let us proceed to scene II.

Brings variables to the stage to be acted upon by characters' lines

Hamlet is speaking to Romeo so this is assigning a value to Hamlet based on the combination of nouns and adjectives

Speak your mind! means output a character

Unlambda

Based on SKI combinator calculus which is closely related to lambda calculus with additional I/O and explicit function application operator

Substitution: S x y z = x z (y z)   [In lambda calc: λ x. λy. λz. xz(yz)]

Constant functions: K x y = x   [λx. λy. x]

Identity: I x = x   [λx. x]

Unlambda example:

Fibonacci numbers output as *

```
```s``s``sii`ki
  `k.*``s``s`ks
 ``s`k`s`ks``s``s`ks``s`k`s`kr``s`k`sikk
  `k``s`ksk
```

```
*
*
**
***
*****
*******
************
********************
*********************************
*******************************************************
...
```

Piet

Hello world program

Programs are 20-color bitmaps

Variables are stored on a stack

The instruction pointer moves
 through colored regions

Changing regions performs
 functions

Ook!

Three syntax elements: `Ook.` `Ook?` and `Ook!`

Pairs of syntax elements form 8 commands

`Ook.` `Ook?` Move the memory pointer to the next cell

`Ook?` `Ook.` Move the memory pointer to the previous cell

`Ook.` `Ook.` Increment the value in current cell

`Ook!` `Ook!` Decrement the value in the current cell

`Ook.` `Ook!` Read a character from stdin into the current cell

`Ook!` `Ook.` Print the character in the current cell

`Ook?` `Ook!` Move to the following matching `Ook?` `Ook?` if current cell is nonzero

`Ook?` `Ook?` Move to the previous matching `Ook?` `Ook!`

Single instruction set computer

Related to esoteric programming languages, we have esoteric instruction set architectures (take CSCI 210 to learn about ISAs!)

Single instruction set computers are ones that have a single (usually complex) instruction

E.g., subtract and branch if less than or equal to zero

Subtract and branch if less than or equal to zero (subleq)

`subleq a, b, c` is the only instruction which has the behavior
```
Mem[b] = Mem[b] – Mem[a]
if Mem[b] ≤ 0 then
    goto c
```

If the third operand is omitted, it is implicitly the address of the following instruction

The operands a, b, and c are memory addresses

Can perform any computation, e.g., to implement `add a, b` we have
```
subleq a, Z
subleq Z, b
subleq Z, Z
```

Z = Z – a (storing -a into Z)
b = b – Z (storing b – -a into b)
Z = Z – Z (storing 0 into Z)

where Z is the address of a 0 value in memory