EECS 368
Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 5 due: 11:59 PM, Monday, October 31
  - One of our SIs, Soujanya, has come up with a great way for you to create, test, and submit Assignment 5.
  - Please review the StudentVideoDemo at the following link for more information:
  - https://drive.google.com/drive/folders/1n1R5b3YihQcbCVyQwUjVBD1IMGMdEvBE
  - If you have any question, please contact Soujanya Ambati at: saisoujanyaambati@ku.edu
  - Soujanya will also be grading Assignment 5.
- Assignment 6 due: 11:59 PM, Monday, November 14

Any Questions?

In-Class Problem Solution

- 24-(10-21) In-Class Problem Solution.pptx

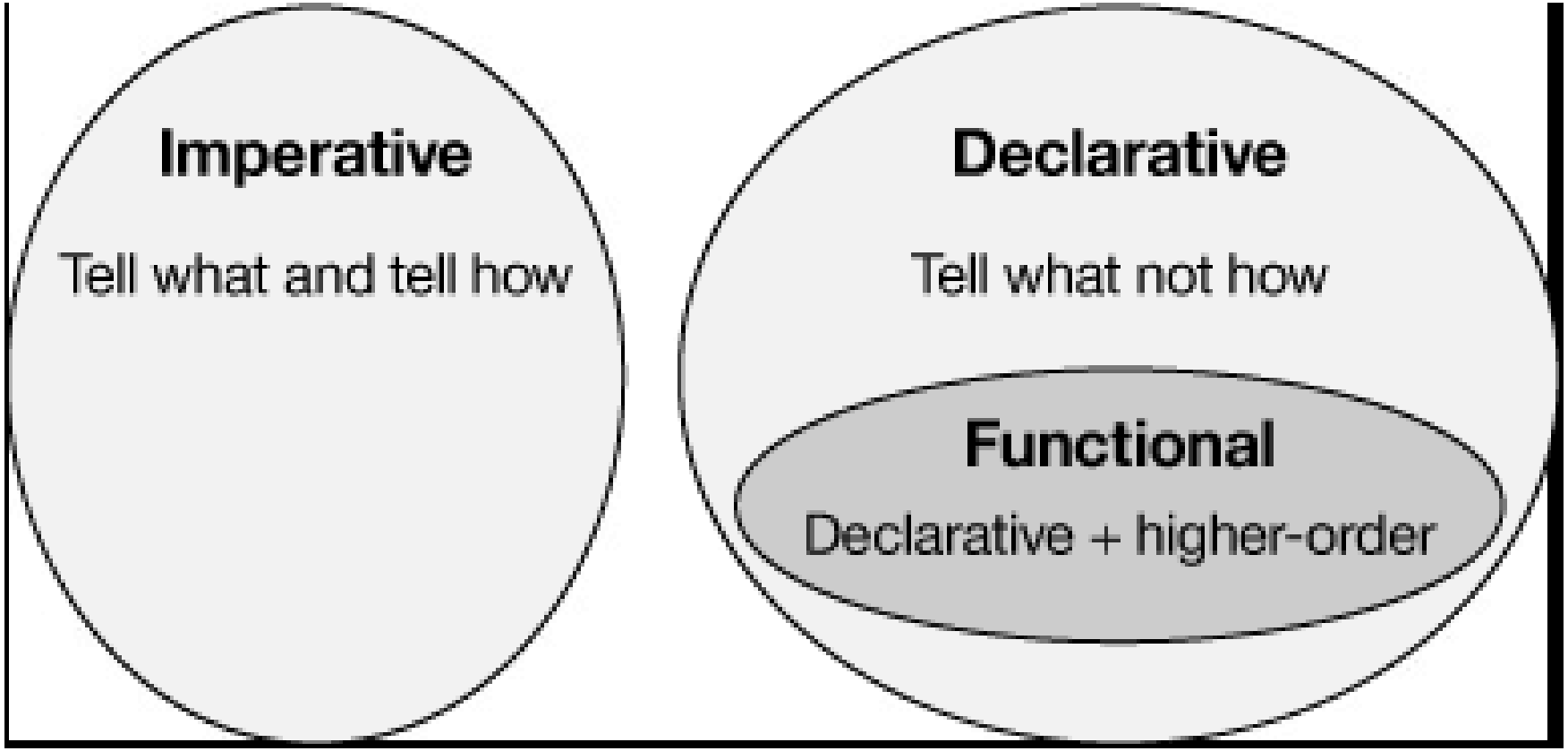
Any Questions?

Imperative vs. Declarative Programming Languages

Recall:

	Imperative	Declarative
Programming Style	Perform step-by-step tasks and manage changes in state	Define what the problem is and what data transformations are needed to achieve the solution
Task	Tell the computer what to do	Describe what you want as an end result
Programmer Focus	how to execute	what to execute
Primary flow control	Iterations, loops, conditionals, and function/method calls	Function calls (including recursion)
Primary manipulation unit	Instances of class or structures	Function as first-class object and data collections
Notable language (for contrasting the style)	Assembly language, C++, Python, JavaScript	SQL, Haskell, Erlang
Real life example	*Go two blocks, make a right turn, proceed for three blocks, arrive at airport*	*Go to nearest airport, please*

Imperative vs. Declarative Programming Languages



EECS 368

- We are going to look at two languages in detail
  - ~~JavaScript (Imperative)~~
  - Haskell (Declarative – Functional)
- ~~We will also look at Cloud Computing from a language point of view.~~
- We will study some DSLs:
  - ~~JavaScript Support Vector Graphics (SVG)~~
  - ~~JavaScript Canvas~~
  - Haskell QuickCheck

Function-Oriented Design Paradigm

- Before we dive into Haskell, we need to back away and see why functions are so important in Haskell.
- They are important in Haskell because it supports the function-oriented design paradigm.

What Is a Design Paradigm?

Recall:

- A paradigm is an example or pattern that that can be copied.
- Paradigms consist of "a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline."
- When applied to software development, a paradigm guides the way that developers view a given problem and organize the solution.

You have learned about two design paradigms:

- Object-oriented design (EECS 168 & 268)
- Function-oriented design (EECS 368)

Function-Oriented vs. Object-Oriented Design

Recall:

Function-Oriented Design:

- The system is designed from a functional viewpoint.
- Function oriented comes from math:

$$y = f(x,y) = x \cdot x + y \cdot y \text{ (sum of squares)}$$

- Style of programming in which the basic method of computation is the application of functions to arguments.

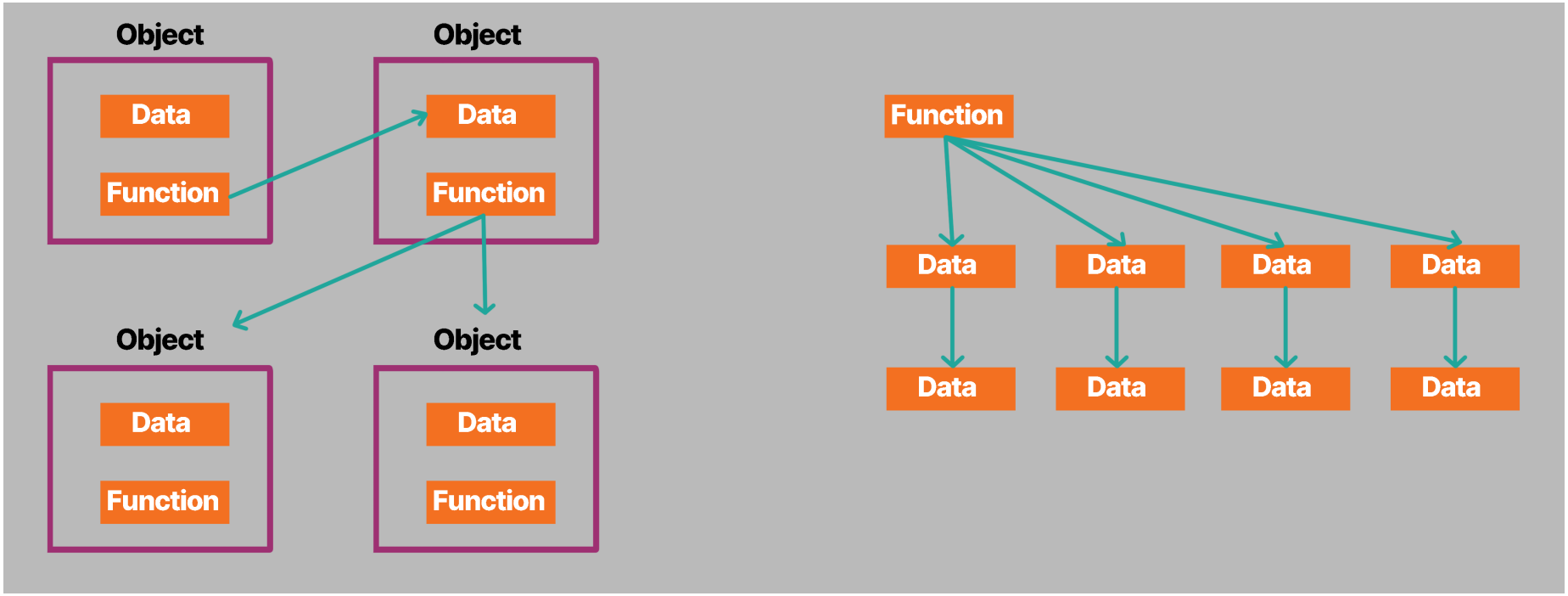
Object-Oriented Design:

- Begins with an examination of the real-world objects.
- Focuses on data that are to be manipulated by the object
- Not on the function performed by the object
- Orthogonal to Function-Oriented Design
- C++ and Java are examples of Object-Oriented programming languages

Object-Oriented vs. Function-Oriented Design

Object-Oriented Design

Function-Oriented Design



What is a Functional Language?

Opinions differ, and it is difficult to give a precise definition, but generally speaking:

- A functional language is one that supports and encourages function-oriented design.

- Most languages support function-oriented design (e.g., JavaScript, Python), but Haskell is considered the premier pure functional language.

Any Questions?

Haskell

- Haskell is the premier pure functional language.
- Haskell is a testbed for new programming ideas.
- Generics in Java came from Haskell.
- List comprehensions in Python came from Haskell.
- STM language support came from Haskell.
- React (in JavaScript) came from Haskell.
- Being a functional programmer will make you a better anywhere programmer.
- Any time you learn a new programming paradigm you get better at all the paradigms you already know.

Function Application

- In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

f(a,b) + c d

- Apply the function f to a and b, and add the result to the product of c and d.

- In Haskell, function application is denoted using space, and multiplication is denoted using *.

f a b + c * d

- Apply the function f to a and b, and add the result to the product of c and d.

- Moreover, function application is assumed to have higher priority than all other operators.

f a + b

- Means (f a) + b, rather than f (a + b)

Java (Imperative) Example

Summing the integers 1 to 10 in Java:

```
int total = 0;
for (int i = 1; i ≤ 10; i++)
   total = total + i;
```

The computation method is variable assignment.

Haskell (Functional) Example

Summing the integers 1 to 10 in Haskell:

```
sum [1..10]
```

The computation method is function application.

Any Questions?

Features of Haskell

- Concise programs
- Powerful type system
- List comprehensions
- Recursive functions
- Higher-order functions
- Functions with side effects
- Generic functions
- Lazy evaluation
- Equational reasoning

Concise Programs

- Due to the high-level nature of the functional style, programs written in Haskell are often much more concise than programs written in other languages.
- For example:

```
sum [1..10]
```

vs.

```
int total = 0;
for (int i = 1; i ≤ 10; i++)
   total = total + i;
```

- Moreover, the syntax of Haskell has been designed with concise programs in mind:
  - Few keywords
  - Allowing indentation to be used to indicate the structure of programs
- Haskell programs are often between two and ten times shorter than programs written in other languages.

Powerful Type System

- Most modern programming languages include some form of type system to detect incompatibility errors, e.g., attempting to add a number and a character.
  - JavaScript is a BIG exception!
- Haskell has a type system that usually requires little type information from the programmer.
- But it allows a large class of incompatibility errors in programs to be automatically detected prior to their execution, using a sophisticated process called type inference.
- The Haskell type system is also more powerful than most languages.
  - Very general forms of polymorphism and overloading
  - Wide range of special purpose features concerning types

List Comprehension

- One of the most common ways to structure and manipulate data in computing is using lists of values.
- Haskell provides lists as a basic concept in the language.
- It also includes a simple but powerful comprehension notation that constructs new lists by selecting and filtering elements from one or more existing lists.
  - Remember set-builder notation from EECS 210:

    S = {x | x is a positive integer less than 100}
- Using the comprehension notation allows many common functions on lists to be defined without the need for explicit recursion.

Recursive Functions

- Most programs involve some form of looping.
- In Haskell, the basic mechanism by which looping is achieved is through recursive functions that are defined in terms of themselves.
- As you may remember from EECS 168, 268, and 210, it can take some time to get used to recursion.
- But many computations have a simple and natural definition in terms of recursive functions.
- And recursive functions can be proven correct using induction.

Higher-Order Functions

- Like JavaScript, Haskell is a higher-order functional language, which means that functions can freely take functions as arguments and produce functions as results.
- Using higher-order functions allows common programming patterns, such as composing two functions, to be defined as functions within the language itself.
  - Remember from 210:

$$f \circ g(x) = f(g(x))$$

- Higher-order functions can be used to define domain-specific languages (DSLs) within Haskell itself, such as for list processing, interactive programming, and parsing.

Functions with Side Effects

- Functions in Haskell are pure functions that take all their inputs as arguments and produce all their outputs as results.
- However, many programs require some form of side effect that would appear to be at odds with purity, such as:
  - reading input from the keyboard
  - writing output to the screen
- Haskell provides a uniform framework for programming with side effects, without compromising the purity of functions using monads (more on this later).

Generic Functions

- Most languages allow functions to be defined that are generic over a range of simple types, such as different forms of numbers.
- Haskell's type system allows new structures and generic functions over them to be defined.

Lazy Evaluation

- Haskell programs are executed using a technique called lazy evaluation.
- Lazy evaluation is based on the idea that no computation should be performed until its result is actually required.
- Lazy evaluation:
  - avoids unnecessary computation
  - ensures that programs terminate whenever possible
  - encourages programming in a modular style using intermediate data structures
  - allows programming with infinite structures (e.g., list of all prime numbers)

Equational Reasoning

- Because programs in Haskell are pure functions, simple equational reasoning techniques can be used to:
  - execute programs
  - transform programs
  - prove properties of programs
  - calculate programs directly from specifications of their intended behavior
- Equational reasoning is particularly powerful when combined with the use of induction to reason about functions that are defined using recursion.

Any Questions?

Historical Background

1930s:

Alonzo Church develops the lambda calculus, a simple but powerful theory of functions.

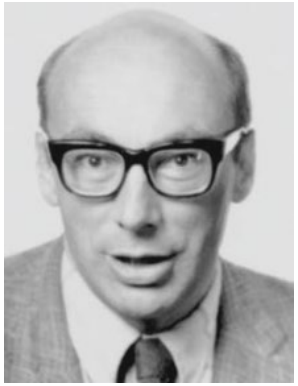
Historical Background

1950s:

John McCarthy develops Lisp, the first functional language, with some influences from the lambda calculus, but retaining variable assignments.

Historical Background

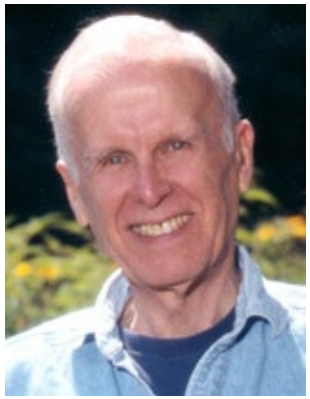
1960s:



Peter Landin develops ISWIM, the first *pure* functional language, based strongly on the lambda calculus, with no assignments.

Historical Background

1970s:



John Backus develops FP, a functional language that emphasizes *higher-order functions* and *reasoning about programs*.

Historical Background

1970s:

Robin Milner and others develop ML, the first modern functional language, which introduced *type inference* and *polymorphic types*.

Historical Background

1970s - 1980s:

David Turner develops a number of *lazy* functional languages, culminating in the Miranda system.

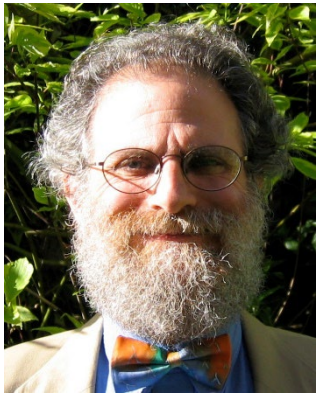
Historical Background

1987:

An international committee starts the development of Haskell, a standard lazy functional language.

Historical Background

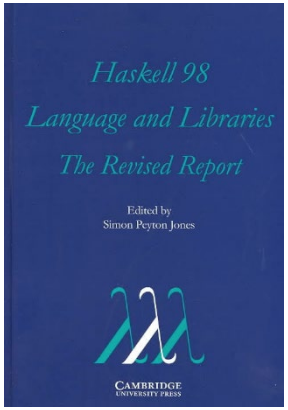
1990s:



Phil Wadler and others develop *type classes* and *monads*, two of the main innovations of Haskell.

Historical Background

2003:



The committee publishes the Haskell Report, defining a stable version of the language; an updated version was published in 2010.

Historical Background

2010-now:

Standard distribution, library support, new language features, development tools, use in industry, influence on other languages, etc.

Any Questions?

Where is Haskell Used?

- EECS 662 – Programming Languages (CS Elective)
- EECS 755 – Software Modeling and Analysis (Cybersecurity Certificate)
- EECS 762 – Programming Language Foundations I (CS Elective)

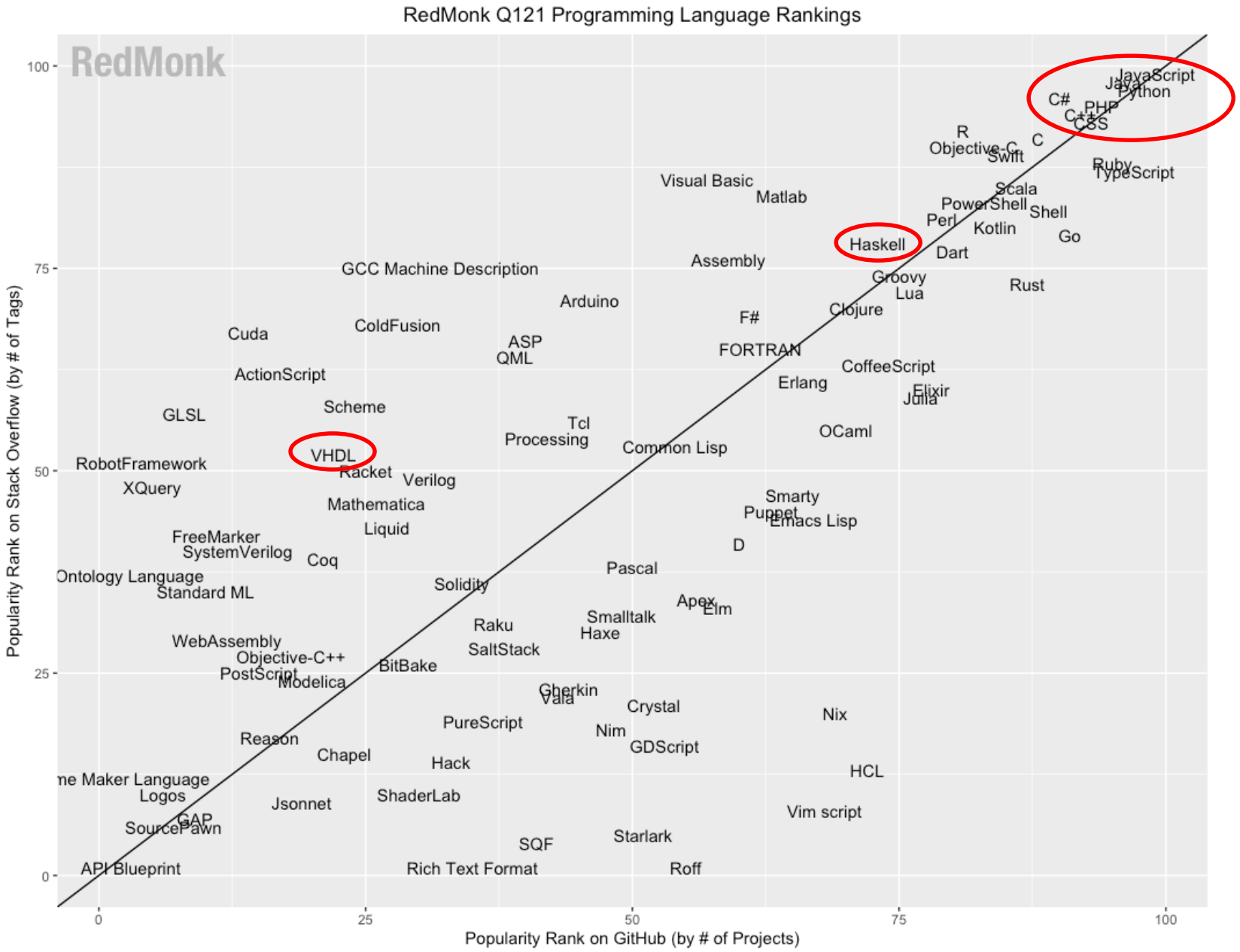
- Haskell has a diverse range of use commercially, from aerospace and defense, to finance, to web startups, hardware design firms and a lawnmower manufacturer:
  - Cardano block chain
  - Google and Grok both develop their machine learning chip compilers in Haskell.
  - Ceberos develops its wafer-scale ML compilers in Haskell
  - Jane Street develops its investment software in Haskell
  - Part of Xen is written in Haskell.

Where is Haskell Used?

- These are lists of other commercial uses of Haskell:

  https://wiki.haskell.org/Haskell_in_industry
  https://github.com/erkmos/haskell-companies
  https://haskellcosm.com

- The main user conference for industrial Haskell use is:
  CUFP - the Commercial Users of Functional Programming Workshop.


RedMonk Q121 Programming Language Rankings
RedMonk
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
25
50
75
100
JavaScript
Java
Python
C#
PHP
C++
CSS
R
Objective-C
C
Swift
Ruby
TypeScript
Visual Basic
Matlab
Scala
PowerShell
Shell
Perl
Kotlin
Go
Haskell
Dart
Assembly
GCC Machine Description
Groovy
Lua
Rust
Arduino
Clojure
F#
Cuda
ColdFusion
ASP
QML
FORTRAN
CoffeeScript
ActionScript
Erlang
Elixir
Julia
GLSL
Scheme
Tcl
Processing
OCaml
VHDL
Common Lisp
RobotFramework
Racket
Verilog
XQuery
Mathematica
Smarty
Puppet
Emacs Lisp
Liquid
FreeMarker
SystemVerilog
Coq
D
Pascal
Ontology Language
Standard ML
Solidity
Apex
Elm
Smalltalk
Raku
Haxe
WebAssembly
SaltStack
Objective-C++
PostScript
BitBake
Modelica
Gherkin
Vala
Crystal
Nix
PureScript
Nim
Reason
GDScript
Chapel
Hack
HCL
Logos
ShaderLab
Jsonnet
Vim script
GAP
SourcePawn
Starlark
SQF
API Blueprint
Rich Text Format
Roff

Any Questions?

Simple Haskell Example

```
double x = x + x
quadruple x = double (double x)

> quadruple 10
40
```

Any Questions?

In-Class Problem

- The Haskell library module Prelude provides a large number of standard functions.
- In addition to the familiar numeric functions, the library also provides many useful functions on lists.
- For example, take selects the first n elements of a list:

```
> take 3 [1,2,3,4,5]
[1,2,3]
```

- Knowing this, what is the result of these:

```
double x = x + x
quadruple x = double (double x)
take (double 2) [1,2,3,4,5,6] → ?
take (quadruple 2) [1,2,3,4,5,6] → ?
take (1+2) [1,2,3,4,5,6] → ?
```