CS4400/5400 Programming Languages

Fall 2024

Instructor: Steven Holtzen

s.holtzen@northeastern.edu

Northeastern University
Khoury College of Computer and Information Sciences

What is a “programming language”?

“A programming language is a system of notation for writing computer programs.”

https://en.wikipedia.org/wiki/Programming_language

What is a “programming language”?

“Computer programming language,

any of various languages for expressing a set of detailed instructions for a digital computer.”

https://www.britannica.com/technology/computer-programming-language

What is a “programming language”?

Programs have exactly a single **meaning**

A programming language is a means for **unambiguously specifying** a **sequence of actions** to be taken by a **computer**.

Programs **do something**

Programs only have meaning in the context of what is executing them

Languages have Two Parts

Syntax

(Form)

What does it look like?

Semantics

(Function)

“Your debt is canceled”

What does it do?

Programming Languages have Two Parts

Syntax

Python

```
x=5
print(x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

Is this *really* what the program does? What level of detail do we want?

Programming Languages have Two Parts

Syntax

JavaScript

```
let x = 5;
console.log(x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

Programming Languages have Two Parts

Syntax

Racket

```
(define x 5)
(display x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

This course is all about **precisely defining programming languages**

Syntax

Formal descriptions as grammars

Semantics

Programs that run programs

Interpreters!

- Grow big languages out of small ones
- Implement new languages

Questions you may have?

Why are there so many programming languages?

Which language should I learn? Should I use?

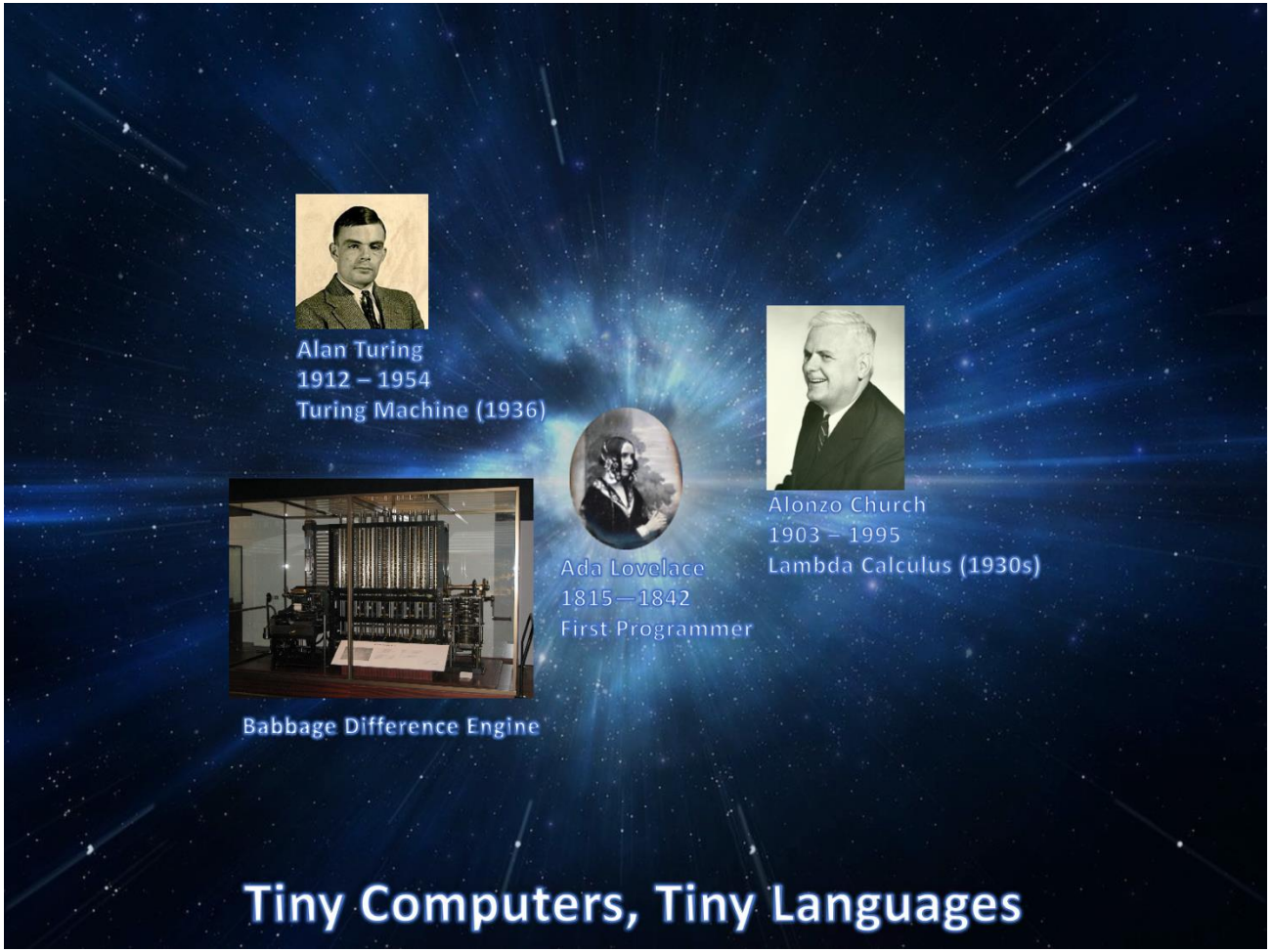
Are some languages worse than others? Better? How can I compare them?

What distinguishes one language from another?

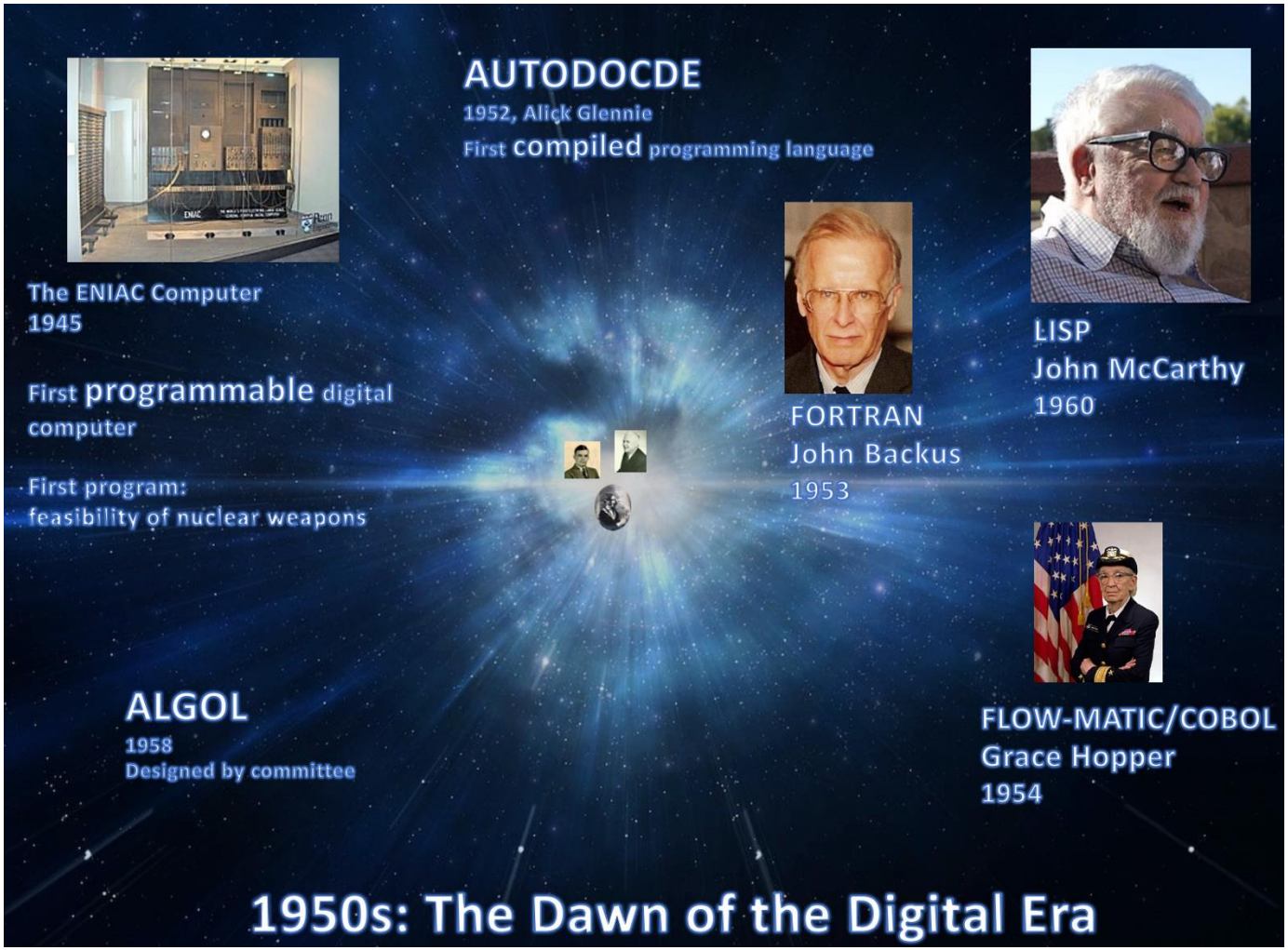
Why are new languages being made today?

How are new languages made?

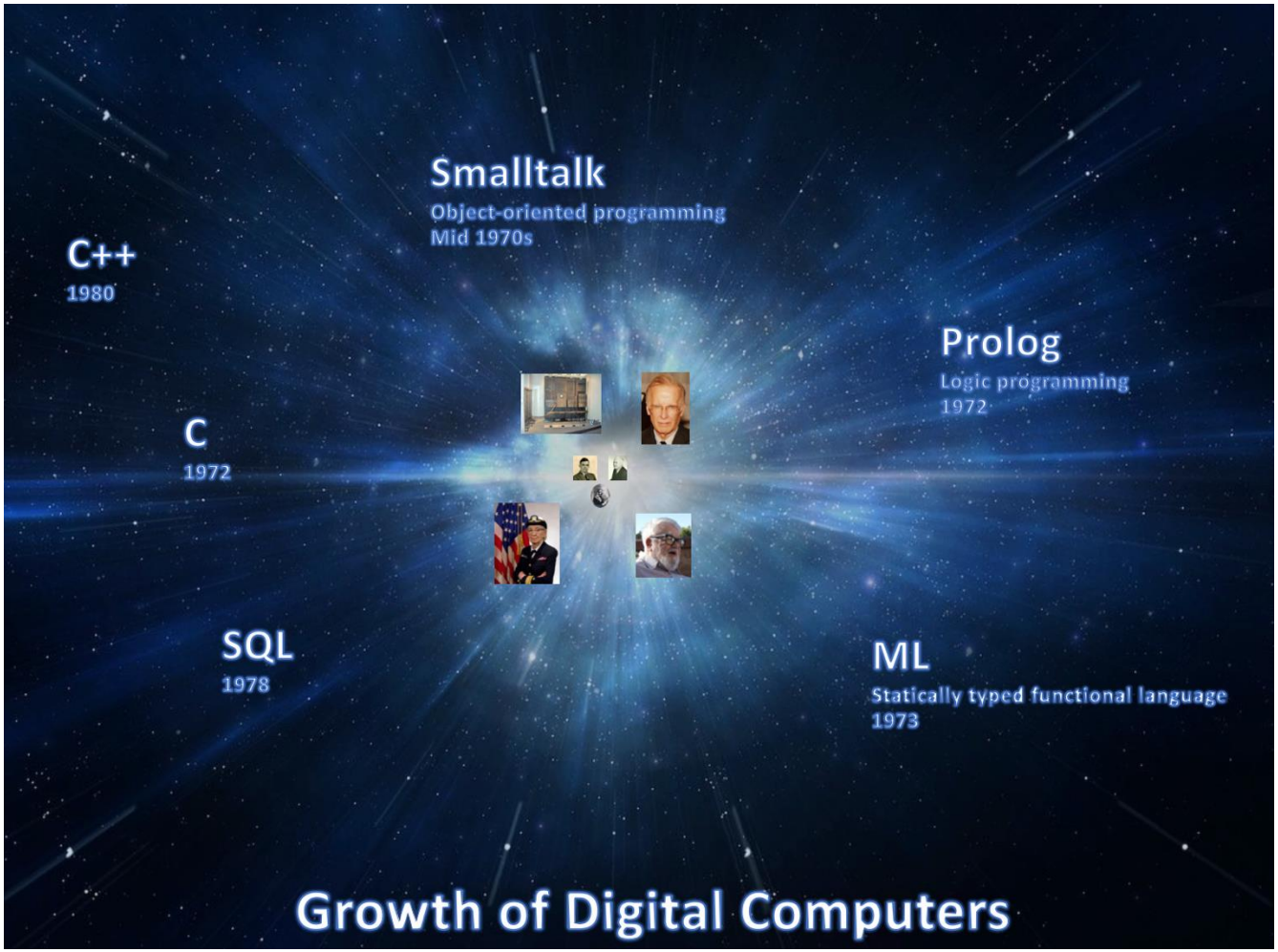
There are many programming languages
and there are more all the time



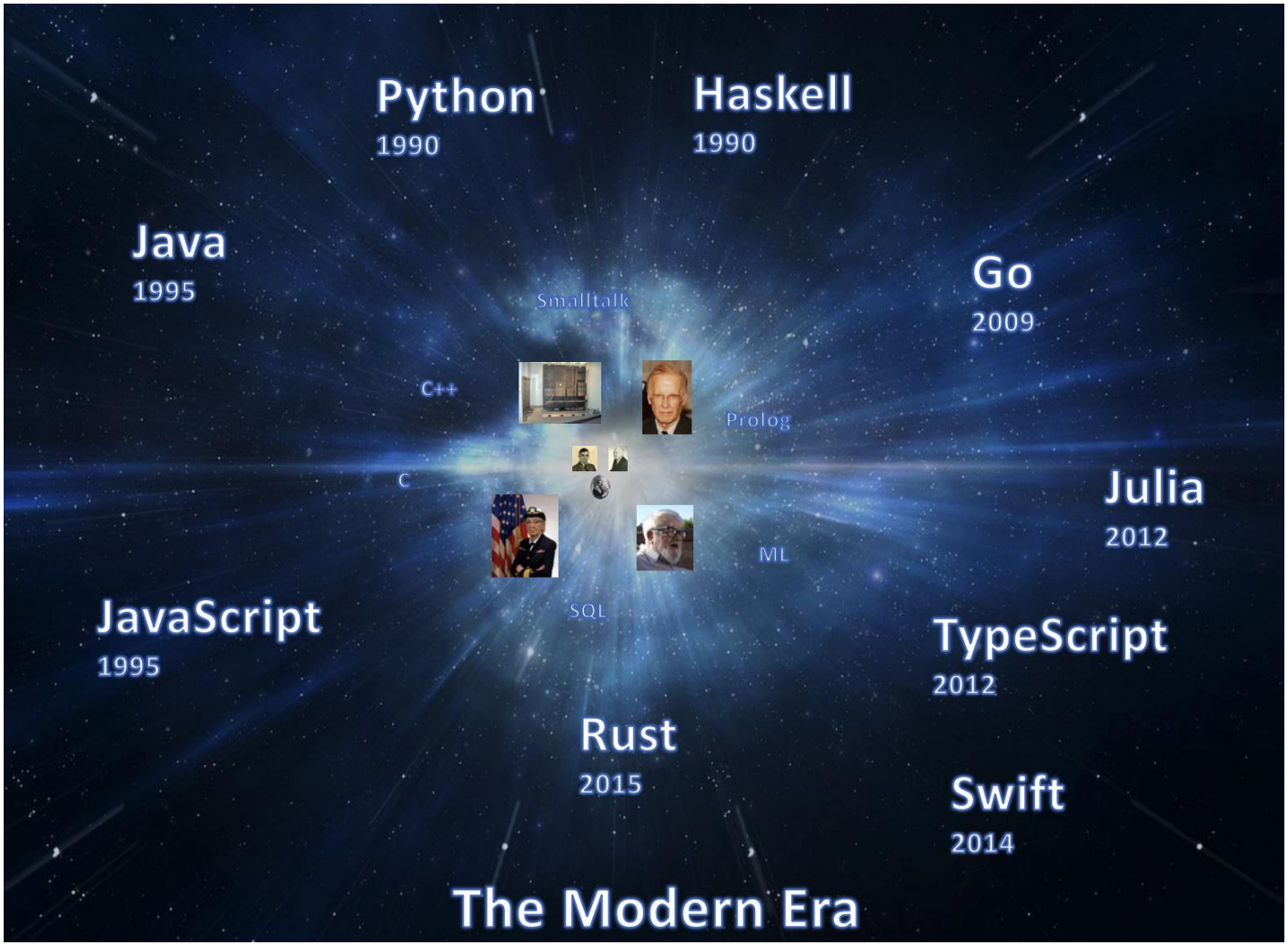
There are many programming languages
and there are more all the time



There are many programming languages
and there are more all the time



There are many programming languages
and there are more all the time



Duality of Programming Languages:

Programming languages are ***mathematical*** *constructs*

$$\frac{x = 5 \qquad y = 10}{x + y \Downarrow 15}$$

Duality of Programming Languages:

Programming languages are ***social*** *constructs*

Why study programming languages?

- Be a more effective programmer
  - Learn how to **choose languages** for your problem
  - Learn how to **design new languages** if needed
- Become equipped to learn new languages quickly
- Be prepared for an evolving world
- Enjoy an aesthetic journey through this elegant field (subjective)

Course structure and logistics

Syllabus and Course Website

- Course website is here:

https://pages.github.khoury.northeastern.edu/sholtzen/cs4400-fall24/

- Link on Canvas and my website
- Has all the course information
  - Assignments
  - Deadlines
  - Syllabus
  - Course notes (+ these slides)

You are expected to read and be aware of all content in the course syllabus!

Course staff

- **Instructor**: Steven Holtzen
  - Assistant professor at Northeastern since 2021
  - Studies probabilistic programming languages

- **Teaching assistants**:
  - Vadym Matviichuk matviichuk.v@northeastern.edu
  - Brianna Marshall marshall.br@northeastern.edu
  - Abdelrahman Madkour madkour.a@northeastern.edu

You are at one of the best schools for PL in the world

https://prl.khoury.northeastern.edu/

Many of the tools we use in this class were developed here!

Module 1: Introduction

- Become familiar with *inductively defined data* and *functional programming in Racket*

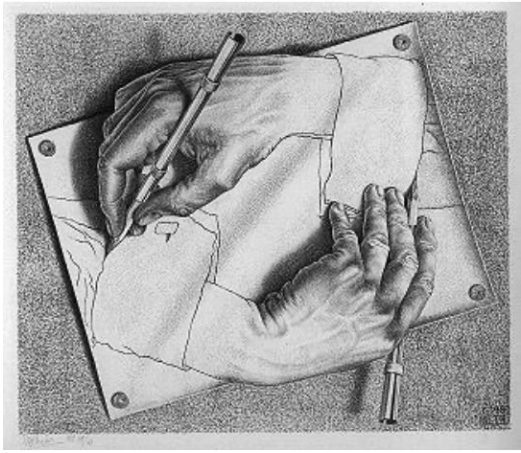
- See the design recipe for programming with inductively defined recursive data

- Become comfortable with core functional programming idioms
  - First-class functions
  - Lists
  - Pattern matching

Module 2: Growing The Lambda Calculus

- We will implement a tiny programming language one feature at a time

  1. Calculator-lang
  2. Let-lang (binding, substitution)
  3. Lambda calculus (functions)
  4. Programming in the lambda calculus
  5. Implementing the lambda calculus efficiently
  6. Recursion in the lambda calculus



Drawing Hands
M. C. Escher 1968

Module 2: Growing The Lambda Calculus

9/11/24	Abstract Syntax	
9/16/24	Let language	
9/18/24	Lambda Calculus	
9/23/24	Programming in Lambda Calculus	
9/25/24	Environments and Closures	
9/30/24	Recursion	

Schedule is subject to change!

Module 2: Why grow the lambda calculus?

- See how a real expressive programming language is made from start to finish
- Learn how to understand formal mathematical descriptions of programs and translate those into implementation
- Learn to appreciate the value of expressive language features by programming in a very restricted language without any

Module 3: Types

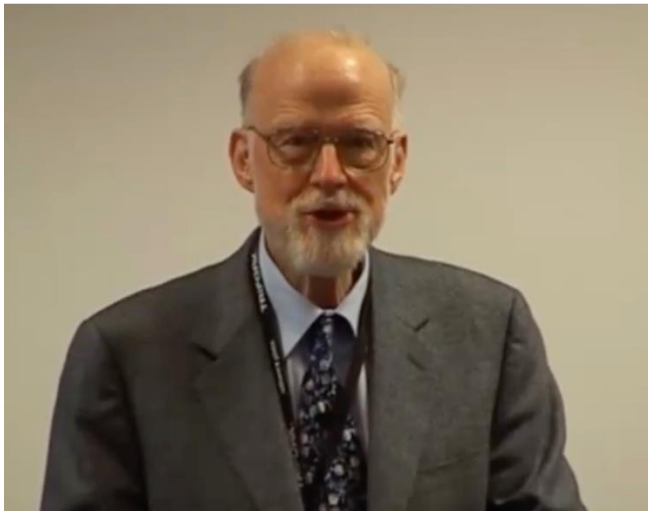
- Types are many things
  - A form of *checked specification* for programs
  - A way of *constraining the expressivity* of programs

- Example: Java

```
import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {

        // Creates a reader instance which takes
        // input from standard input - keyboard
        Scanner reader = new Scanner(System.in);
        System.out.print("Enter a number: ");

        // nextInt() reads the next integer from the keyboard
        int number = reader.nextInt();

        // println() prints the following line to the output
        System.out.println("You entered: " + number);
    }
}
```

Type annotation

Module 3: Types



Tony Hoare

Types can't prevent all bugs: this is a valid Java program:

```
public class Example {
  public static void main(String[] args) {          Object obj = null;
        obj.hashCode();
  }
}
```

Recommended viewing:
https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

Module 3: Types

https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/

FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

ONCD › BRIEFING ROOM › PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the problem of software measurability to enable the development of better diagnostics that measure cybersecurity quality.

Module 3: Why study types?

- You will encounter type systems in your day-to-day programming
- Type systems are getting increasingly sophisticated (memory safety, Rust)
- Gaining a good appreciation of how type systems are designed and implemented prepares you for a more well-typed future

Module 3: Types

10/2/24	Simple Types	
10/7/24	Simply-typed Lambda Calculus	
10/9/24	Extending Simple Types	
10/14/24	NO CLASS (Indigenous People's)	
10/16/24	Mutable State and References	
10/21/24	Polymorphism	
10/23/24	Modules	
10/28/24	Recursive Types	

Module 4: Control

- Programs don’t run in order: control flow can jump around
  - If expressions
  - Function calls
  - Exceptions
  - Async/await
  - Callbacks
- How do we add these features to our languages? How are they implemented?

Module 4: Control

10/30/24	Tail Form	
11/4/24	Exceptions	
11/6/24	Continuation-passing Style	

Module 4: Why study control?

- Learn how to implement your own control-flow primitives
  - Lets you add them to languages that don’t have them!
  - More deeply understand how compilers are implemented
  - Understand the cost of using various control-flow operations

Module 5: Topics

- We explore some modern, advanced, or emerging topics in programming languages

11/13/24	Macros	
11/18/24	Laziness	
11/20/24	Effects	
11/25/24	Probabilistic Programming	

Graded content

- **Assignments (50% Of Total)**: Roughly weekly. Posted on this webpage, turned in on Gradescope. You may discuss the problems with other students, but you must submit your own solutions.
- **Programming Projects (10%)**: There will be 1 larger programming assignment worth 10%. This will span 2 to 3 weeks.
- **Module quizzes (40% of total)**: One after each module. These will cover content from the module (non-cumulative), will be take-home and given 24 hours.

Dates for these quizzes posted on the course webpage soon.

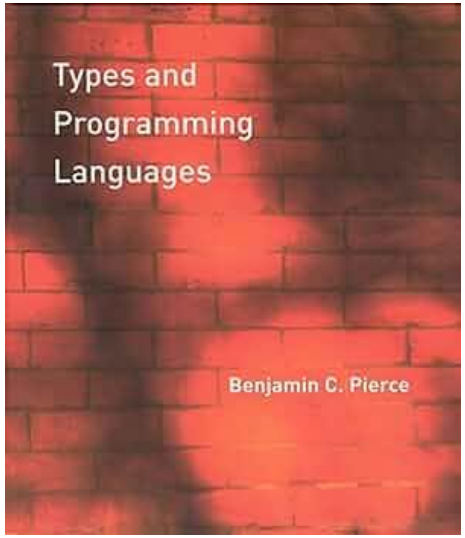
Grading policies

Letter grades will be assigned according to a standard grading threshold based on percentage of total points:

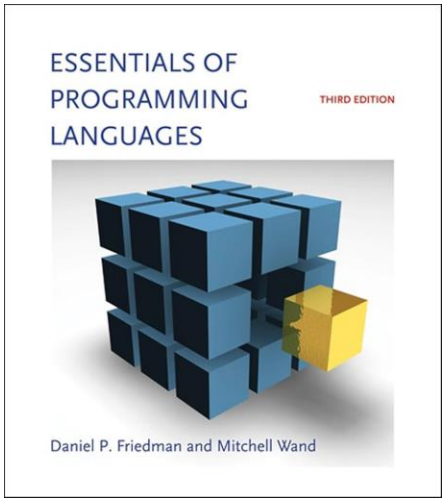
Score range	Letter grade
>93	A
≥90	A-
≥87	B+
≥83	B
≥80	B-
≥77	C+
≥73	C
≥70	C-
≥67	D+
≥63	D
≥60	D-
<60	F

Course resources

- Course notes posted on the main course webpage
- Supplementary textbooks:



Types and Programming Languages
Pierce



Essentials of Programming Languages
Friedman and Wand

Input/Output

- Discussion forum on Piazza (link to join on Canvas)
  - Major announcements made on Piazza; you are responsible for ensuring that you receive these

- Assignments turned in on Gradescope
  - There will be an autograder
  - The autograter score is not the final score (some test cases will be hidden)
  - **You must test your code!**

Support

- Office hours
  - Each TA offers 2 hours of weekly office hours
- Piazza
  - Ask questions on Piazza
  - Try to direct as many course questions there as possible (preferred over email)
- Email the instructor if you have an issue, I want to help ☺

Questions?

Introduction to Racket