Three Expectations about CS 381

• I learn how to program
• I learn details about particular programming languages
• I will see a comparison of different programming languages
• This is the Haskell class
Three *Myths* about CS 381

- I learn how to program
- I learn details about *particular* programming languages
- I will see a comparison of *different* programming languages
- This is the Haskell class
The What, How, And Why
What is CS 381 about?

What exactly is a programming language?

int x;
int f(int z) { x := f(y) ...};
{ int x;
 f(3)
}

Recursion?

Runtime Stack

Value or address?

When to evaluate?

Parameter Passing

Imperative programming = Programming “by sticky notes”

Based on what?

Abstract Syntax

Alternatives?

Programming Paradigms

“Determine” value of RHS and “assign it” to x

Which x?

Scoping

Deterministic value of RHS and “assign it” to x
The Role of Haskell in CS 381

- **Example** of a non-imperative programming paradigm (just like Prolog)
- **Tool** for describing language concepts (syntax, semantics, scope, typing)

- **Reuse** (no need for extra math)
- **More Haskell practice** (we can test and play with definitions)

**Executable PL theory**

<table>
<thead>
<tr>
<th>Metalanguage</th>
<th>English</th>
<th>Math</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precise</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checkable</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Executable</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Introduction

The correct use of language can be critical

Due to his grammar mistake, Wilbur found a position. It just wasn’t the one he wanted.
I Introduction

- About “computer science”
- The role of programming languages
- How to study programming languages?
CS Stereotypes

Countess Ada Lovelace

Introduction
What is Computer Science?

“Computer science is no more about computers than astronomy is about telescopes.”

Edsger Dijkstra (?)

Computer Science = The Science of Computing
What is Computer Science?
Science and Engineering

Understand & Explain

Science
Physics
Chemistry
Theoretical CS

builds on
enables

Engineering

Build

ME, EE, ...
CE, ...
SE, ...

Introduction
What is Computation?

Systematic Transformation of Representation

Systematic
Intensional
Description

Transformation
Function

Representation
Abstraction that preserves particular features

Description given in a Language
Multiplication

\[ \log (x \cdot y) = \log x + \log y \]
Origami

1. Start with a square paper.
2. Fold the paper in half.
3. Fold the paper in half again.
4. Fold the paper in half three times.
5. Make a triangle by folding the paper.
6. Make a second triangle by folding the paper.
7. Make a third triangle by folding the paper.
8. Make a fourth triangle by folding the paper.
9. Make a fifth triangle by folding the paper.
10. Make a sixth triangle by folding the paper.
11. Make a seventh triangle by folding the paper.
12. Make an eighth triangle by folding the paper.
13. Make a ninth triangle by folding the paper.
14. Make a tenth triangle by folding the paper.
15. Make an eleventh triangle by folding the paper.
16. Make a twelfth triangle by folding the paper.
17. Make a thirteenth triangle by folding the paper.
18. Make a fourteenth triangle by folding the paper.
19. The finished product.

How to make an origami crane

Introduction
Introduction

Logo

Moving a “Turtle”

Transformation

2D Graphics

Representation

Anything visualizable
Robozzle

- Moving a Robot
- Transformation
- Robot on a Grid
- Representation
- Navigation Puzzle
I Introduction

- About “computer science”
- The role of programming languages
- How to study programming languages?
Two Aspects

Computer Science

Algorithmic Thinking

Design Thinking

Problem solving

“How to do ...”

“How to represent ...”

HCI

Representations (for problem solving)

Language design

...
Introduction

Rules for aligning scales

By example

Domain: Numbers

Design Thinking

Algorithmic Thinking

By example

English

Visual Rules

101100
Introduction

Rules for paper folding

Design Thinking

Algorithmic Thinking

Domain:

Art

By example

English

Visual Rules

Rules for paper folding

en.wikipedia.org/wiki/Euric
en.wikipedia.org/wiki/Code_of_Euric

Introduction
Impact of Languages & Programs

- **Multiplier**
  - Design a Language
  - Empower people to ...
  - Faster and more reliable problem solving

- **Multiplier**
  - Develop a Program
  - Empower people to ...

- **Multiplier**
  - Use a Program
  - Algorithmic Thinking
  - Empower people to ...

- **Multiplier**
  - Design Thinking
  - Faster and more reliable problem solving

Introduction
Domain-Specific Languages

\[ F = ma \]
\[ E = mc^2 \]
Example Domain: Chess

Many Notations ...

**Algebraic**
1. e4 e5
2. Nf3 Nf6
3. Bb5 a6
4. Bxc6 dxc6

**Descriptive**
1. P-K4 P-K4
2. N-KB3 N-QB3
3. B-N5 P-QR3
4. BxN QPxN

**ICCF**
1. 5254 5755
2. 7163 2836
3. 6125 1716
4. 2536 4736

Introduction
Example Domain: Music

Many Notations ... (1862-1918)

Edim9
C#m7+

Introduction
DSLs vs. Programming Languages

Program: Description of a computation

Programming Language: Set of programs

⇒ A programming language defines a range of computations

complete: Java, C, Haskell, ...

partial: Domain-specific languages
(SQL, Excel, make, Latex, HTML, LabView, ...)

Introduction
The Reach of Language Design

Languages & Language design:

- Integral part of most CS activities
- Huge impact on expressiveness & usability
- Multiplier & enabler of expressiveness
- Many applications far beyond CS

Language Design is vital to CS
I Introduction

- About “computer science”
- The role of programming languages
- How to study programming languages?
Approach & Tools

Elements of programming language definition & understanding:

- Define **abstract syntax**
- Define **semantics**
  - Scoping
  - Parameter Passing
  - Exceptions
- Define **type system** *(not always)*
- Distinguish **paradigms**

Example languages
  - Haskell
  - Prolog

Metalanguage

Examples
# Learning Objectives

<table>
<thead>
<tr>
<th>Class Section</th>
<th>1 Haskell</th>
<th>2 Abstract Syntax</th>
<th>3 Semantics</th>
<th>4 Types</th>
<th>5 Scope</th>
<th>6 Parameter Passing</th>
<th>7 Exceptions</th>
<th>8 Paradigms</th>
<th>9 Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Abstract Syntax</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Static &amp; Dynamic Scoping</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Static &amp; Dynamic Typing</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Parameter Passing</td>
<td></td>
<td>✓</td>
<td></td>
<td>❌</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>5 Runtime Stack</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Polymorphism</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>7 Exception Handling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>❌</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Paradigms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>9 Semantics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>