CSE 341 : Programming Languages

Lecture 1

Hello World!

Welcome to ML

Zach Tatlock

Spring 2014

Hello! My name is…

Zach Tatlock

ztatlock@cs

CSE 546

New faculty ➔ You'll learn PL. I'll master teaching.

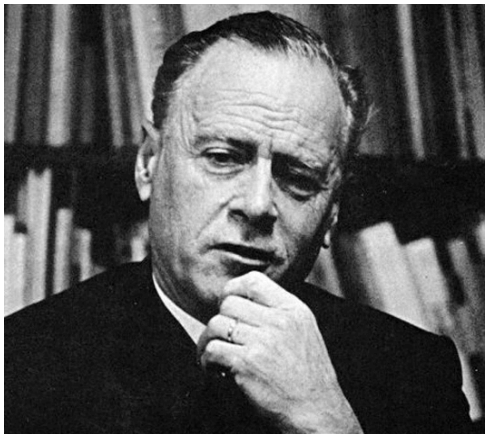
I **really** like studying programming languages.

Super stoked to explore PL with all of you.

Why?

We shape our tools and thereafter our tools shape us.

Marshall McLuhan



M. K. Asante

I discover that I think in words. The more words I know, the more things I can think about...

Reading was illegal because if you limit someone's vocab, you limit their thoughts. They can't even think of freedom because they don't have the language to.

I **really** like studying programming languages.

Super stoked to explore PL with all of you.

Why?

PL helps us *break free* to think thoughts, ask questions, and solve problems that would otherwise be inaccessible.

Welcome!

Welcome!

We have 10 short weeks to learn *the fundamental concepts* of PL.

Curiosity and persistence will get you everywhere.

We'll become better programmers:

- Even in languages we won't use
- Learn the core ideas around which *every* language is built, despite countless surface-level differences and variations

Today's class:

- Administrivia
- Dive into ML: HW 1 due Wed next week

Concise to-do list

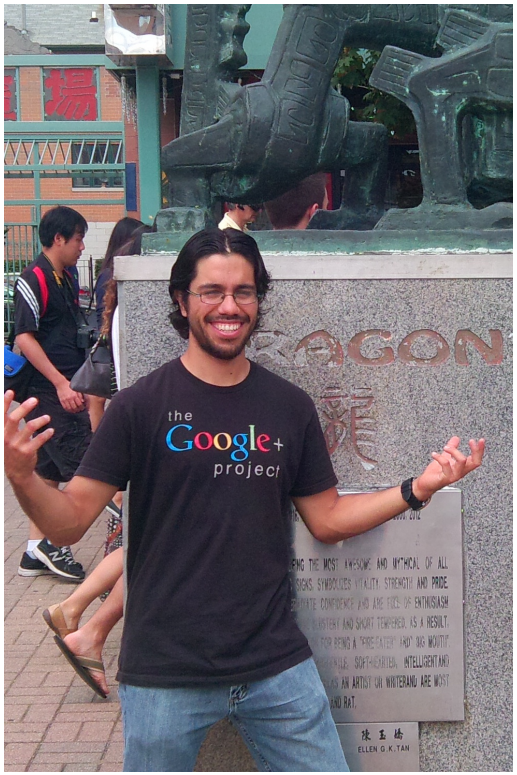
In the next 24-48 hours:

1. Read course web page:
   `http://www.cs.washington.edu/education/courses/cse341/14sp/`
2. Read all course policies (posted soon)
3. Adjust class email-list / Piazza settings as necessary

4. **Set up Emacs and ML**
   - Installation/configuration/use instructions on web page
   - Essential; non-intellectual
   - No reason to delay!

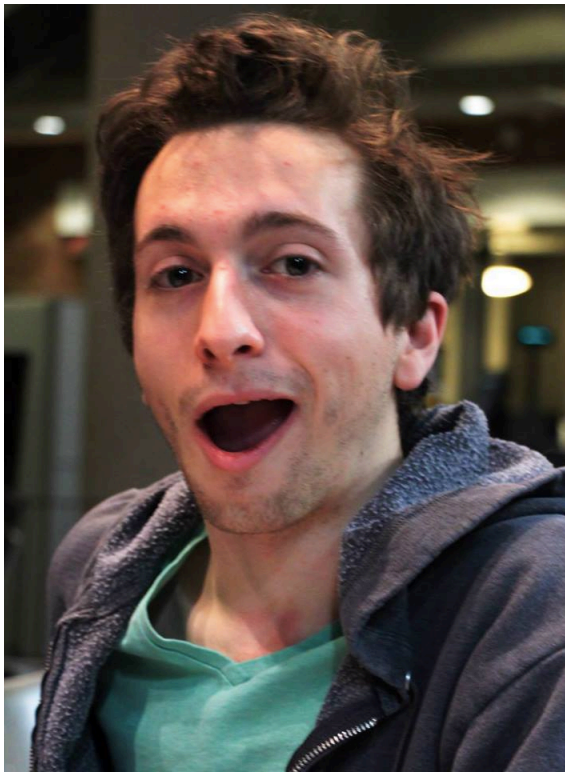
Our Incredible Guides

super ultra helpful, extraordinarily smart, stellar smiles



Armando Diaz Tolentino

Riley Klingler



Max Sherman

Our Guide in Spirit

(spiritual guide?)

Dan Grossman

Creator of this flavor of 341.

Staying in touch

- Course email list: **cse341a_sp14@u.washington.edu**
  - Students and staff already subscribed
  - You must get announcements sent there
  - Fairly low traffic
- **Piazza! Piazza! Piazza!**
  - https://piazza.com/washington/spring2014/cse341/home
  - For all discussions
  - We'll be there!
  - You can ask anonymous (to classmates) questions!
- Cookies with the Professor (that's me!)
  - 5 minute chat, 2% of grade

Lecture:

- Slides, code, and reading notes / videos posted
  - May be revised after class
  - *Take notes*: materials may not describe everything
  - The more ways we engage, the better we learn
- Ask questions, focus on key ideas
- Engage actively
  - We'll start pretty much on time (beginning matters most!)
  - *Write* down ideas and code as we go
  - We'll make lectures interactive, fun, and valuable

Section

- Required: will usually cover new material
- Sometimes more language or environment details
- Sometimes main ideas needed for homework
- *Will* meet this week: using Emacs and ML

Material often also covered in reading notes / videos

Reading Notes and Videos

- Posted for each “course unit:” reading notes and videos that go over most (all?) of the material (and some extra stuff?)

- So why come to class?
  - Videos can make class-time much more valuable (and fun)
    - Answer your questions without rushing
    - Can point to occasional optional topics/videos
    - Can try different things in class, not just recite material
  - Interact with peers and yours truly!
  - (probably) see me trip over stuff

Textbooks, or lack thereof

- Will treat the “textbooks” as optional, possibly useful references
  - Look up details you want/need to know, but often in free online resources too
- Can provide second explanations, but (because!) they often take a fairly different approach
  - If the “341 view” confuses you, lecture, reading notes, and video might all confuse you
- Some topics aren’t in the texts
- Don’t be surprised when we essentially ignore the texts
  - List on web page what sections are most relevant
- *Most but maybe not all of you will do fine without the texts*

Office hours

- Regular hours and locations on course web
  - Changes as necessary announced on Piazza

- Use them
  - *Please visit me*
  - Ideally not *just* for homework questions (but that's good too)

Homework

- Seven total
- To be done individually
- Doing the homework involves:
  1. Understanding the concepts being addressed
  2. Writing code demonstrating understanding of the concepts
  3. Testing your code to ensure you understand and have correct programs
  4. “Playing around” with variations, incorrect answers, etc.

  Only (2) is graded, but focusing on (2) makes homework harder
- Challenge problems: Low points/difficulty ratio

HEADS UP: Homeworks are terse!

- Homeworks tend to be worded very precisely and concisely
  - Technical issues deserve precise technical writing
  - Conciseness values your time as a reader
  - You should try to be precise too
- *Skimming or not understanding why a word or phrase was chosen can make the homework harder*
- By all means ask if a problem is confusing
  - Being confused is normal and understandable
  - I make a mistakes too


work
© Alamy

Academic Integrity

- Read the course policy carefully (posted this afternoon)
  - Clearly explains how you can and cannot get/provide help on homework and projects

- Always explain any unconventional action

- Academic integrity is the foundation of the university system
  - Great trust with little sympathy for violations
  - Honest work is the most important feature of a university

- When in doubt, ask!

Exams

- Midterm: … week 5-ish

- Final: Late in finals week (probably June 12/13)
  - I know, bummer city
  - Yeah, you have to be here
  - Unfortunately, probably not much I can do about it

- Same concepts, but different format from homework
  - More conceptual (but write code too)
  - Will post old exams
  - Closed book/notes, but you bring one sheet with whatever you want on it

Coursera Doppelgänger

- Dan taught this course to thousands of people around the world
- You are not allowed to participate in that class!
  - Please do not search for related homework problems!
- This should have little impact on you
  - Two courses are separate
  - 341 is a great (the greatest?) class
  - We are committed to this offering being the best ever
- Big win for us:
  - We can use videos and other materials
  - More time to interact in class and build things

Questions?

Anything I forgot about course mechanics before we discuss, you know, programming languages?

What this course is about

- Many essential concepts relevant in any programming language
  - And how these pieces fit together
- Use ML, Racket, and Ruby:
  - They let various important concepts “shine”
  - Using multiple languages shows how the same concept just can “look different” or actually be slightly different
  - In many ways simpler than Java
- Big focus on *functional programming*
  - Not using *mutation* (assignment statements) (!)
  - Using *first-class functions* (can’t explain that yet)
  - But many other topics too

Why learn this?

To free our minds from the shackles of imperative programming.

A strange environment

- Next 4-5 weeks will use
  - ML language
  - Emacs editor
  - Read-eval-print-loop (REPL) for evaluating programs
- Get things installed and configured ASAP!
  - Either in the department labs or your own machine
  - We've written thorough instructions (questions welcome)
- Only then can you focus on the content of Homework 1
- Working in strange environments is a CSE life skill

Mindset

- "Let go" of all programming languages you already know
- For now, treat ML as a "totally new thing"
  - Time later to compare/contrast to what you know
  - For now, "oh that seems kind of like this thing in [Java]" will confuse you, slow you down, and you will learn less
- Start from a blank file…

A very simple ML program

[The same program we just wrote in Emacs; here for review]

```
(* My first ML program *)

val x = 34;

val y = 17;

val z = (x + y) + (y + 2);

val q = z + 1;

val abs_of_z = if z < 0 then 0 - z else z;

val abs_of_z_simpler = abs z
```

A variable binding

```
val z = (x + y) + (y + 2); (* comment *)
```

More generally:

```
val x = e;
```

- *Syntax*:
  - *Keyword* **`val`** and *punctuation* = and **`;`**
  - *Variable* **`x`**
  - *Expression* **`e`**
    - Many forms of these, most containing *subexpressions*

The semantics

- Syntax is just how you write something

- Semantics is what that something means
  - Type-checking (before program runs)
  - Evaluation (as program runs)

- For variable bindings:
  - Type-check expression and extend static environment
  - Evaluate expression and extend dynamic environment

So what is the precise syntax, type-checking rules, and evaluation rules for various expressions? Good question!

ML, carefully, so far

- A program is a sequence of *bindings*
- *Type-check* each binding in order using the *static environment* produced by the previous bindings
- *Evaluate* each binding in order using the *dynamic environment* produced by the previous bindings
  - Dynamic environment holds *values*, the results of evaluating expressions
- So far, the only kind of binding is a *variable binding*
  - More soon

Expressions

- We have seen many kinds of expressions:

  `34` **`true`** **`false`** **`x`** ***`e1`*****`+`*****`e2`** ***`e1`*****`<`*****`e2`**

  `if` ***`e1`*** **`then`** ***`e2`*** **`else`** ***`e3`***

- Can get arbitrarily large since any subexpression can contain subsubexpressions, etc.

- Every kind of expression has
  1. Syntax
  2. Type-checking rules
     - Produces a type or fails (with a bad error message ☹)
     - Types so far: **`int`** **`bool`** **`unit`**
  3. Evaluation rules (used only on things that type-check)
     - Produces a value (or exception or infinite-loop)

Variables

- Syntax:

  sequence of letters, digits, _, not starting with digit

- Type-checking:

  Look up type in current static environment

  – If not there fail

- Evaluation:

  Look up value in current dynamic environment

Addition

- Syntax:

  e1 **`+`** ***e2*** where ***e1*** and ***e2*** are expressions

- Type-checking:

  If ***e1*** and ***e2*** have type **`int`**,

  then ***e1*** **`+`** ***e2*** has type **`int`**

- Evaluation:

  If ***e1*** evaluates to **`v1`** and ***e2*** evaluates to **`v2`**,

  then ***e1*** **`+`** ***e2*** evaluates to sum of **`v1`** and **`v2`**

Values

- All values are expressions
- Not all expressions are values
- A value “evaluates to itself” in “zero steps”
- Examples:
  - **`34`**, **`17`**, **`42`** have type **`int`**
  - **`true`**, **`false`** have type **`bool`**
  - **`()`** has type **`unit`**

Slightly tougher ones

What are the syntax, typing rules, and evaluation rules for conditional expressions?

What are the syntax, typing rules, and evaluation rules for less-than expressions?

The foundation we need

We have many more types, expression forms, and binding forms to learn before we can write “anything interesting”

Syntax, typing rules, evaluation rules will guide us the whole way!

HW 1: functions, pairs, conditionals, lists, options, local bindings

- Earlier problems require less

Will not add (or need):

- Mutation (a.k.a. assignment): use new bindings instead
- Statements: everything is an expression
- Loops: use recursion instead