PL: Lecture #1

Tuesday, September 10th

Intro to CS4400/CS5400

- General plan for how the course will go.
- Administrative stuff. (Mostly going over the web pages.)

  https://pl.barzilay.org/

Intro to Programming Languages

> 👉 ***PLAI §1***

- Why should we care about programming languages? (Any examples of big projects *without* a little language?)
- What defines a language?
  - syntax
  - semantics
  - libraries (runtime)
  - idioms (community)
- How important is each of these?
  - libraries give you the run-time support, not an important part of the language itself. (BTW, the line between “a library” and “part of the language” is less obvious than it seems.)
  - idioms originate from both language design and culture. They are often misleading. For example, JavaScript programmers will often write:

```
function explorer_move() { doThis(); }
function mozilla_move() { doThat(); }
if (isExplorer)
  document.onmousemove = explorer_move;
else
  document.onmousemove = mozilla_move;
```

    or

```
if (isExplorer)
  document.onmousemove = function() { doThis(); };
else
  document.onmousemove = function() { doThat(); };
```

    or

```
document.onmousemove =
  isExplorer ? function() { ... }
             : function() { ... };
```

    or

```
document.onmousemove =
  isExplorer ? () => { doThis(); } : () => { doThat(); };
```

or

```
document.onmousemove = isExplorer ? doThis : doThat;
```

How many JavaScript programmers will know what this does:

```
function foo(n) {
  return function(m) { return m+n; };
}
```

or this:

```
n => m => m+n;
(x,y) => s => s(x,y);
```

or, what seems fishy in this? —

```
const foo = (x,y) => bar(x,y)
```

Yet another example:

```
let x = "";
while (foo())
  x += whatever();
```

How would you *expect* this code perform? How do you think it does in the reality of many uses of JS by people who are not really programmers?

- Compare:
  - `a[25]+5` (Java: exception)
  - `(+ (vector-ref a 25) 5)` (Racket: exception)
  - `a[25]+5` (JavaScript: exception (or NaN))
  - `a[25]+5` (Python: exception)
  - `$a[25]+5` (Perl: 5)
  - `a[25]+5` (C: ***BOOM***)

  ➡ syntax is mostly in the cosmetics department; semantics is the real thing.
- Another example:
  - `a + 1 > a` (Python: always true)
  - `(> (+ a 1) a)` (Racket: always true)
  - `a + 1 > a` (C: sometimes true)

- How should we talk about semantics?
  - A few well-known formalisms for semantics.
  - We will use programs to explain semantics: the best explanation *is* a program.
  - Ignore possible philosophical issues with circularity (but be aware of them). (Actually, they are solved: Scheme has a formal explanation that can be taken as a translation from Scheme to logic, which means that things that we write can be translated to logic.)
  - We will use Racket for many reasons (syntax, functional, practical, simple, formal, statically typed, environment).