Lecture #1 (plus practice session)

Administrative stuff.

Lecturer: Eran Omri
Email address: omrier@ariel.ac.il
Room: 11.2.11
Office hours: TBD

References:
Main:
Eli Barzilay's course -- http://pl.barzilay.org/
Introduction to Programming Languages (CS4400/CS5400)
Based on -- PLAI book --
http://pl.barzilay.org/plai.pdf

Also good to know:
Mira Blalaban's course -- http://www.cs.bgu.ac.il/~ppl122/Main
And her book -- http://www.cs.bgu.ac.il/~mira/ppl-book.pdf

Language Installation Instructions:

1. Install Racket (and DrRacket)
2. Run DrRacket
3. Choose File > Install .plt file
4. In the "web" tab, enter:
http://pl.barzilay.org/pl.plt

Course overview

- General plan for how the course will go.
  - We will be designing a programming language
  - We will start with a very simple language – for performing simple arithmetic operations.
  - With time, we will expand the language to enable programmers to perform more sophisticated tasks.
- The goal of the above process is to take the point of view of programming languages designers (while keeping in mind that of programmers).
  - Through that we hope to understand some of the issues that come up when designing programming languages.
  - Should we all be designers? No, but you would like to understand what lies behind many decisions (don't want to feel like taking your car to the mechanic and be overcharged)
- Why should we care about programming languages? (Any examples of big projects *without* a little language?)

PLAI Chapter 1

What are the essentials of a language?

- **Syntax** -- the formal rules of how to construct programs (phrases).
- **Semantics** – The true meaning of things.

An important difference between syntax and semantics:

A good way to think about this is the difference between the string "42" stored in a file somewhere

(two ASCII values), and the number 42 stored in memory (in some representation). You could also continue with the above example: there is nothing wrong with **"robbery"** – it's just a word, but **robbery** is something you'll go to jail for.

* How important is each of these?

Syntax:

- Compare:

```
a[25]+5                    (Java: exception)

(+ (vector-ref a 25) 5)    (Racket: exception)

a[25]+5                (JavaScript: NaN or undefined)

a[25]+5                   (Python: exception)

$a[25]+5                  (Perl: 5)

a[25]+5                  (C: <<<BOOM>>>)

a[25]+5                 (ML: not an array ref at all)
```

-> syntax is mostly in the cosmetics department.

-> semantics is the real thing.

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).

The evaluation function that Racket uses is actually – a function that takes a **piece of syntax** and returns (or executes) **its semantics**.

Introduction to Racket

* General layout of the parts of Racket:

- The Racket language is (mostly) in the Scheme family, or more generally in the Lisp family;

- Racket: the core language implementation (language and runtime), written mostly in C;

- The actual language(s) that are available in Racket have lots of additional parts that are implemented in Racket itself;

- GRacket: a portable Racket GUI extension, written in Racket too;

- DrRacket: a GRacket application (also written in Racket);

- Our language(s)...

* **Documentation**: the Racket documentation is your friend (But beware that some things are provided in different forms from different places)

Side-note: (programming languages are alive and changing)

E.W. DIJKSTRA

"Goto Statement Considered Harmful."

This paper tries to convince us that **the well-known goto statement should be eliminated from our programming languages**…

…I doubt that any real-world **program will ever be written in such a style**… Publishing this **would waste valuable paper**… **30 years from now, the goto will still be alive and well** and used as widely as it is today.

Quick Introduction to Racket

Racket syntax... Similar to other Sexpr-based languages.

Reminder: the parentheses can be compared to C/etc function cal parens - they always mean that some function is applied:
Reminder: In C - foo(int x);

This is the reason why (+ (1) (2)) won't work: if you use C syntax that would be "+(1(), 2())" but "1" isn't a function so "1()" is an error.

`define' expressions: are used for creating new **bindings**.

Note: do not try to use them to change values. For example, you should not try to write something like (define x (+ x 1)) in an attempt to mimic `x = x+1'. It will not work.

End of class #1
