CIS 352

Programming Languages

Spring 2020

Kristopher Micinski, Jack Vining, Yihao Sun

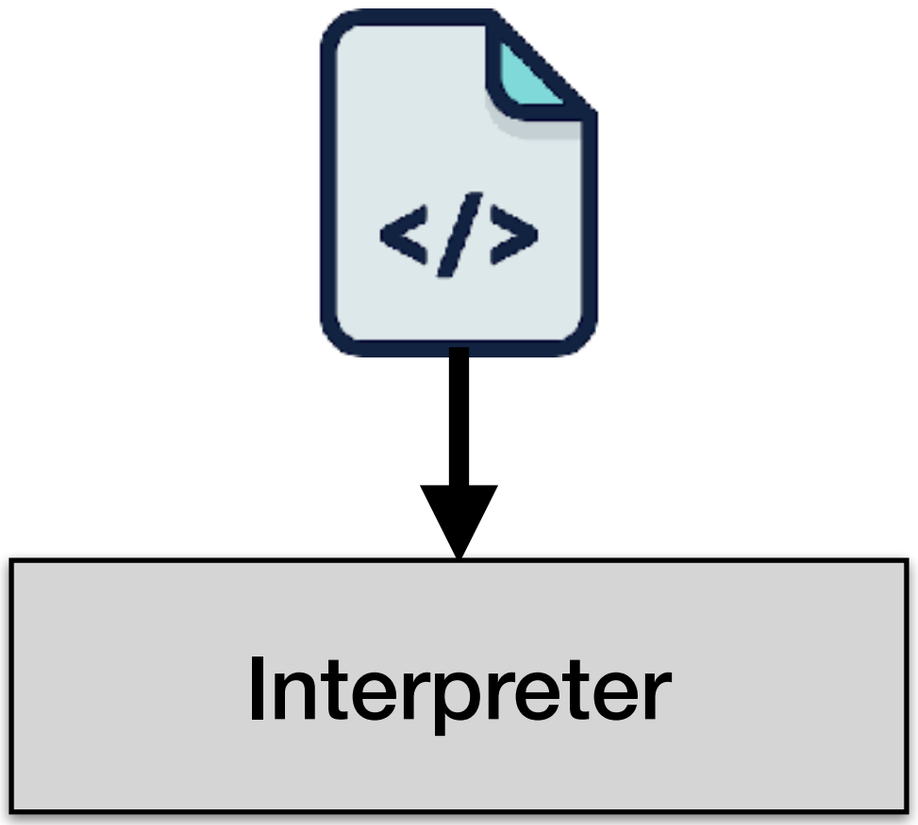
The purpose of this class is to make you a better programmer

“Programming for programming’s sake”

Why study programming languages?

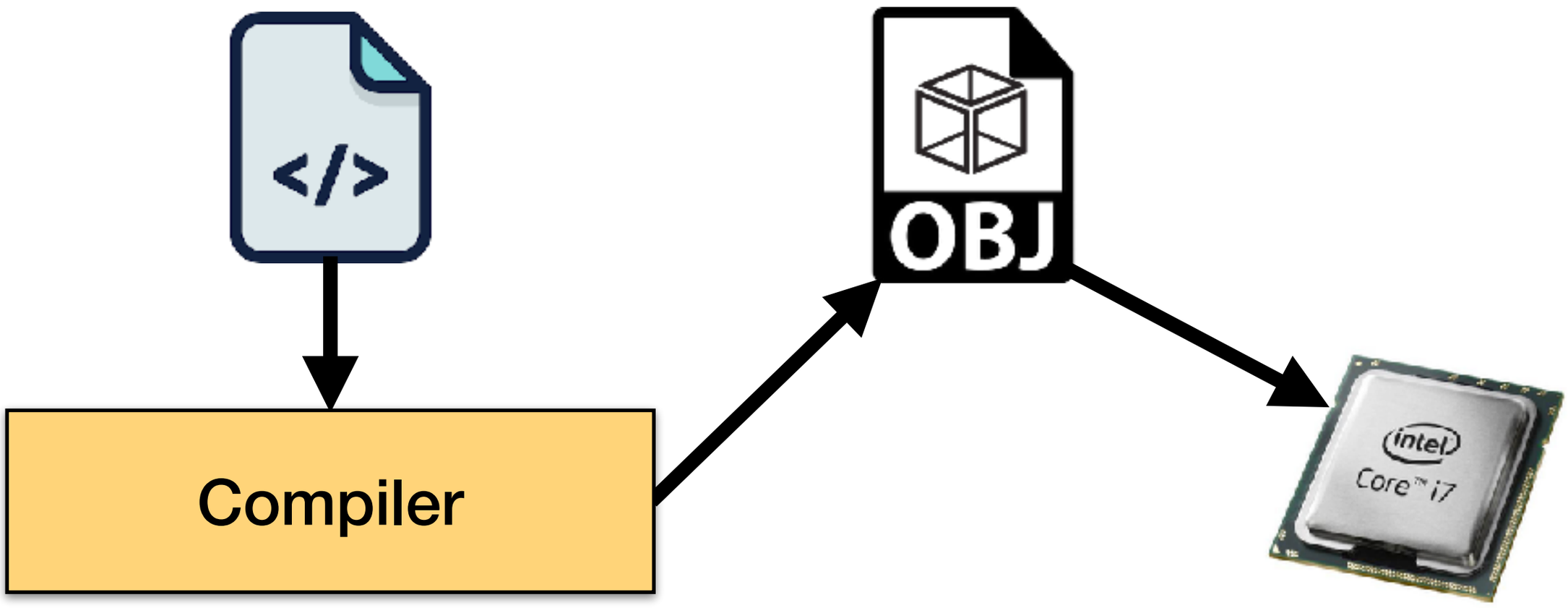
Key idea: learn programming languages by **building** them

Key idea: learn programming languages by **building** them



Consumes code and executes it

Translates code into lower-level language



Then sent to a **different** interpreter
(such as a physical CPU)

In this class, we will be writing several interpreters / compilers

But for relatively small languages

Key idea: study **core concepts** in isolation

The only language you need…

$$
\begin{aligned}
e ::= & \ (\lambda(x)\ e) \\
& \ x \\
& \ (e_0\ e_1)
\end{aligned}
$$

The only language you need…

$$
\begin{aligned}
e ::= & \ (\lambda(x)\ e) \\
& \ x \\
& \ (e_0\ e_1)
\end{aligned}
$$

But this is cumbersome
lacks many key ideas!

Stack, heap, control (if), builtins (+),
closures, effects, libraries, runtime, etc…

What programming ***paradigms*** have you heard of?

Programming languages: paradigms

- ***Imperative languages*** emphasize issuing commands that tell the machine what *to do next* at each step of evaluation.

- ***Structured languages*** emphasize structured control-flow (i.e., not unstructured goto commands) that can be properly nested, especially sequencing, conditionals, and looping constructs (while, for, do).

- ***Procedural programming*** is imperative programming with subroutines —emphasizes abstracting behaviors over data (***procedural abstraction***).

- ***Object-oriented programming*** emphasizes encapsulation of behaviors (methods) and data (fields) within classes, abstract modular schema for program values, that are instantiated as resiliant, self-contained objects at run-time. Inheritance hierarchies used to promote code-reuse.

- ***Reactive programming*** emphasizes responding to events.

Programming languages: paradigms

- ***Dynamic languages*** emphasize permitting arbitrary manipulation of program values, control, and the environment at runtime. Primarily these use duck typing / structural typing. A related paradigm is that of ***reflective*** programming—dynamically modifying types at runtime.

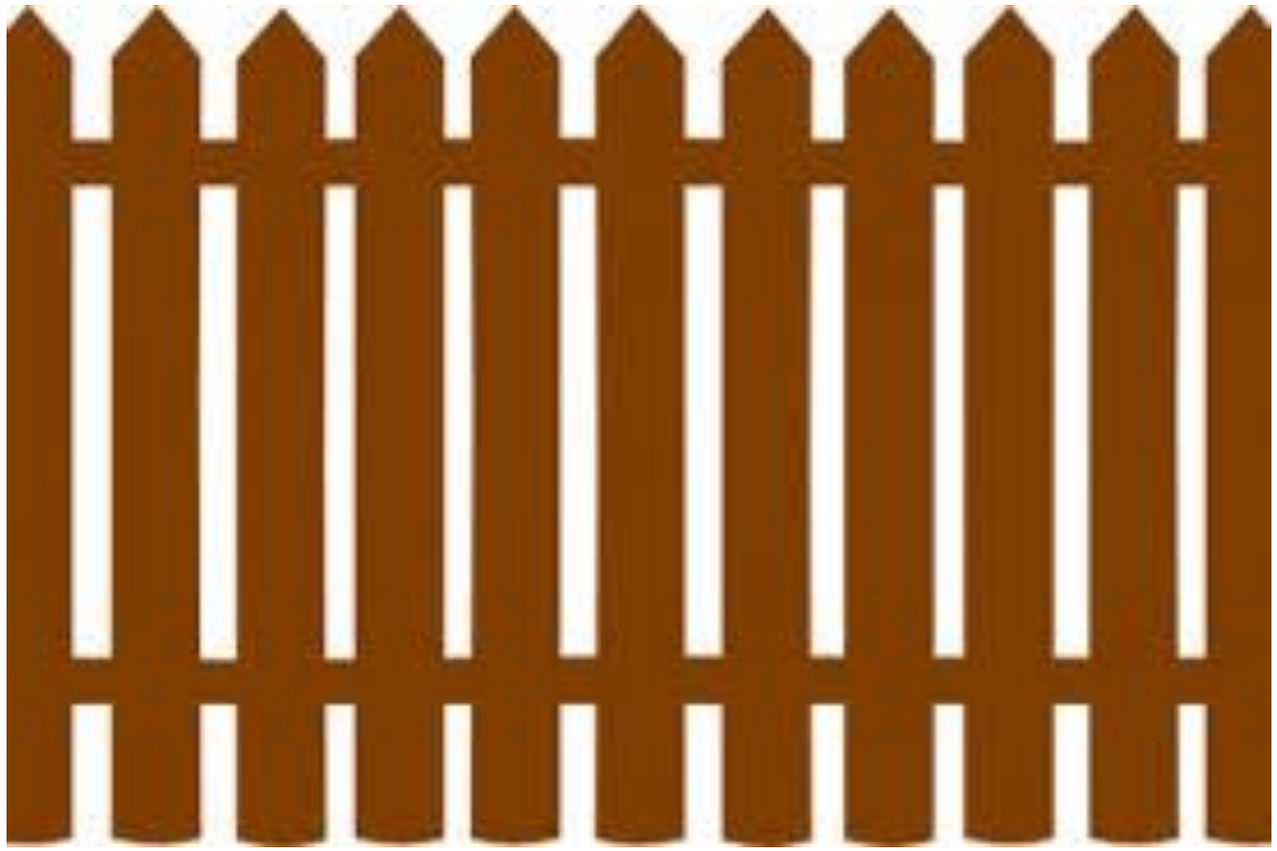
- ***Static languages*** emphasize bounding program behavior ahead-of-time. Primarily these use nominal typing and are type-checked.

- ***Array languages*** emphasize concisely manipulating arrays, matrices.

- ***Functional programming*** emphasizes immutability, like math. Programs are constructed from pipelines of composed functions that transform inputs to outputs without affecting the surrounding environment.

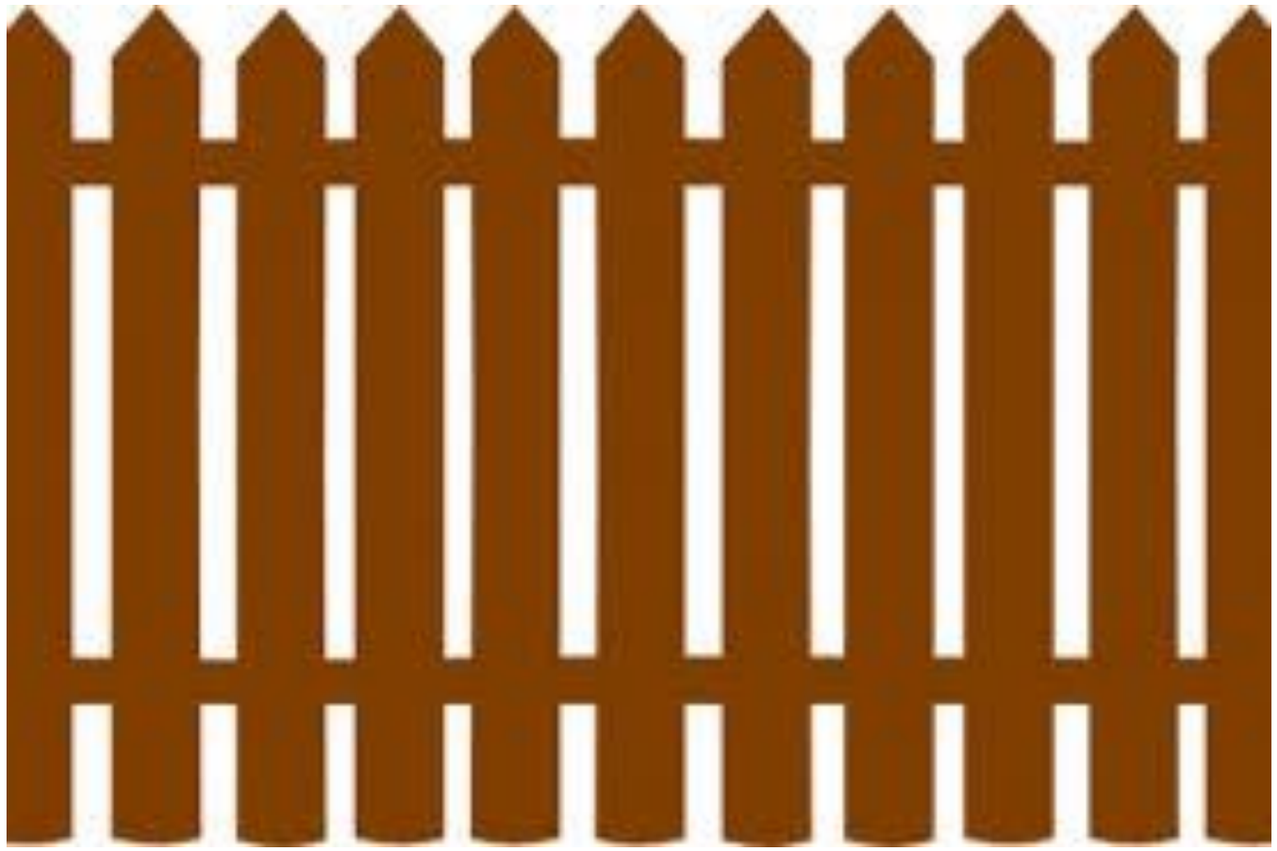
- ***Logic programming*** emphasizes declarations, propositions, logical constraints. The programmer states what must be true of a solution.

Programming languages: imperative paradigm



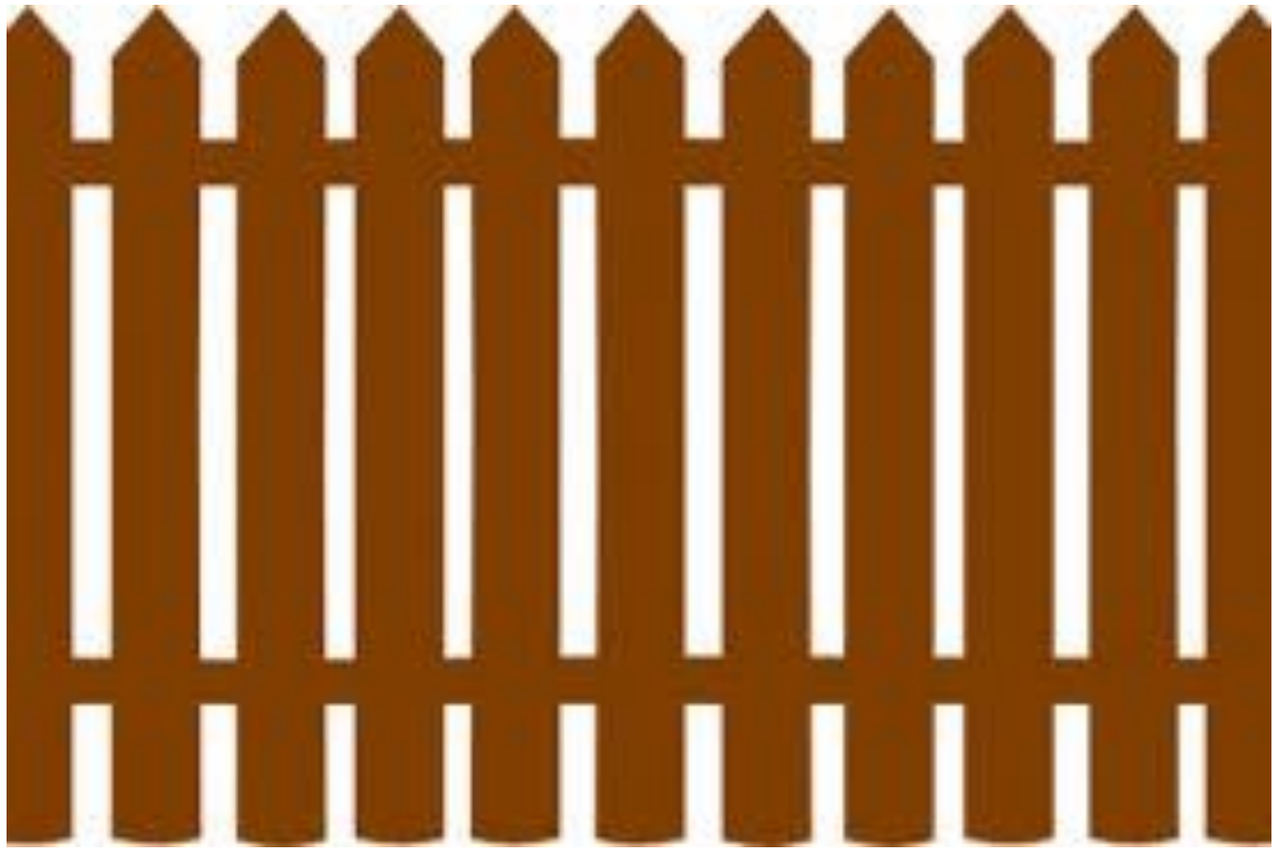
```
Place first board and rails
While fence incomplete:
    move half-a-foot to the left
    position a new board
    position a nail
    hammer nail into top rail
    ...
```

Programming languages: functional paradigm



```
function build_fence(len):
    if len == 1:
        return rails_and_first_picket()
    else:
        return add_one_picket(build_fence(len-1))
```

Programming languages: logical paradigm

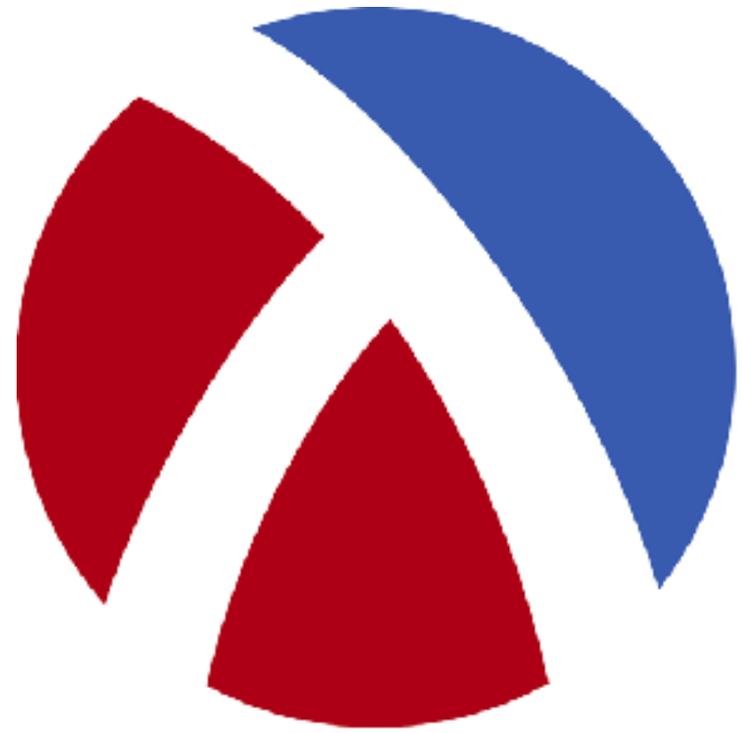


```
def fence.
fence is 5 ft tall.
fence has two rails.
fence has 50 pickets,
    each picket is 4” wide
    every picket is 2” from at least one other.
```

Racket

`https://racket-lang.org/`

- We will be using **Racket**
- Racket is the best language for **writing interpreters**
- Unique mix of features:
  - Structured / functional programming
  - Dynamically typed
  - Language-oriented programming



Grade breakdown

56% : 8 coding projects are 7% of your grade each.

0% : Weekly exercises will be posted that are ***optional/extra credit***.

10% : 2 coding exams are 5% of your grade each.

14% : A midterm worth 14%.

20% : A final worth 20%.

(If you earn <60% of the points for projects, labs, or exams,
your letter grade may be dropped to match.)

Projects

- Intro to Racket
- Pagerank
- Quadtrees
- Closure-creating interpreter for core Scheme
- Church-encoding compiler from Scheme -> Lambda
- Interpreter for ANF Scheme w/ call/cc (CEK)
- Interpreter for Scheme + set! (CES, store-passing)
- Logic programming in Mini-Kanren

Automated grading

`https://autograde.org/`

Should get uname/pw by **tonight**

First assignment: **next Monday night**

All submissions are graded using Racket 7.5 and Python 3.7
on an Ubuntu 18.04 LTR server.

If you have any trouble configuring this (or a compatible environment)
on your home machine, I highly recommend you develop with:

+

(OS X and non-Debian-based linux distros are likely to work with minimal headache;
Windows users have reported success using the Linux Subsystem for Windows.)

Academic Honesty

- *Assignments* and *Exams* ***must*** be completed ***alone***.
    - You may not collaborate, discuss solutions, screen share, copy code, look over someone's shoulder…
    - We can and do catch cheating; even when clever…
    - ***Ask us*** if unsure whether something is permitted.
- Anything normally considered cheating on assignments or exams is ***permitted, only for*** *Examples* or *Exercises*.
    - So long as you're making a sincere attempt to learn and understand solutions, you ***may*** work with others, collaborate on problems, or even share code, ***only*** for problems marked *exercise* or *example*.

Tasks for the first week

1) **Configure your system.** Download the latest version of Racket from https://racket-lang.org/

2) **Setup your autograder account.** You should receive an email invitation to use the autograder before the first lab session. Change your password and download the first assignment **"a0"**

3) **Sign up for Slack.** Ask us any questions via Slack