• Peer learning:
  – more Ruby OOP
• Let’s talk through some problems:
  – SaaS architecture
  – REST
• Demos/livecoding:
  – Sinatra
  – Cookies. Mmmm, cookies.
  – Maybe we’ll build a simple RESTful app
class Student
  def name
    capitalize_words(@student_name)
  end
end

A ☐ **Illegal**: accessor method must have same name as instance variable
B ☐ **Illegal**: accessor method must return "raw" instance variable
C ☐ **Legal**: can use obj.student_name to get "raw" value of this attribute
D ☐ **Legal**, but no way to get "raw" value of this attribute
class String
  def curvy?
    !("AEFHIKLMNTVWXYZ".include?(self.upcase))
  end
end

A  String.curvy?("foo")

B  "foo".curvy?

C  self.curvy?("foo")

D  curvy?("foo")
What is this? What year?

• 1950s: 90 minutes to process reservation
• 80% of unsold seats were due to manual processing errors
• 1957: contract ($40M = $300M today)
• 1961: two 7090s, 1500 terminals, 20K reserv./day
• 1964: all AA reservations via SABRE
• 1976: travel agents get terminals
• 1980s/90s: access via Compu$erve, AOL
Microquiz question 1

• In the Model-View-Controller architecture common to many SaaS apps, which of the three components holds the main business logic of the app?
  – (A) model
  – (B) view
  – (C) controller
Microquiz question 2

• Which explanation *most closely* describes the key idea behind REST (REpresentational State Transfer)?
  – (A) The client and server must agree on the representation of an object and its state
  – (B) Client requests should specify a resource and a state-inspection or state-change operation to be done on the resource
  – (C) A representation of the state of the session is what allows “stateless” HTTP requests to be associated with a single session
§2.1  100,000 feet
• Client-server (vs. P2P)

§2.2  50,000 feet
• HTTP & URLs

§2.3  10,000 feet
• XHTML & CSS

§2.4  5,000 feet
• 3-tier architecture
• Horizontal scaling

§2.5  1,000 feet—Model-View-Controller
(vs. Page Controller, Front Controller)

§2.6  500 feet: Active Record models (vs. Data Mapper)
§2.7  500 feet: RESTful controllers (Representational
State Transfer for self-contained actions)
§2.8  500 feet: Template View (vs. Transform View)
Web at 100,000 feet

- The web is a client/server architecture
- It is fundamentally request/reply oriented
Client-Server vs. Peer-to-Peer

- High-level architecture of the overall system
  - Soon we’ll talk about architecture “inside” boxes
- Client & server each *specialized* for their tasks
  - Client: ask questions on behalf of users
  - Server: wait for & respond to questions, serve many clients
- Design Patterns capture common structural solutions to recurring problems
  - Client-Server is an *architectural pattern*
Nuts and bolts: TCP/IP protocols

- IP (Internet Protocol) *address* identifies a physical network interface with four *octets*, e.g. **128.32.244.172**
  - Special address **127.0.0.1** is “this computer”, named *localhost*, even if not connected to the Internet!

- TCP/IP (Transmission Control Protocol/Internet Protocol)
  - IP: no-guarantee, best-effort service that delivers *packets* from one IP address to another
  - TCP: make IP reliable by detecting “dropped” packets, data arriving out of order, transmission errors, slow networks, etc., and respond appropriately
  - TCP *ports* allow multiple TCP apps on same computer

- Vint Cerf & Bob Kahn: 2004 Turing Award for Internet architecture & protocols, incl. TCP/IP
  
  
  ```
  GET /bears/ HTTP/0.9 200 OK
  ```
  
  ```
  GET /bears/ HTTP/0.9 200 OK
  ```
Now that we’re talking, what do we say?

Hypertext Transfer Protocol

- an ASCII-based request/reply protocol for transferring information on the Web

- **HTTP request** includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request

- **HTTP response** from server
  - Protocol version & Status code =>
  - Response headers
  - Response body

**HTTP status codes:**
- 2xx — all is well
- 3xx — resource moved
- 4xx — access problem
- 5xx — server error
Demo: HTTP via netcat

(SaaS plumbing)
Demo: Sinatra
Sites that are really programs
(SaaS)

• How do you:
  – “map” URI to correct program & function?
  – pass arguments?
  – invoke program on server?
  – handle persistent storage?
  – handle cookies?
  – handle errors?
  – package output back to user?

• Frameworks support these common tasks
Developer environment vs. medium-scale deployment

- **Developer:**
  - file.sqlite3
  - SQLite adapter
  - Rails library
  - rack
  - Webrick

- **Medium-scale deployment:**
  - MySQL
  - MySQL adapter
  - Rails library
  - rack
  - thin
  - Page cache
  - Apache w/mod_rails + caching mode

- **Database cache:**
  - PostgreSQL
  - Database cache
  - "Dynos" running apps
  - HTTP servers & static asset caches

- **Large-scale curated deployment, e.g. Heroku:**
The MVC Design Pattern

- Goal: separate organization of data (model) from UI & presentation (view) by introducing controller
  - mediates user actions requesting access to data
  - presents data for *rendering* by the view
- Web apps may seem “obviously” MVC by design, but other alternatives are possible...

```
Model
  • Read data
  • Update data

View
  • User actions
  • Directives for rendering data

Controller
  • Data provided to views *through controller*
```
• Observation: *HTTP is stateless*

• Early Web 1.0 problem: how to guide a user “through” a flow of pages?
  – use IP address to identify returning user?
    ✗ public computers, users sharing single IP
  – embed per-user junk into URI query string?
    ✗ breaks caching

• Quickly superseded by *cookies*
  – Watch: *screencast.saasbook.info*

• Most frameworks manage tamper-evident cookies for you
Summary: Sinatra demo

• Simple framework provides mechanics for URL parsing, params[] extraction, basic app layout surrounding pages, rendering of HTML-with-embedded-code (erb here)

• session[] gives an easy-to-use wrapper around Cookies to make them look like persistent hash

• You’ll further de-mystify Sinatra in HW#2, and many concepts will transfer to Rails
REST (Representational State Transfer)

- Your app manipulates one or more types of resources.
- Everything the app does can be characterized as some kind of operation on one or more resources.
  - create a new instance of some resource
  - show me resource(s) matching some criterion
  - modify an existing instance of some resource
  - destroy a particular instance of a resource
  - sometimes: CRUD (create/read/update/delete)
How was it?

• What aspects of this lecture did you like/should we do more of?
• …did you dislike/should we do less of?
• Any specific topics from the examples that you’re still confused about/want clarification?
• …do poll on Piazza now!