6.033 Spring 2022
Lecture #1: Complexity, modularity, abstraction
plus an intro to client/server models
what is a system?
“a set of interconnected components that has an expected behavior observed at the interface with its environment.”

what makes building systems difficult?
complexity

why do we care?
complexity limits what we can build
why do we care?
complexity limits what we can build

how do we mitigate complexity?
with design principles such as modularity and abstraction

how do we enforce modularity?
one way is to use a client/server model

the browser is the client in this example

Class Browser
(on machine 1)

```python
def main():
    html = browser_load_url(URL)
    ...
```

```python
def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

stub

Class Server
(on machine 2)

```python
def server_load_url():
    ...
    return html
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(url)
    reply = html
    send reply
```

stub

the browser is the client in this example
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

```python
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
**Problem:** We just bought two copies of item

There are ways to deal with this issue — for example, giving each request a unique ID, and keeping track of those IDs on the server — but then new problems arise: for example, what happens if the server crashes in the middle of handling a request?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?  

**fault-tolerance/reliability:** how does our system deal with failures (☠)? machines crashing, network links breaking, etc.  

**security:** how does our system cope in the face of targeted attacks (😈)?  

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar ((#)?)?  

how do our design and implementation choices impact people and communities? who makes those choices?
http://mit.edu/6.033

has all of the class material, due dates, deadlines, etc.

Piazza

dis this where announcements happen. there will be things that we post on Piazza that aren’t on the website

Canvas

strictly for submitting assignments and seeing your grades. we won’t make announcements or post meaningful content here.

we care about you as people more than we care about any deadline

if you need help ask for it. we need to balance the needs of a large group of students and the needs of the staff, but we will work with you to help as much as we can.
**complexity** limits what we can build, but can be mitigated with design principles such as **modularity** and **abstraction**

One way to **enforce modularity** is with a **client/server model**, where the two modules reside on different machines and communicate with RPCs; network/server failures are still an issue.

You will see these principles applied over and over in this class.

A student once told me that I say “modularity” in almost every lecture, which seems correct.

**Next lecture:** naming, which allows modules to communicate.

**After that:** operating systems, which enforce modularity on a single machine.