The Internet
Wilhem Kautz
CS106A, Stanford University
Based on Slides by Chris Piech and Mehran Sahami
First, a cool demo

Chat Client

Chat Server:

Server running...

```
{'command': 'getMsgs', 'params': {'index': 0}}
{'command': 'getMsgs', 'params': {'index': 0}}
{'command': 'getMsgs', 'params': {'index': 0}}
{'command': 'newMsg', 'params': {'msg': 'Hello world?', 'user': 'Chris'}}
{'command': 'getMsgs', 'params': {'index': 1}}
{'command': 'newMsg', 'params': {'msg': 'Here I am!!', 'user': 'Laura'}}
{'command': 'getMsgs', 'params': {'index': 1}}
{'command': 'newMsg', 'params': {'msg': 'This is fun!', 'user': 'Laura'}}
{'command': 'getMsgs', 'params': {'index': 2}}
{'command': 'newMsg', 'params': {'msg': 'Wahooooo :-)', 'user': 'Chris'}}
{'command': 'getMsgs', 'params': {'index': 3}}
{'command': 'newMsg', 'params': {'msg': 'We are on the internet...', 'user': 'Chris'}}
{'command': 'getMsgs', 'params': {'index': 4}}
{'command': 'newMsg', 'params': {'msg': 'This is like low-budget WhatsApp', 'user': 'Chris'}}
{'command': 'getMsgs', 'params': {'index': 5}}
{'command': 'getMsgs', 'params': {'index': 6}}
{'command': 'getMsgs', 'params': {'index': 6}}
{'command': 'newMsg', 'params': {'msg': 'But we made it, which is cool.', 'user': 'Laura'}}
{'command': 'getMsgs', 'params': {'index': 6}}
{'command': 'getMsgs', 'params': {'index': 0}}
{'command': 'newMsg', 'params': {'msg': 'Hi everyone! Terry here too', 'user': 'Terry'}}
{'command': 'getMsgs', 'params': {'index': 7}}
{'command': 'newMsg', 'params': {'msg': 'Hi Terry!', 'user': 'Laura'}}
{'command': 'getMsgs', 'params': {'index': 7}}
{'command': 'getMsgs', 'params': {'index': 8}}
{'command': 'newMsg', 'params': {'msg': 'The internet is a wild place...', 'user': 'Terry'}}
{'command': 'getMsgs', 'params': {'index': 9}}
{'command': 'getMsgs', 'params': {'index': 9}}
```

Chat Client:

- The Internet is a Wild Place...
- [Chris] Hello world?
- [Laura] Here I am!!
- [Chris] This is fun!
- [Chris] Wahooooo :-)
- [Chris] We are on the internet...
- [Chris] This is like low-budget WhatsApp
- [Chris] But we made it, which is cool.
- [Terry] Hi everyone! Terry here too
- [Laura] Hi Terry!
- [Terry] The internet is a wild place...

Piech + Sahami, CS106A, Stanford University
<review>
class Dog:
    def __init__(self):
        self.times_barked = 0
    
    def bark(self):
        print('woof')
        self.times_barked += 1

def main():
    simba = Dog()
    juno = Dog()
    simba.bark()
    juno.bark()
    simba.bark()
    print(simba.__dict__)
    print(juno.__dict__)
1. What happens when you make a new one?
2. What variables does each instance store?
3. What **methods** can you call on an instance?
Did I mention that a class is like a fancy dictionary?
Classes define new variable *types*
Classes decompose your program across files
Define New Variable Types

Song

Playlist

User

Song Player

Song Retriever

Piech + Sahami, CS106A, Stanford University
</ review>
One reason programming is fun is because of the internet...
Smart Phone Access

Advanced Economies

Emerging Economies

- Smartphone
- Mobile
- No phone
For the fourth time ever in CS106A:
Learning Goals

1. Write a program that can respond to internet requests
How does your phone communicate with Facebook?
The program on your phone talks to the program at Facebook
JavaScript with HTML are the languages of websites.

Kotlin is the language of Android phones.

Swift is the language of Apple phones.

Face Book Server
Is this legit?

wkautz@stanford.edu is now logged in

Face Book Server
Send me the **full name** for wkautz@stanford.edu

"Wilhem Kautz"
Send me the **cover photo** for wkautz@stanford.edu
Send the **profile photo** for
wkautz@stanford.edu

Face Book Server
Send the **status** for wkautz@stanford.edu

"chillin"
Set the **status** for wkautz@stanford.edu to be “lecturing”
Send me the status for wkautz@stanford.edu

Face Book Server

“lecturing”

Wilhem Kautz

Status: Wil is lecturing

Set status:
The internet is just many programs sending messages (as Strings)
Background: The Internet

The internet is just many programs sending messages (as Strings)
The internet is just many programs sending messages (as Strings)
The internet is just many programs sending messages (as Strings)
The internet is just many programs sending messages (as Strings)
There are two types of internet programs. Servers and Clients
Internet 101
Computers on the internet
Servers are computers (running code)

Face Book Server

= 

Image of a Facebook server room
Facebook’s closest datacenter is here.
Get status for wkautz@stanford.edu
The Internet
teaching

The Internet

Face Book Server
The Internet
Many computers can connect to the same server.
The Internet

Facebook datacenter

REQUEST

Chris's computer (facebook.com)

RESPONSE

Wil's phone (facebook app)

REQUEST

Your mom's computer (linux shell)

RESPONSE

"Client"

"Server"

"Client"
Most of the Internet

Server / Clients

Aka “the backend”

Aka “the cloud”

Aka “the brains”
Most of the Internet

Server / Clients

Aka “the backend”
Aka “the cloud”
Aka “the brains”
Aka “the frontend”
Aka “the GUI”
Today, the server
A server’s main job is to respond to requests
A Server’s Simple Purpose

Request
From a client

Response
To the client

Server
A Server’s Simple Purpose

Request
someRequest

String
serverResponse

ChatServer
Starting server on port 8080...
getMsgs
newMsg
Added new message
getMsgs
Returned 1 messages
getMsgs
Returned 1 messages
newMsg
Added new message
getMsgs
Returned 1 messages
getMsgs
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

# turn on the server
def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

# enjoy
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

# enjoy
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)

# enjoy
# handle server requests (must be in a class)

def handle_request(self, request):
    # return a string response!

# turn on the server

def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

# enjoy
Servers on one slide

1. # handle server requests (must be in a class)
   ```python
def handle_request(self, request):
    # return a string response!
   ```

2. # turn on the server
   ```python
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)
   ```

3. # enjoy
### 1

```python
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

### 2

```python
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)
```

### 3

```python
# enjoy
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

    # enjoy
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

# turn on the server

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

# enjoy
Servers on one slide

1. # handle server requests (must be in a class)
   ```python
def handle_request(self, request):
    # return a string response!
```

2. # turn on the server
   ```python
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)
```

3. # enjoy
What is a Port?

80 is a special port for a server that wants to talk to web browsers.
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!

# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)

# enjoy
What is a Request?

/* Request has a command */
command (type is string)

/* Request has parameters */
params (type is dict)

// methods that the server calls on requests
request.command
request.params
class Request:
    '''
    The request class packages the key information from an internet request. 
    An internet request has both a command and a dictionary of parameters. 
    This class defines a special function __str__ which means if you have
    instance of a request you can put it in a print function.
    '''

def __init__(self, request_command, request_params):
    # every request has a command (string)
    self.command = request_command
    # every request has params (dictionary). Can be {}
    self.params = request_params

def get_params(self):
    # a 'getter' method to get the params
    return self.params

def get_command(self):
    # a 'getter' method to get the command
    return self.command

def __str__(self):
    # a special method which says what happens when you 'print' a re
    return 'command=' + self.command + ' params=' + str(self.params)
from SimpleInternet import run_server
import json

class MyServer:
    def __init__(self):
        ''' You can store data in your server! '''
        pass

    # this is the server request callback function.
    def handle_request(self, request):
        ''' This function gets called every time someone makes a
        request to our server.''
        return 'hello world'

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server to handle internet requests!
    run_server(handler, 8000)
Who makes requests?
Who makes requests?

Other programs can send requests!

response = requests.get('https://xkcd.com/353/')
Who makes requests?

Other programs can send requests!

response = requests.get('https://xkcd.com/353/)

Web browsers can send requests!
Anatomy of a Browser Request
The protocol. Usually http or https
Anatomy of a Browser Request

The webaddress of the computer that will respond to the request.
Anatomy of a Browser Request

The request command
First Server Example!

```python
from SimpleInternet import run_server
import json

class MyServer:
    def __init__(self):
        ''' You can store data in your server! '''
        pass

    # this is the server request callback function.
    def handle_request(self, request):
        ''' This function gets called every time someone makes a request to our server.'''
        return 'hello world'

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server to handle internet requests!
    run_server(handler, 8000)
```

Piech + Sahami, CS106A, Stanford University
Hit Counter

6668934
Recall Requests

/* Request has a command */
command (string)

/* Request has parameters */
params (dict)

// methods that the server calls on requests
request.command
request.params
Requests are like Remote Method Calls

Server has a bunch of discrete things it can do:
- make_toast
- blend
Requests are like Remote Method Calls

Server has a bunch of discrete things it can do

get_status

add_user
Requests are like Remote Method Calls

Server

get_status

add_user
Requests are like Remote Method Calls

request.get_command() => "get_status"

get_status

add_user
Requests are like Remote Method Calls

To make toast, I need a parameter which is the kind of bread

get_status
Requests are like Remote Method Calls

I was given a parameter!

get_status
Requests are like Remote Method Calls

request.params["userName"]

get_status
Requests are like Remote Method Calls

get_status
Requests are like Remote Method Calls

get_status
Requests are like Remote Method Calls
```python
def handle_request(self, request):
    cmd = request.command
    if cmd == 'get_status':
        user = request.params['userName']
        status = self.get_status(user)
        return status
```

Must be a string!
Requests are like Remote Method Calls
Requests are like Remote Method Calls
Requests are like Remote Method Calls
Requests are like Remote Method Calls
Requests are like Remote Method Calls
Requests are like Remote Method Calls
Requests responses are strings, often encoded using JSON
Recall JSON

import json

# load data
data = json.load(open('ages.json'))

# save data
json.dump(data, open('ages.json'))
Recall JSON

import json

# load data
data = json.load(open('ages.json'))

# save data
json.dump(data, open('ages.json'))
Recall JSON

import json

# load data
data = json.load(open('ages.json'))

# save data
json.dump(data, open('ages.json'))

# write a variable to a string
data_str = json.dumps(data)
Time for a little chat
Chat Server and Client

Chat Client

Server running...

- {'command': 'getMsgs', 'params': {'index': '0'}}
- {'command': 'newMsg', 'params': {'msg': 'Hello world?', 'user': 'Chris'}}
- {'command': 'getMsgs', 'params': {'index': '0'}}
- {'command': 'getMsgs', 'params': {'index': '0'}}
- {'command': 'newMsg', 'params': {'msg': 'Here I am!!', 'user': 'Laura'}}
- {'command': 'getMsgs', 'params': {'index': '1'}}
- {'command': 'newMsg', 'params': {'msg': 'This is fun!', 'user': 'Laura'}}
- {'command': 'getMsgs', 'params': {'index': '2'}}
- {'command': 'getMsgs', 'params': {'index': '1'}}
- {'command': 'newMsg', 'params': {'msg': 'Wahooooo :-)', 'user': 'Chris'}}
- {'command': 'getMsgs', 'params': {'index': '3'}}
- {'command': 'newMsg', 'params': {'msg': 'We are on the internet...', 'user': 'Chris'}}
- {'command': 'getMsgs', 'params': {'index': '4'}}
- {'command': 'newMsg', 'params': {'msg': 'This is like low-budget WhatsApp', 'user': 'Chris'}}
- {'command': 'getMsgs', 'params': {'index': '5'}}
- {'command': 'getMsgs', 'params': {'index': '3'}}
- {'command': 'getMsgs', 'params': {'index': '6'}}
- {'command': 'getMsgs', 'params': {'index': '6'}}
- {'command': 'newMsg', 'params': {'msg': 'But we made it, which is cool.', 'user': 'Laura'}}
- {'command': 'getMsgs', 'params': {'index': '6'}}
- {'command': 'getMsgs', 'params': {'index': '6'}}
- {'command': 'newMsg', 'params': {'msg': 'Hi everyone! Terry here too', 'user': 'Terry'}}
- {'command': 'getMsgs', 'params': {'index': '7'}}
- {'command': 'newMsg', 'params': {'msg': 'Hi Terry!', 'user': 'Laura'}}
- {'command': 'getMsgs', 'params': {'index': '7'}}
- {'command': 'getMsgs', 'params': {'index': '8'}}
- {'command': 'newMsg', 'params': {'msg': 'The internet is a wild place...', 'user': 'Terry'}}
- {'command': 'getMsgs', 'params': {'index': '9'}}
- {'command': 'getMsgs', 'params': {'index': '9'}}
addMsg
{
    'msg': 'Hello world',
    'user': 'C'
}
history = [
    '[C] Hello world'
]

getMsgs
{
    'index' : 0
}
history = [
    '[C] Hello world'
]

'"[C] Hello world"'
```python
history = [
    ['[C] Hello world'],
]

addMsg = {
    'msg': 'I'm here too',
    'user': 'B',
}
```

Chat Client

```
> [C] Hello world

Send
```

```
I'm here too
```

Send
history = [
  '[C] Hello world',
  '[B] I'm here too'
]

'Got it'
history = [
    '[C] Hello world',
    '[B] I am here too'
]

getMsgs
{
    'index' : 1
}
history = [
    '[C] Hello world',
    '[B] Im here too'
]

'"[B] Im here too"']"
history = [
    '[C] Hello world',
    '[B] I'm here too'
]

def getMsgs:
    return {'index': 0}
history = [
    '[C] Hello world',
    '[B] I'm here too'
]

'"[C] Hello world",
"[B] I'm here too"]'
Chat Server

addMsg
msg = text
user = user

getMsgs
index = start_index
Learning Goals

1. Write a program that can respond to internet requests
Things we saw along the way

```
data_str = json.dumps(data)
```