The Internet

Thanks to Mehran Sahami and Chris Piech for today’s slides!
• Assignment #6 has been released!

• Remaining lecture schedule
  – Friday: last content lecture!
  – Tuesday: Life after 106A + AMA
  – Wednesday: Final review
  – Thursday: no lecture, OH during lecture time

• Acknowledging end-of-quarter stress
  – Take care of yourselves and each other
  – Be proud of what you have accomplished!
Today

• Review classes
  – Very briefly

• Let’s talk about the internet!
  – How the internet works at a high level
  – Learn how we can use classes to implement a key part of it (make a server!)
<review>
Dog.py

```python
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```python
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```
```python
def main():
simba = Dog()
juno = Dog()
simba.bark()
juno.bark()
simba.bark()
print(simba.__dict__)
print(juno.__dict__)
```

def __init__(self):
    self.times_barked = 0

def bark(self):
    print('woof')
    self.times_barked += 1

class Dog:
```

1. What happens when you make a new one?
2. What **variables** does each instance store?
2. What **methods** can you call on an instance?
def main():
    simba = Dog()
    juno = Dog()
    simba.bark()
    juno.bark()
    simba.bark()
    print(simba.__dict__)
    print(juno.__dict__)

class Dog:
    def __init__(self):
        self.times_barked = 0
    def bark():
        print('woof')
        self.times_barked += 1

Terminal:
{ times_barked : 2 }
{ times_barked : 1 }

Did I mention that a class is like a fancy dictionary?
Classes define new variable types
Classes decompose your program across files
</review>
One reason programming is fun is because of the internet...
Smart Phone Access

Advanced Economies

- Smartphone: 76
- Mobile: 17
- No phone: 6

Emerging Economies

- Smartphone: 45
- Mobile: 33
- No phone: 17

Piech + Sahami, CS106A, Stanford University
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
Facebook Server

JavaScript with HTML are the languages of websites

Kotlin is the language of Android phones

Swift is the language of Apple phones
Is this authenticated login?

Facebook Server

sahami@cs.stanford.edu is now logged in

sahami@cs.stanford.edu
Send me the **full name** for

```
sahami@cs.stanford.edu
```

"Mehran Sahami"
Send me the **cover photo** for

{sahami@cs.stanford.edu}

Facebook Server
Send the profile photo for sahami@cs.stanford.edu

Facebook Server

Mehran Sahami
Send the status for sahami@cs.stanford.edu

Facebook Server

“teaching”
Set the status for sahami@cs.stanford.edu to be “eating”
Send me the status for sahami@cs.stanford.edu

Facebook Server

"eating"
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)
Background: The Internet

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

The internet is just many programs sending messages (as *Strings*)

Facebook datacenter

“Server”

“Being super TA”

“response”

Your computer (facebook.com)

“Client”
The internet is just many programs sending messages (as *Strings*)
There are (generally) two types of internet programs: Servers and Clients
Internet 101
Computers on the internet
Servers are computers (running code)
Facebook’s closest datacenter is here
The Internet

Facebook Server
The Internet

Facebook Server

Get status for sahami@cs.stanford.edu
Facebook Server

The Internet
Facebook Server

The Internet

eating
Facebook Server

The Internet
Many computers can connect to the same server
The Internet

Your computer (facebook.com)

Mehran’s phone (facebook app)

Facebook datacenter

“Server”

Your mom’s computer (linux shell)

REQUEST

RESPONSE

REQUEST

RESPONSE

“Client”
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
# 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 = 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()  # has handleRequest method

    # 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
# 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
What is a Port?
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 = MyServer()

    # 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
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
```
def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    SimpleServer.run_server(handler, 8000)
    # enjoy
```
class Request:
    ""
    Request class packages the key information from an internet request.
    ""

def __init__(self, request_command, request_params):
    # Every request has a command (string)
    self.command = request_command
    # Every request has params (dictionary). Can be empty: {}. 
    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 allows you to 'print' a request as a string.
        return str(self.dict)
import SimpleServer

# We define a class to handle server requests class
class MyFirstServer:
    def __init__(self):
        pass

    # This is the server request callback function.
def handle_request(self, request):
        print(request)
        return 'Happy Thursday, wonderful cs106a!!'

def main():
    # Make the server handler
    handler = MyFirstServer()
    # Start the server to handle internet requests at specified port
    SimpleServer.run_server(handler, 8000)
Who makes requests?
Who makes requests?

Other programs can send requests!

Web browsers can send requests!
Anatomy of a Browser Request
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
Anatomy of a Browser Request

The request params
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

get_status

add_user

Server
Requests are like Remote Method Calls

```
request.get_command()
=> "get_status"
```

Server

- `get_status`
- `add_user`

Get status

Add user
Requests are like Remote Method Calls

To make toast, I need a parameter

get_status
Requests are like Remote Method Calls

I was given a parameter!

request.params["userName"]

get_status
Requests are like Remote Method Calls

sahami

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
    if cmd == 'add_user':
        user = request.params['userName']
        status = self.add_user(user)
```
Time for a little chat
Chat Server and Client

Chat Client

Send: The internet is a wild place...

Messages

Return:

| [Chris] | Hello world? |
| [Laura] | Here I am! |
| [Chris] | This is fun! |
| [Chris] | Hi everyone! Terry here too |
| [Laura] | Hi Terry! |
| [Chris] | The internet is a wild place... |

Server running...

{'command': 'getMsgs', 'params': {'index': '0'}}

{'command': 'newMsg', 'params': {'msg': 'Hello world?', 'user': 'Chris'}}

{'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': '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': 'newMsg', 'params': {'msg': 'It is cool.', 'user': 'Chris'}}

{'command': 'getMsgs', 'params': {'index': '6'}}

{'command': 'newMsg', 'params': {'msg': 'But we made it, which is cool.', 'user': 'Laura'}}

{'command': 'getMsgs', 'params': {'index': '8'}}

{'command': 'newMsg', 'params': {'msg': 'Hi everyone! Terry here too', 'user': 'Terry'}}

{'command': 'getMsgs', 'params': {'index': '9'}}

{'command': 'newMsg', 'params': {'msg': 'Hi Terry!', 'user': 'Laura'}}

{'command': 'getMsgs', 'params': {'index': '7'}}

{'command': 'newMsg', 'params': {'msg': 'The internet is a wild place...', 'user': 'Terry'}}

{'command': 'getMsgs', 'params': {'index': '9'}}
addMsg
{
    'msg' : Hello world,
    'user' : 'C'
}

```python
history = []
```

![Chat Client](image)
history = 
    ['[C] Hello World']

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

'"[C] Hello world"']

Chat Client

> [C] Hello world

Send

Chat Client
history = ['[C] Hello World']

addMsg {
    'msg': 'Im here too'
    'user': 'B'
}

Chat Client
> [C] Hello world

Chat Client
Im here too  Send
history = [
    '[C] Hello world',
    '[B] Im here too'
]

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

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

'[["[B] Im here too"]]'
getMsgs
{
    'index': 0
}

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

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

Chat Client
> [C] Hello world
> [B] Im here too

Chat Client
> [C] Hello world
> [B] Im here too
addMsg
msg = text
user = user

getMsgs index = start_index
Bajillion Extension

Search Engine

Piech + Sahami, CS106A, Stanford University
Recap

• The internet is just a bunch of connected computers

• Each computer is either a server or a client

• We can use classes in Python to make servers!

• A server’s one job is to handle requests
Bonus
Internet sends data as strings…

How do you send a list or a dictionary?
Requests responses are strings, often encoded using JSON
```python
import json

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

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

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

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