

A photograph of a Stanford University campus scene. In the foreground, a person wearing a helmet and a backpack is riding a bicycle from left to right. The background features a large, modern building with a light-colored facade and dark-framed windows. Several tall palm trees are planted in concrete planters along the sidewalk. Bicycles are parked in racks near the building. The sky is blue with some light clouds.

Internet

Neel Kishnani, CS106A, Summer 2025

Stanford | ENGINEERING

**Credit to Mehran Sahami
and Chris Gregg for these
awesome slides!**



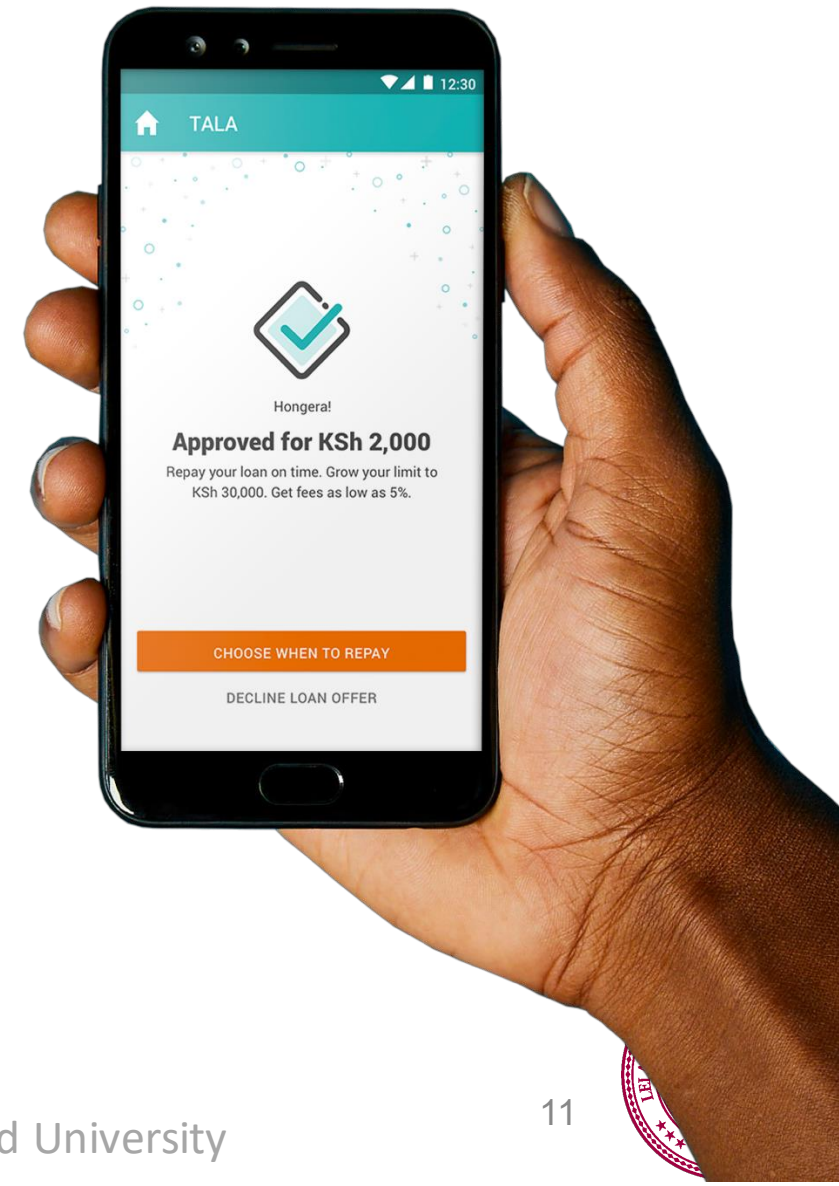
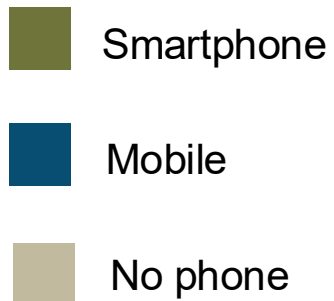
One reason programming is
fun is because of the
internet...

Smart Phone Access

Advanced Economies



Emerging Economies



Learning Goals

1. Write a program that can respond to internet requests



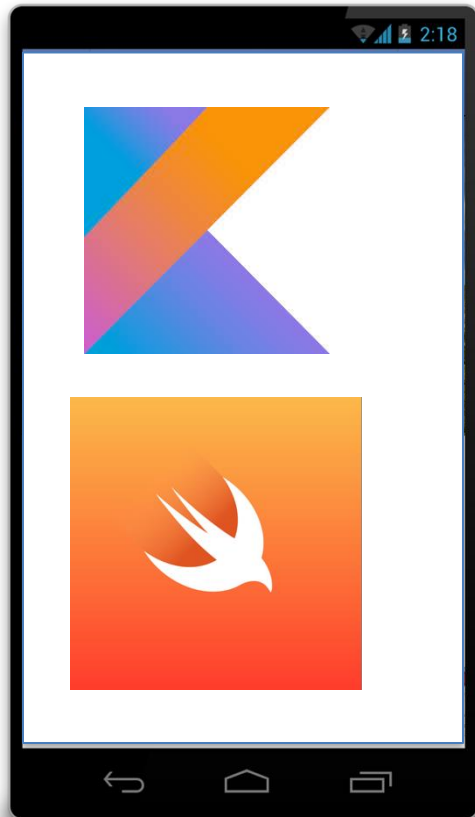
How does your phone
communicate with instagram?

The program on your **phone**
talks to the program at
Instagram

JavaScript
with HTML
are the
languages
of websites

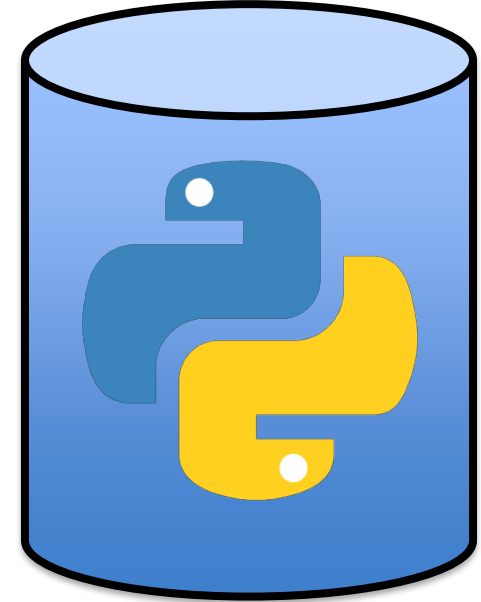


Kotlin is the
language
of Android
phones



Swift is the
language
of Apple
phones

Instagram Server

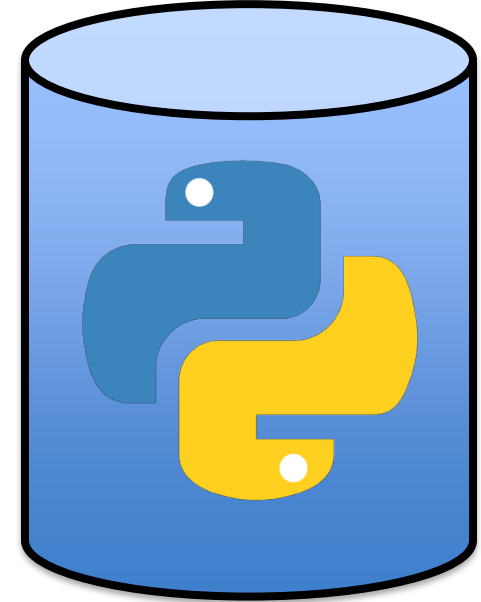




Send the
profile photo
for
taylorswift



Instagram Server

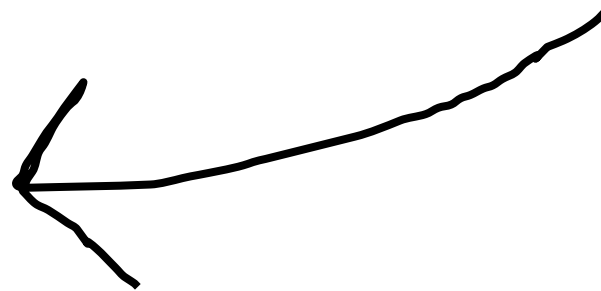
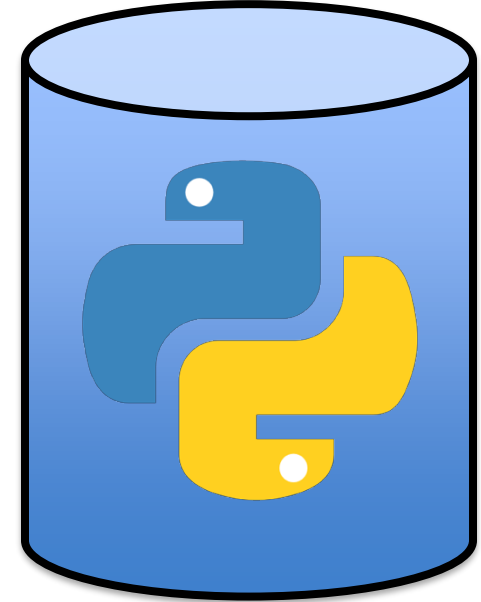


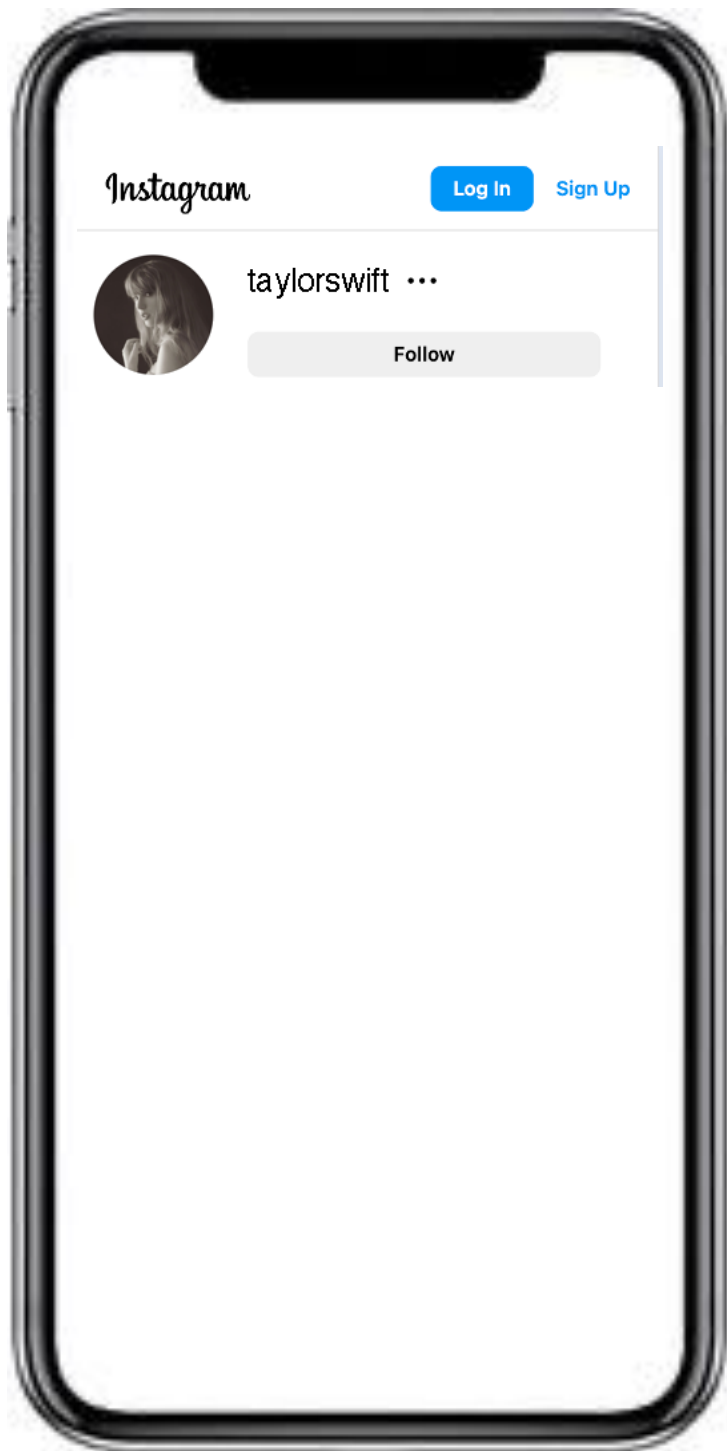


Send the
profile photo
for
taylorswift



Instagram Server

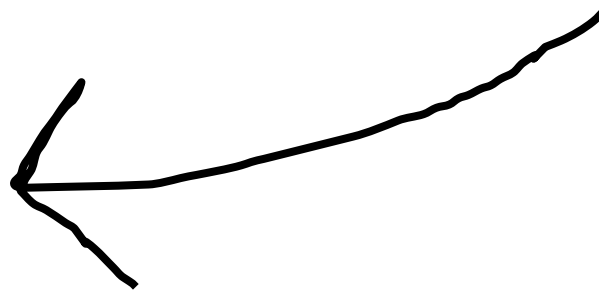
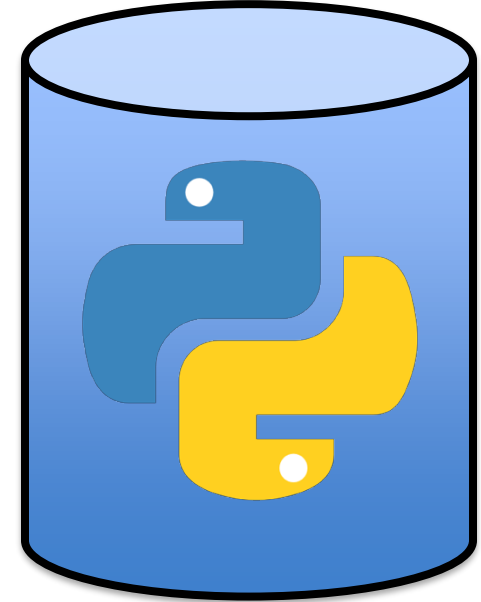


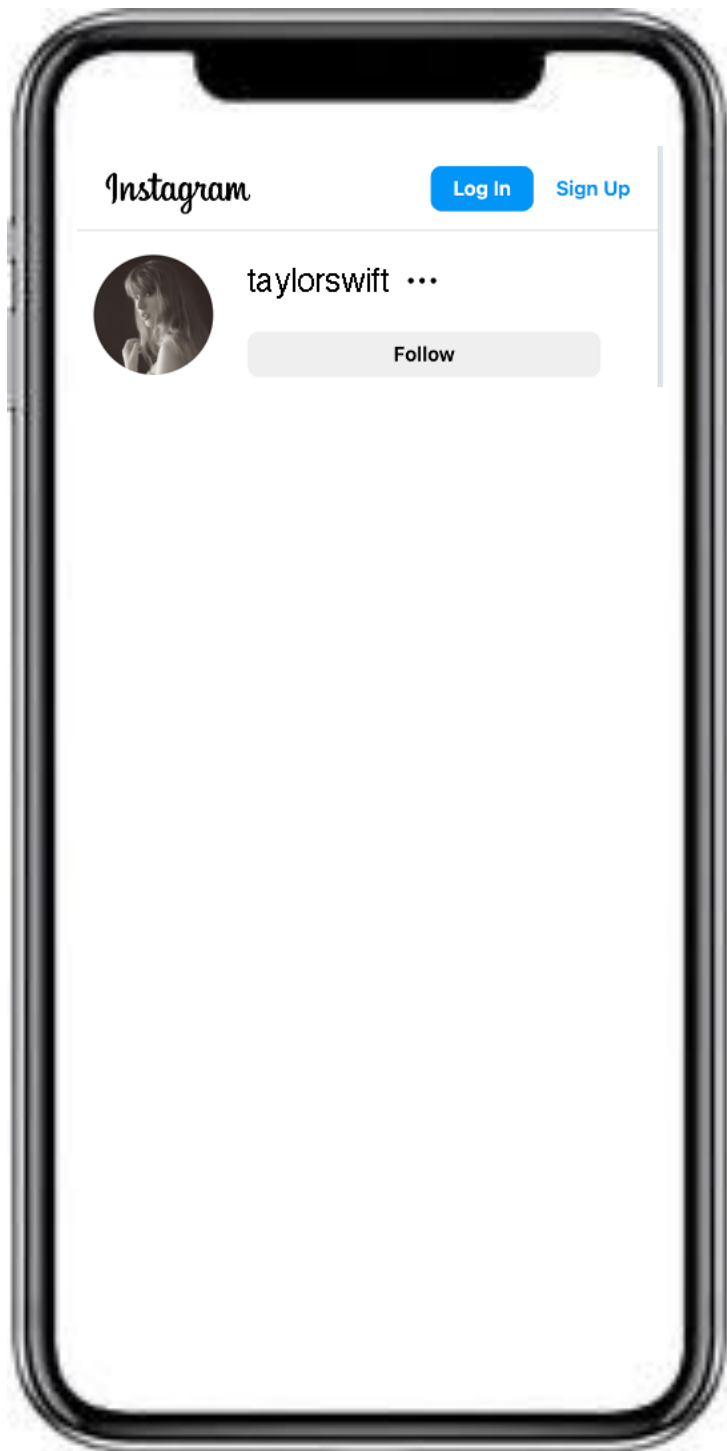


Send the
profile photo
for
taylorswift

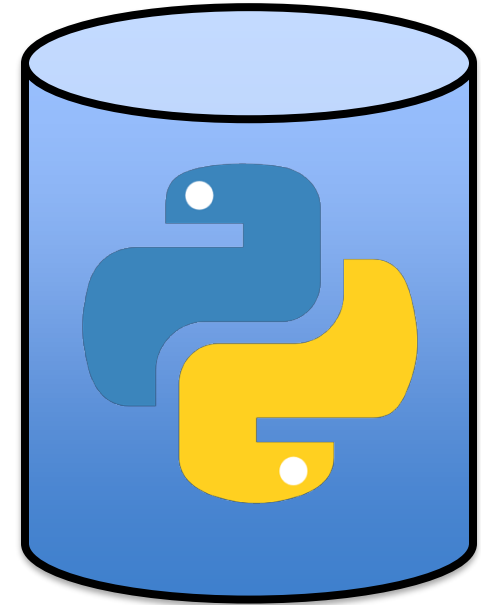


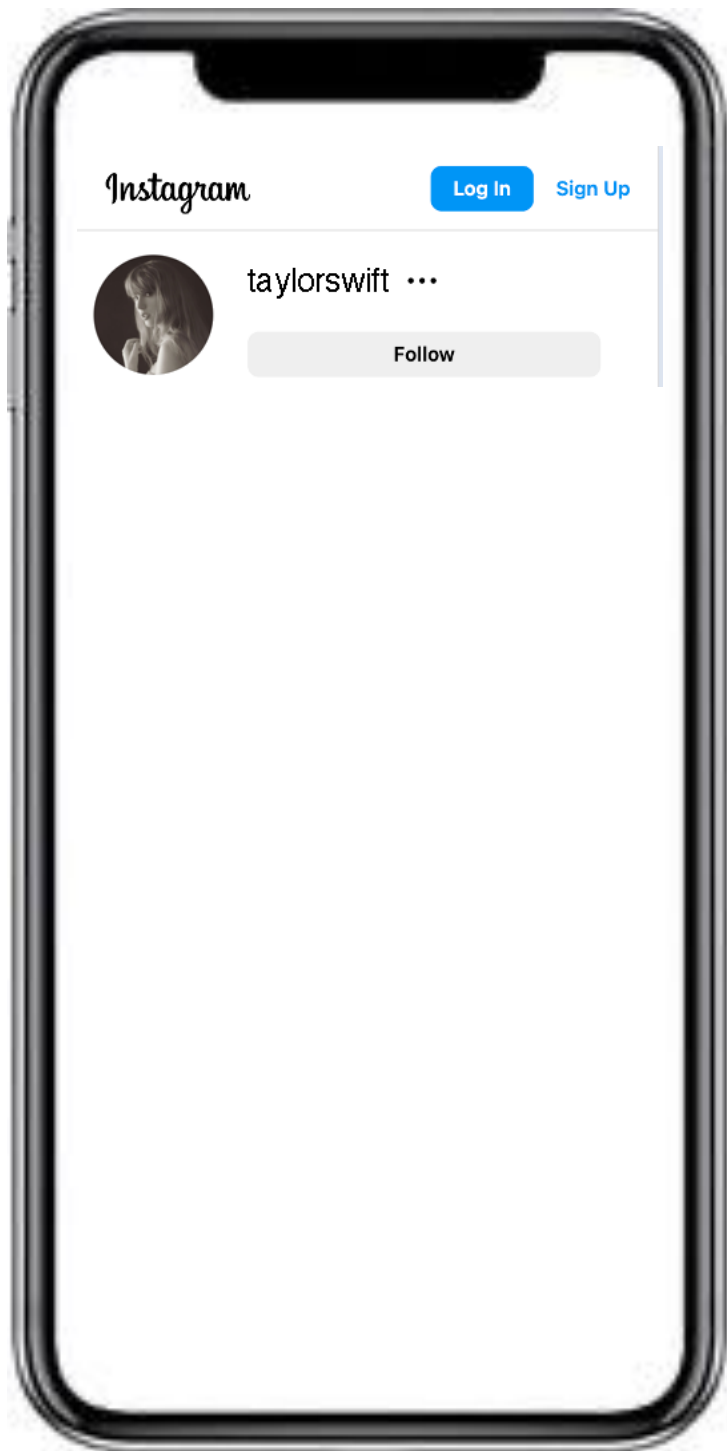
Instagram Server





Instagram Server

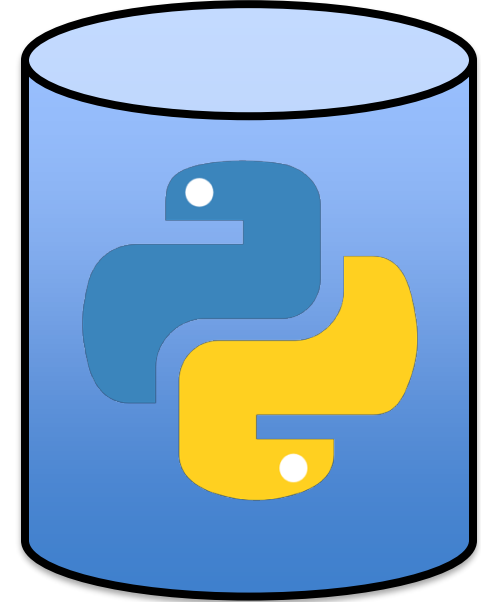


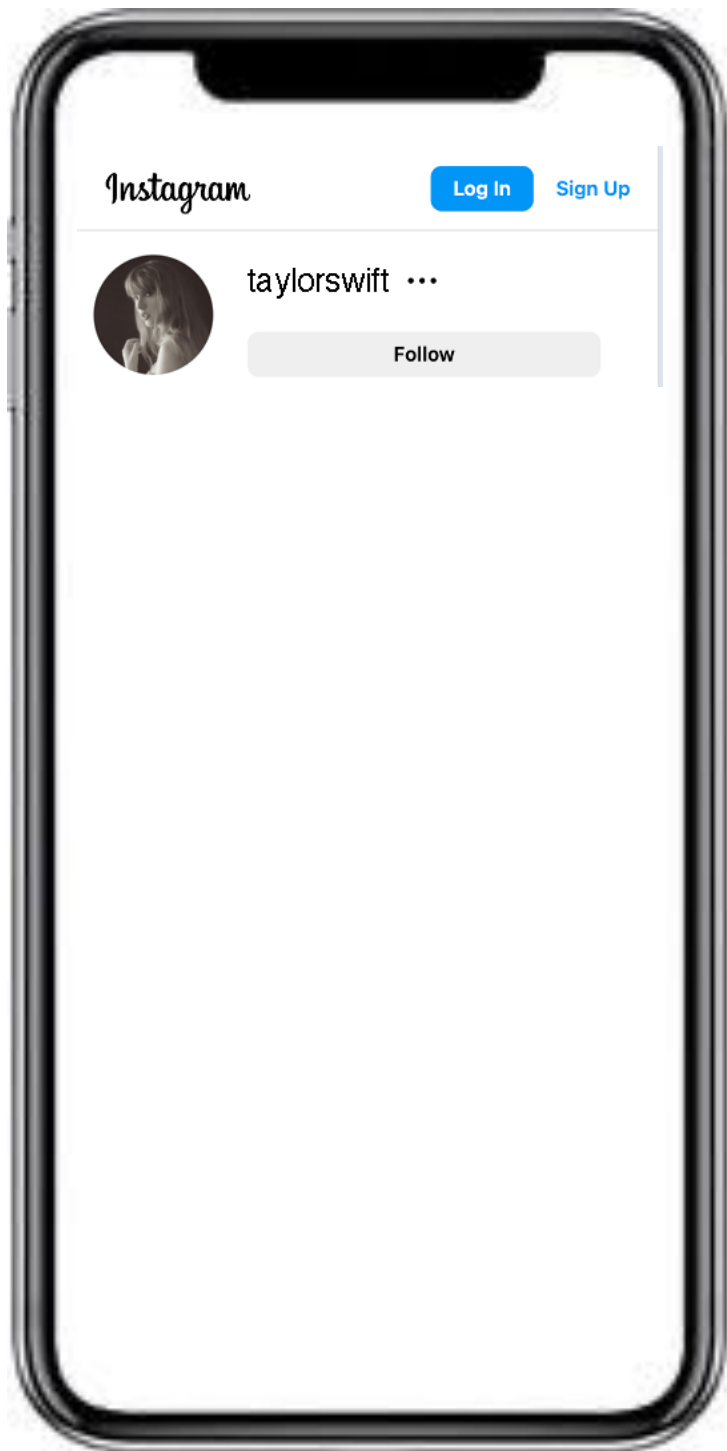


Send the
user info for
taylorswift



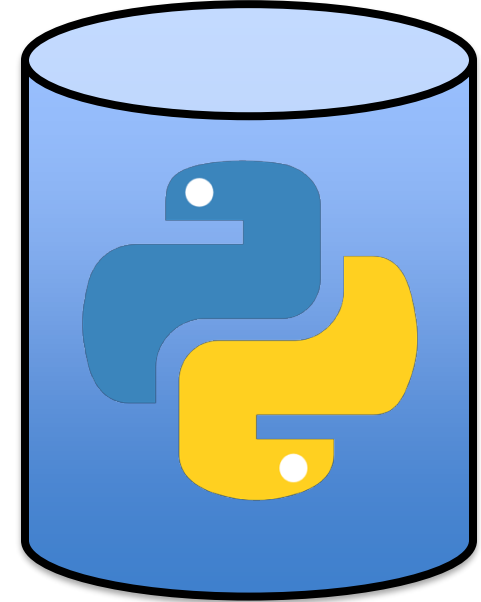
Instagram Server





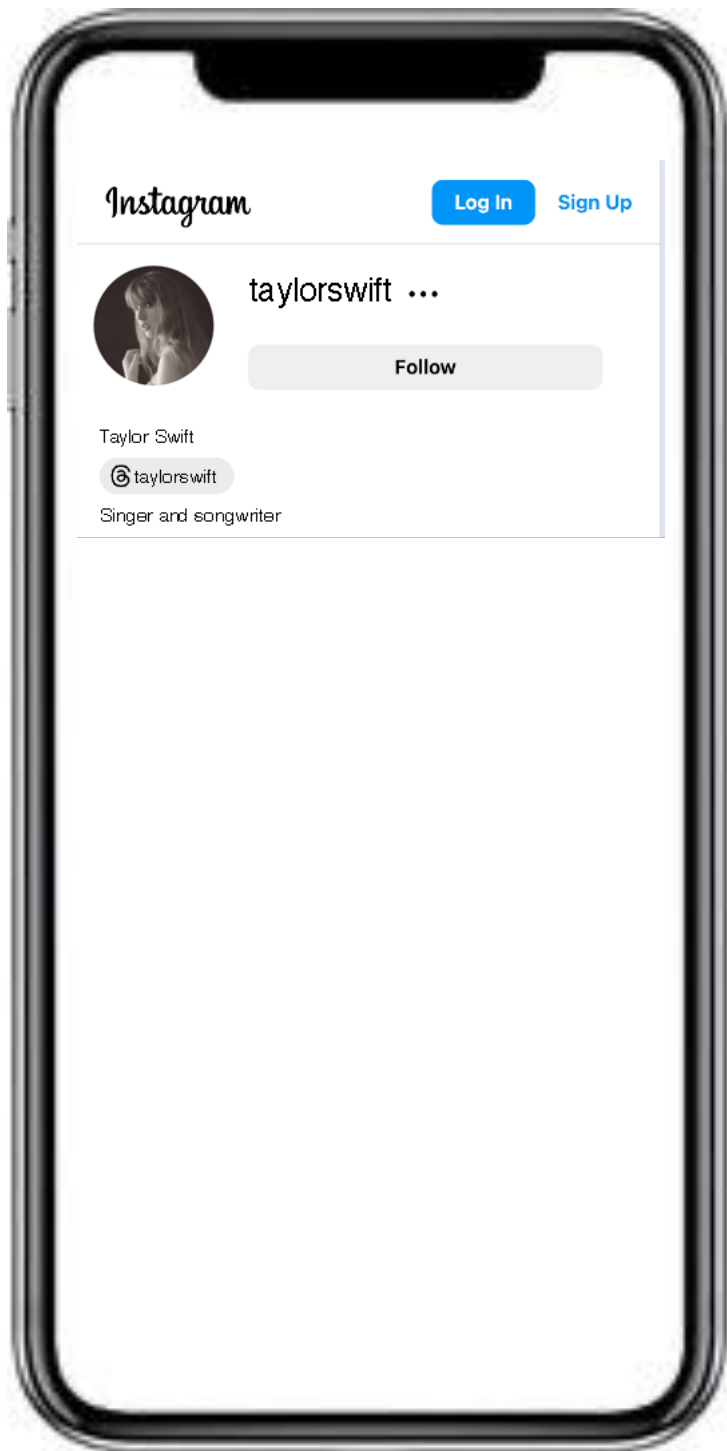
Send the
user info for
taylorswift

Instagram Server



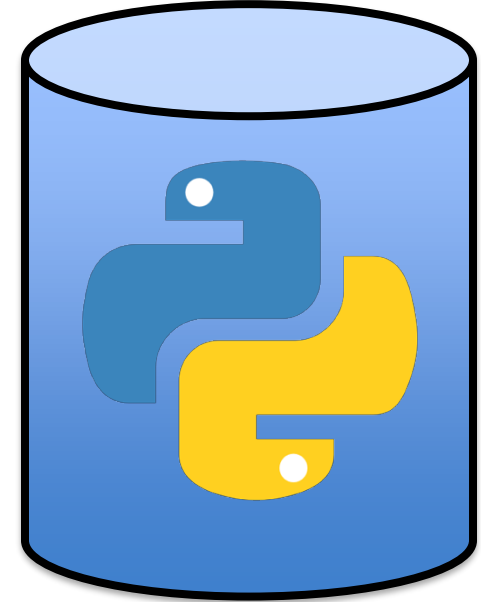
```
{  
  "name": "Taylor Swift",  
  "bio" : "Singer  
          and songwriter"  
}
```





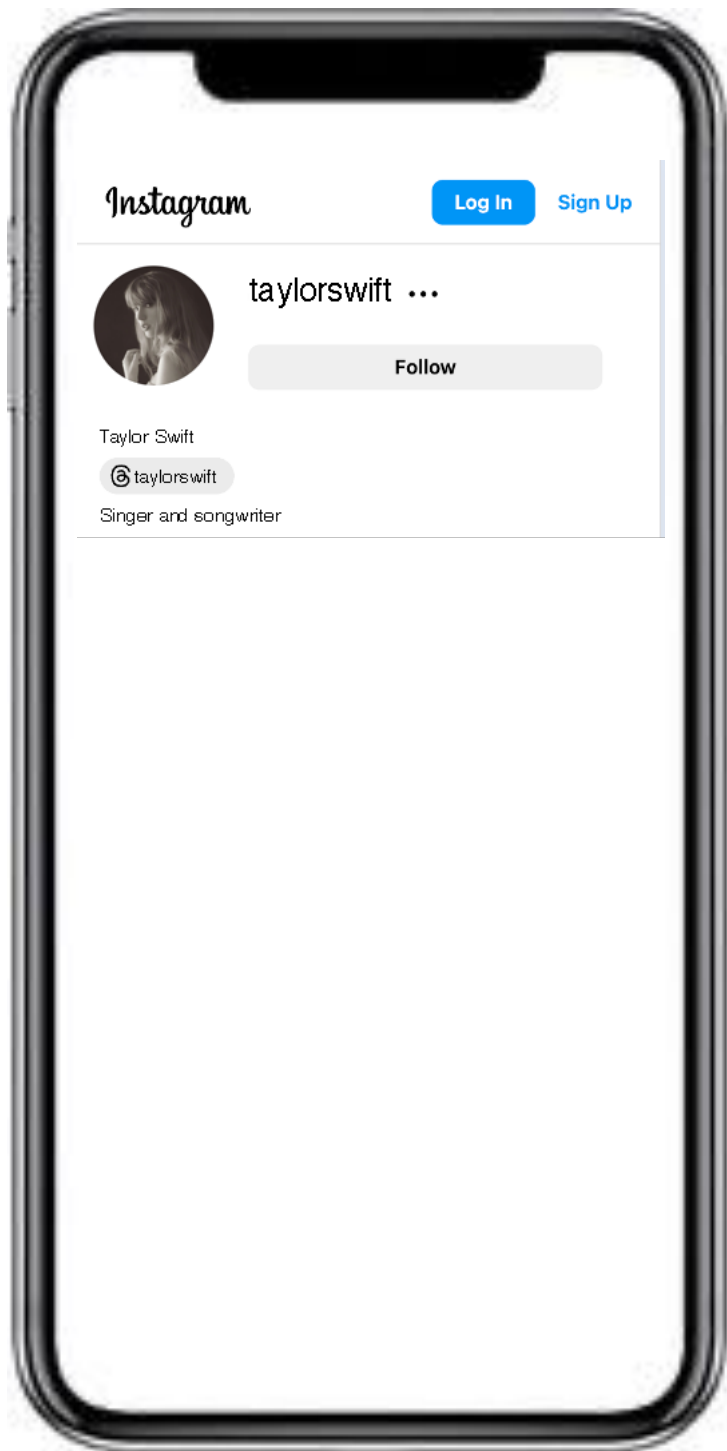
Send the
user info for
taylorswift

Instagram Server

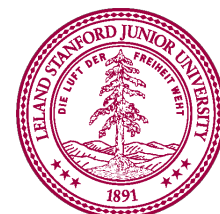


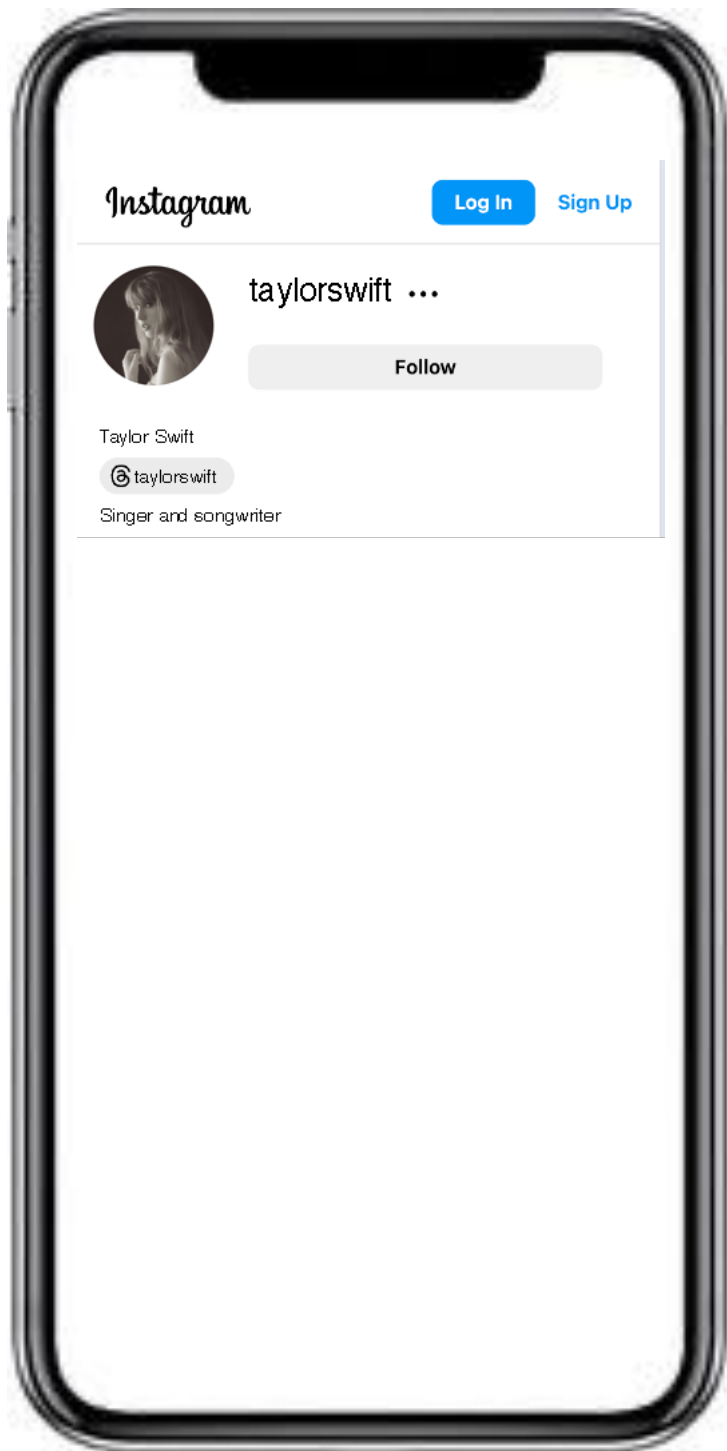
```
{  
  "name": "Taylor Swift",  
  "bio" : "Singer  
          and songwriter"  
}
```





Instagram Server

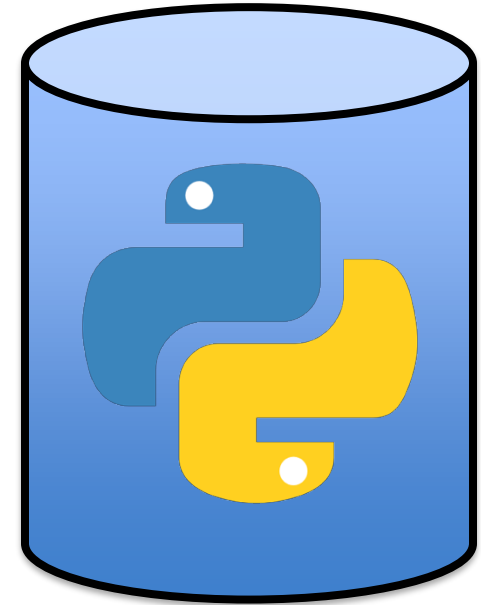


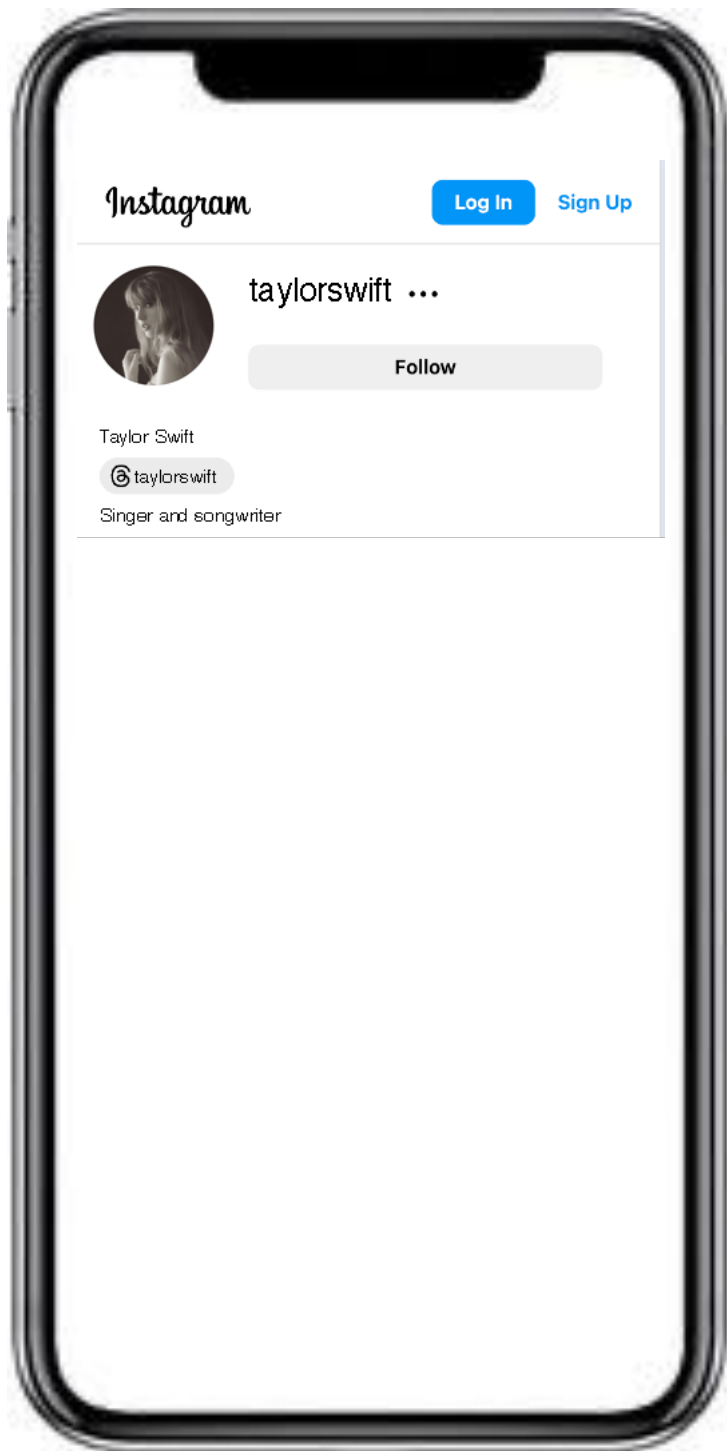


Send the
posts for
taylorswift



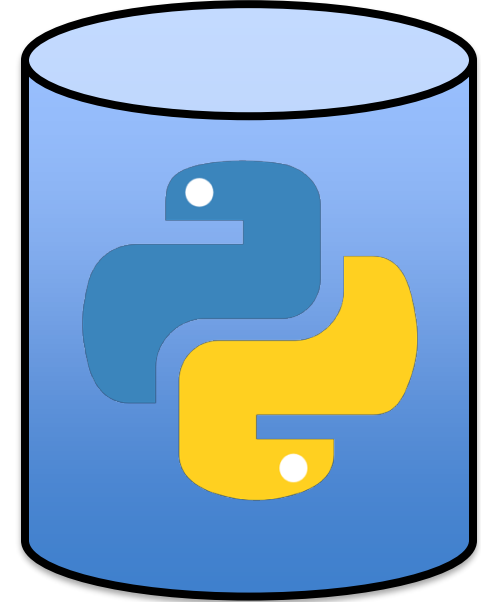
Instagram Server





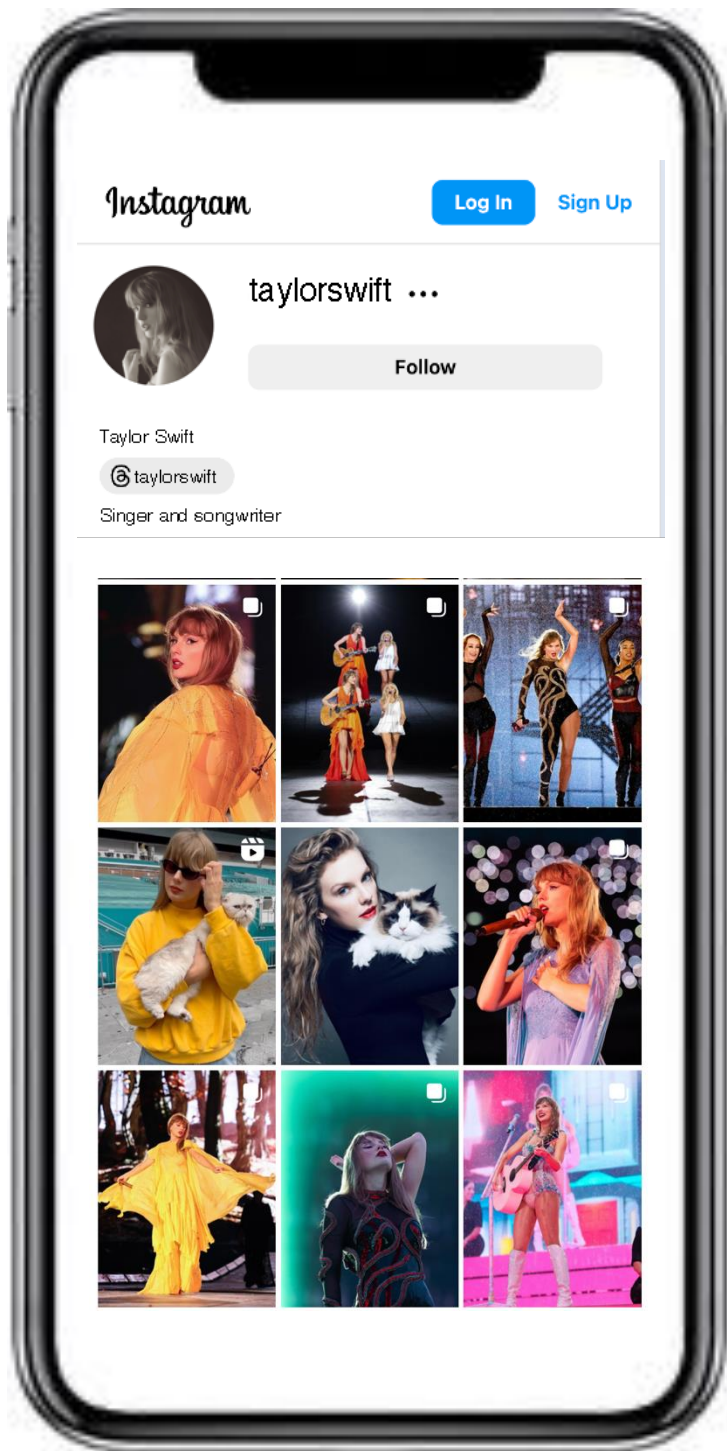
Send the
posts for
taylorswift

Instagram Server



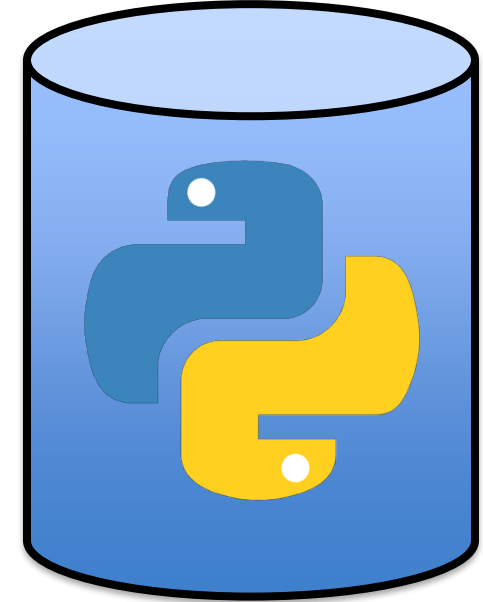
[
 "https://cdninstagram.com/CHvCmgPsJdO",
 "https://cdninstagram.com/Aslkj23dser",
 ...
]





Send the
posts for
taylorswift

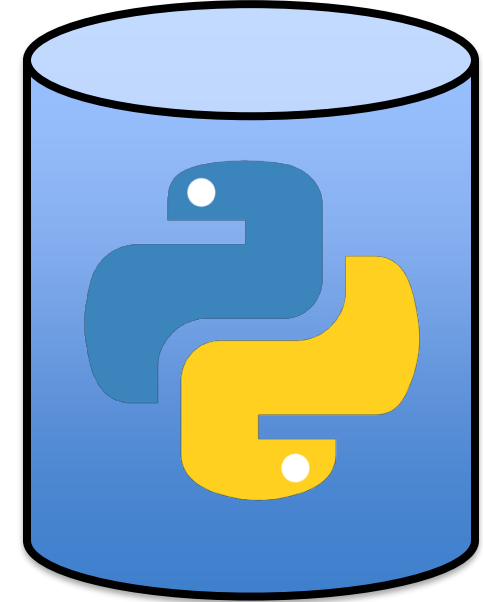
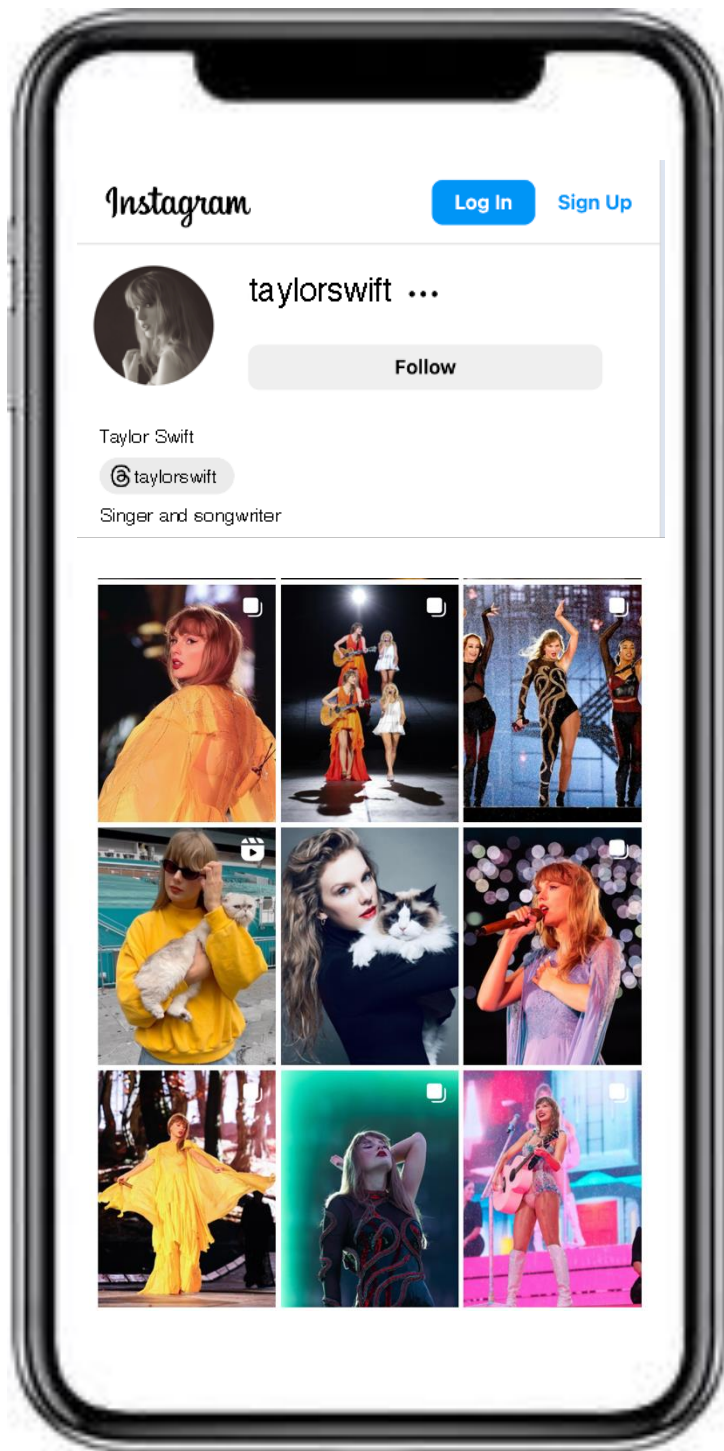
Instagram Server

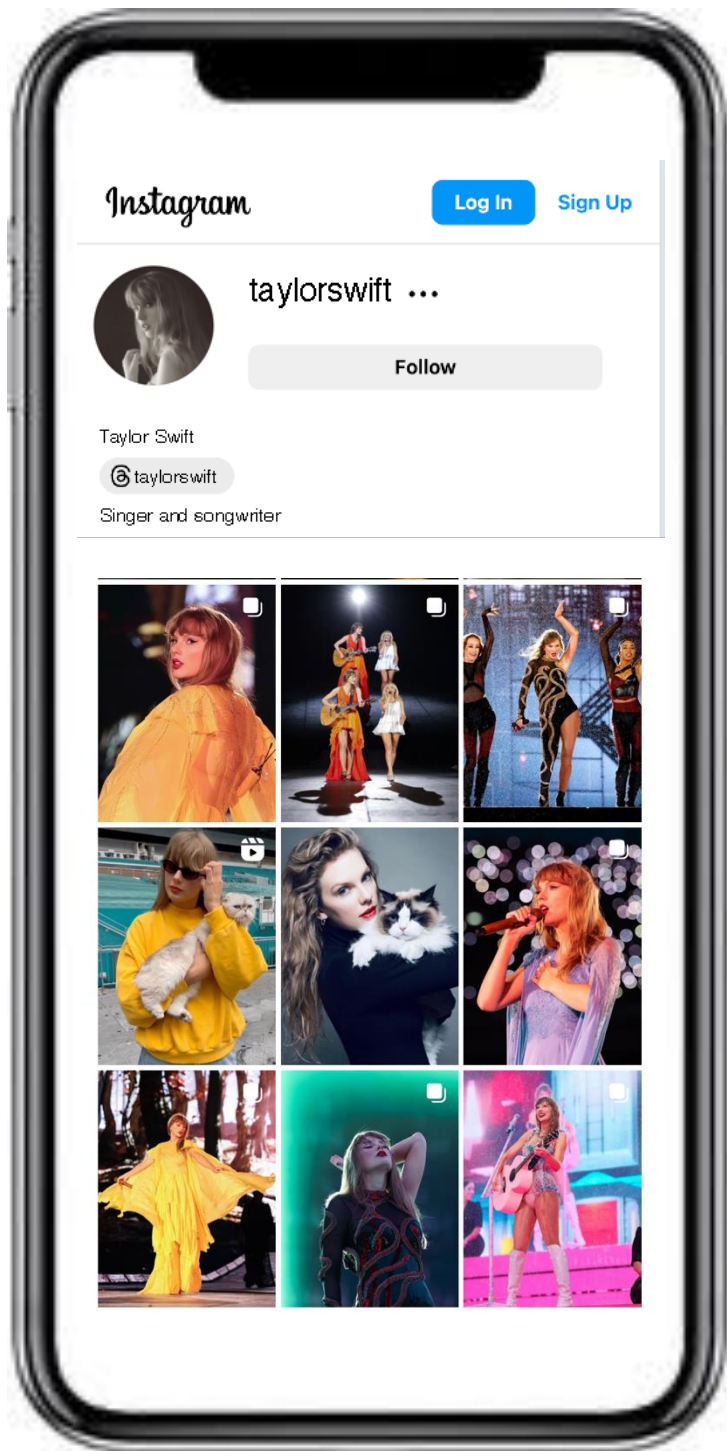


[
 "https://cdninstagram.com/CHvCmgPsJdO",
 "https://cdninstagram.com/Aslkj23dser",
 ...
]



Instagram Server

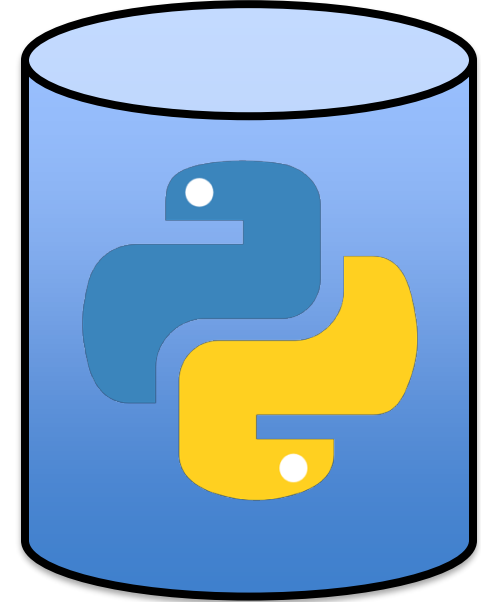


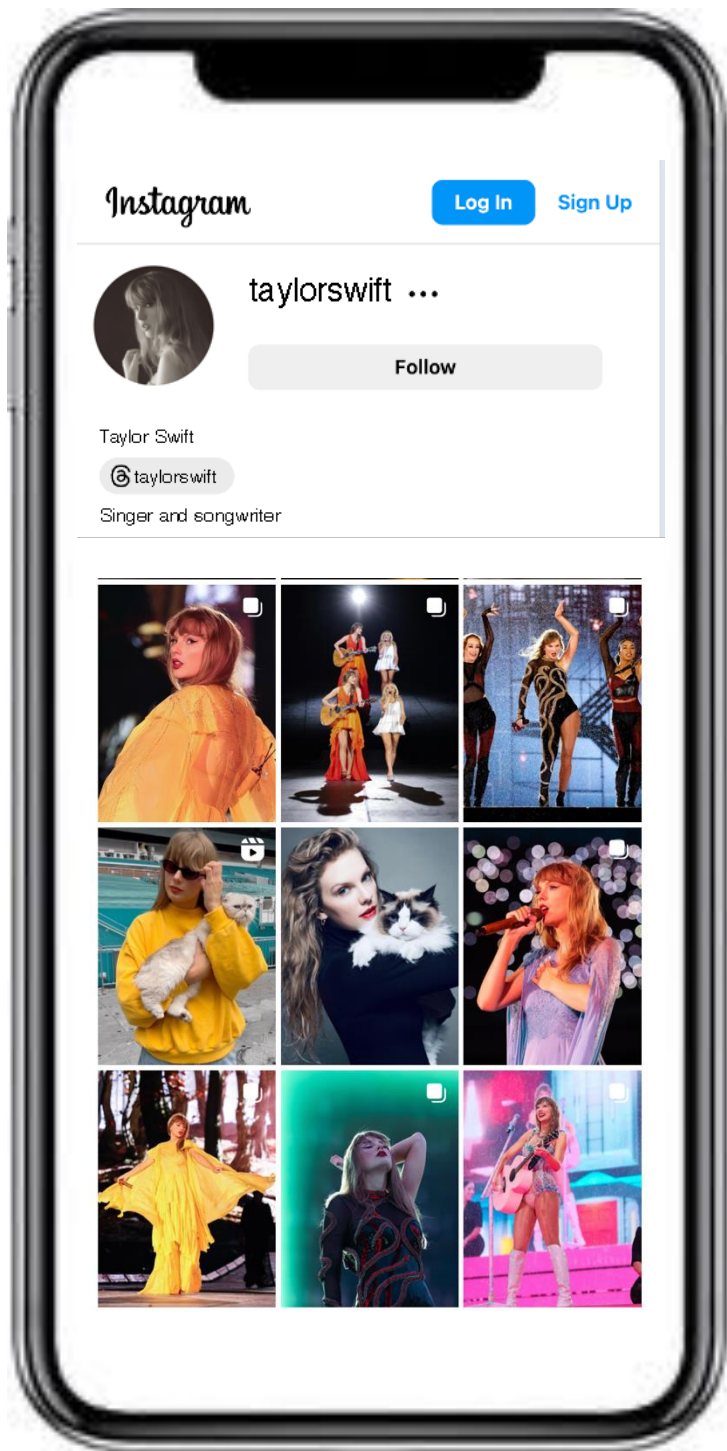


Set the bio for
taylorswift
to be "Singing"



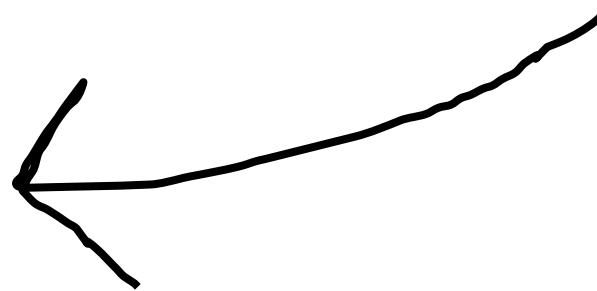
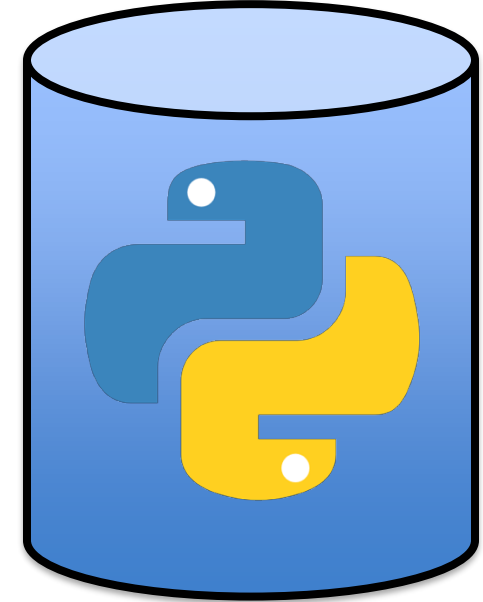
Instagram Server





Set the bio for
taylorswift
to be "Singing"

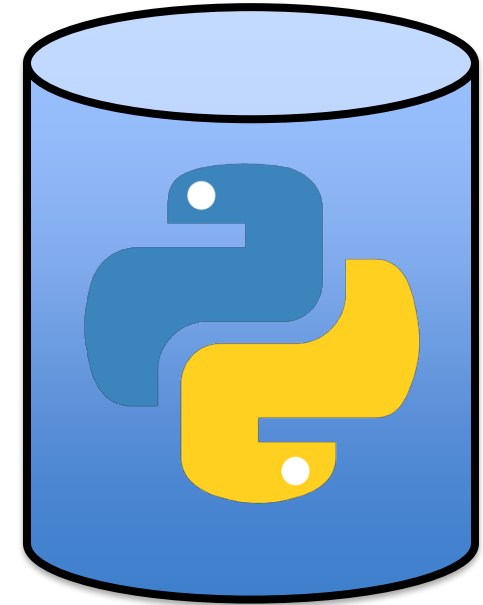
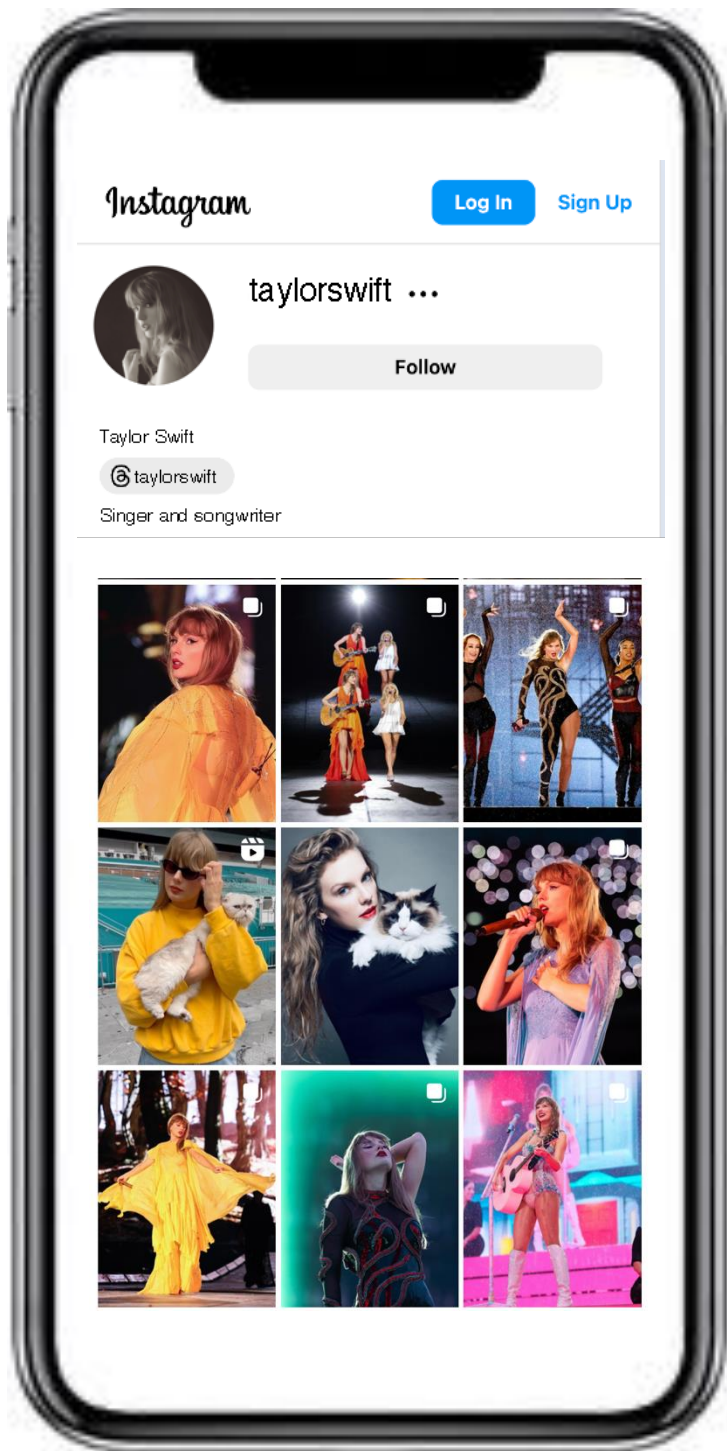
Instagram Server

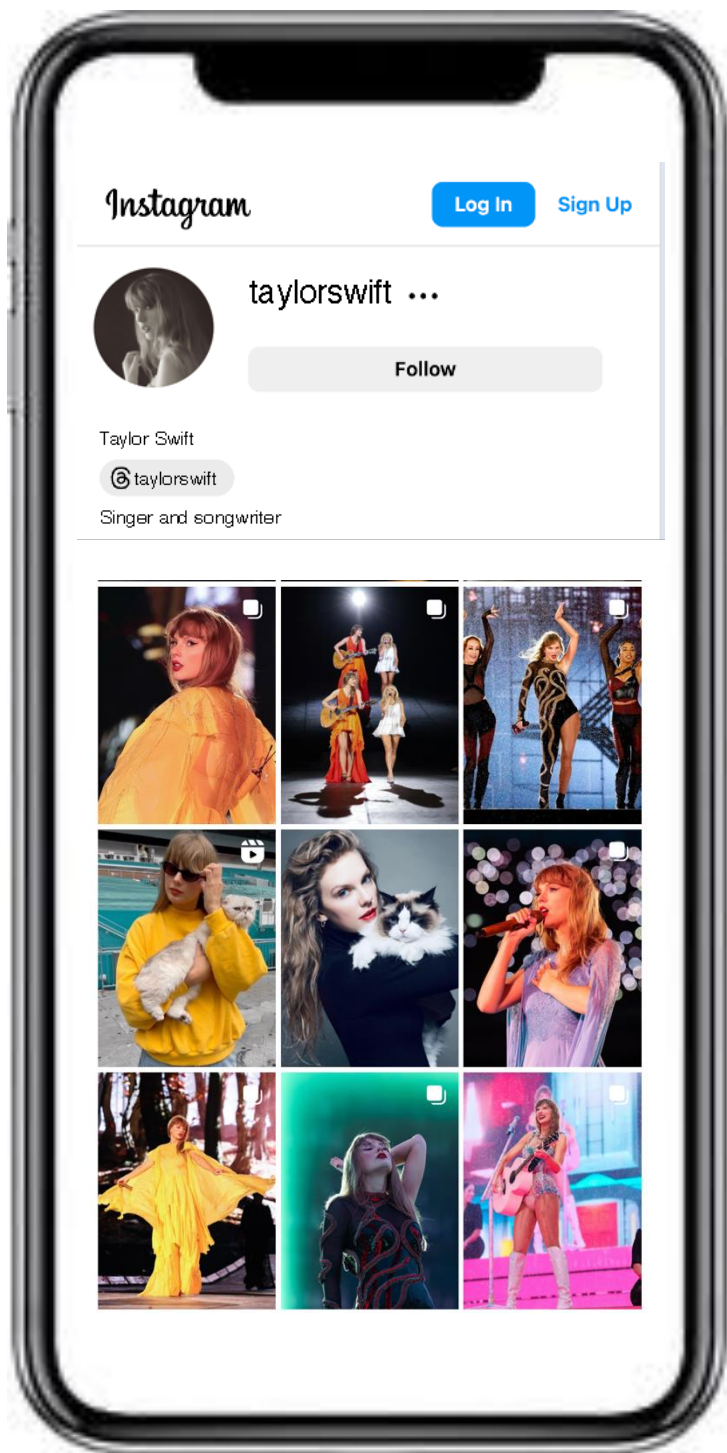


"success"



Instagram Server

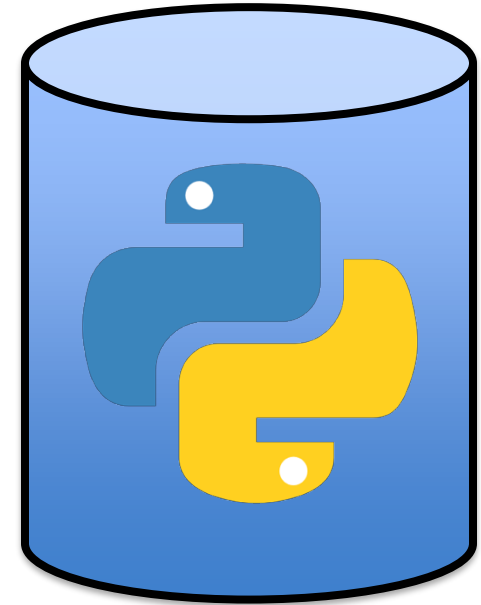


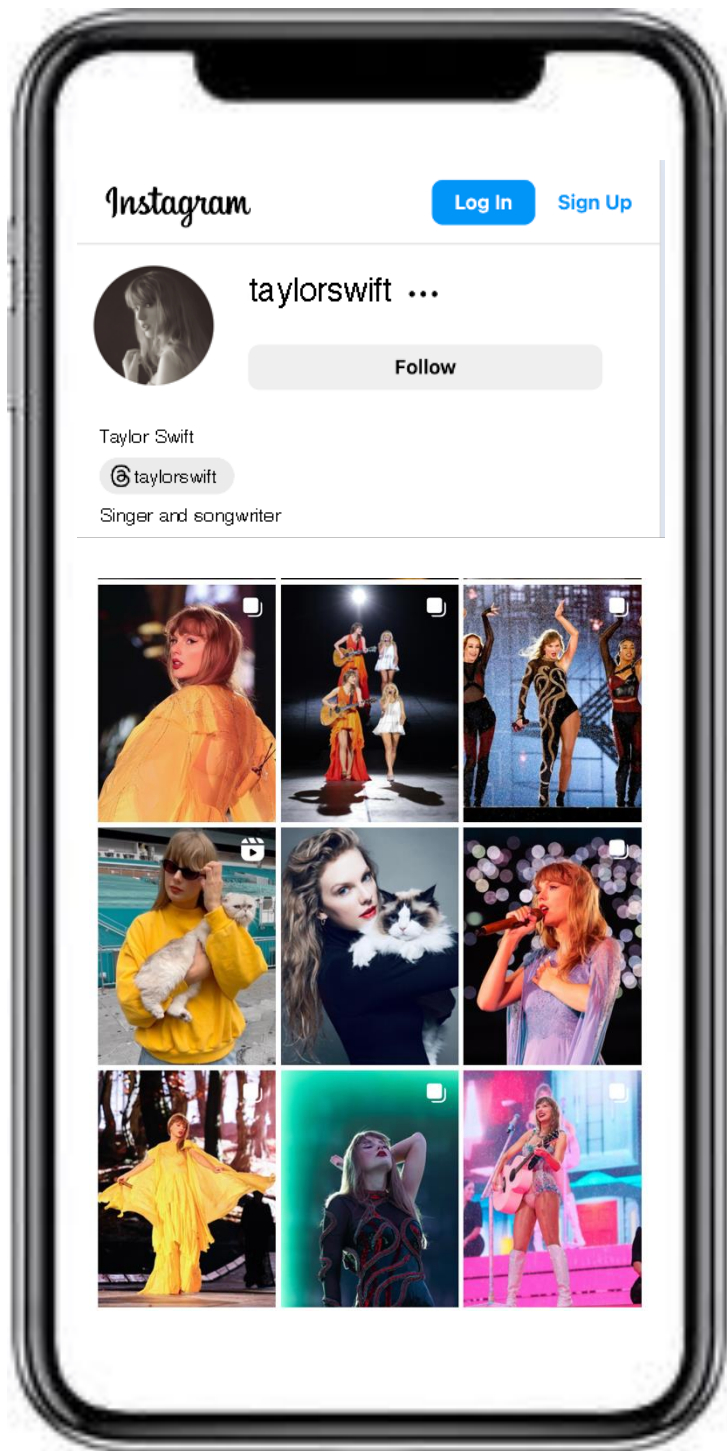


Send the
user info for
taylorswift



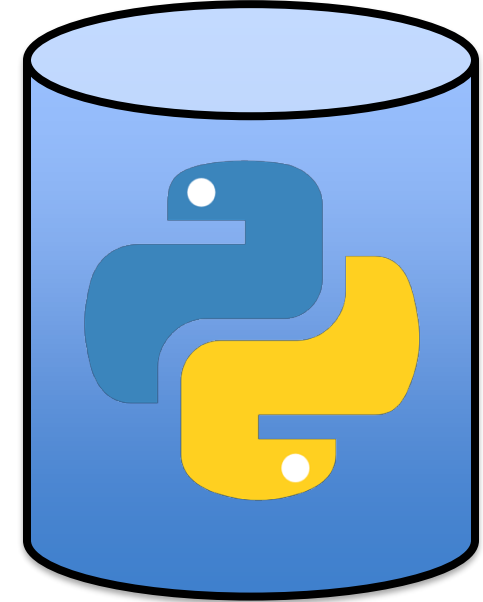
Instagram Server





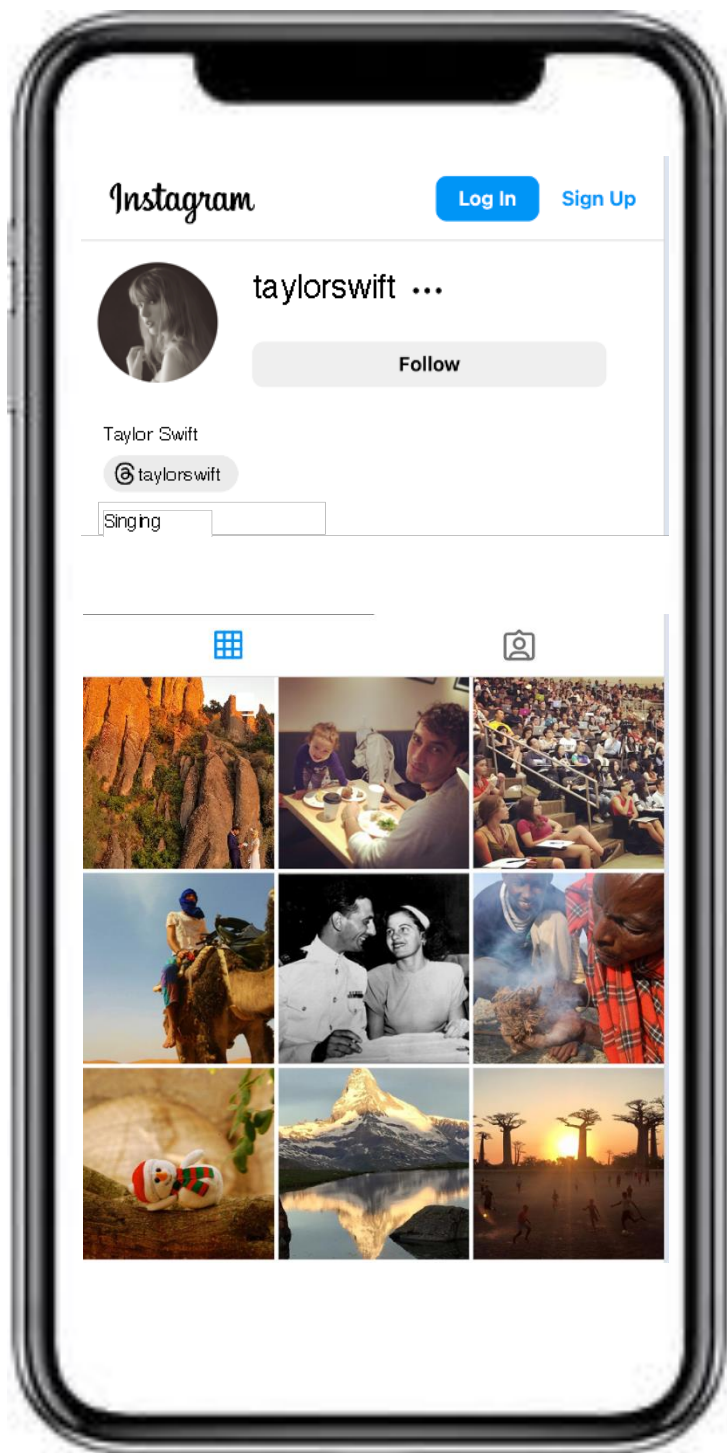
Send the
user info for
taylorswift

Instagram Server



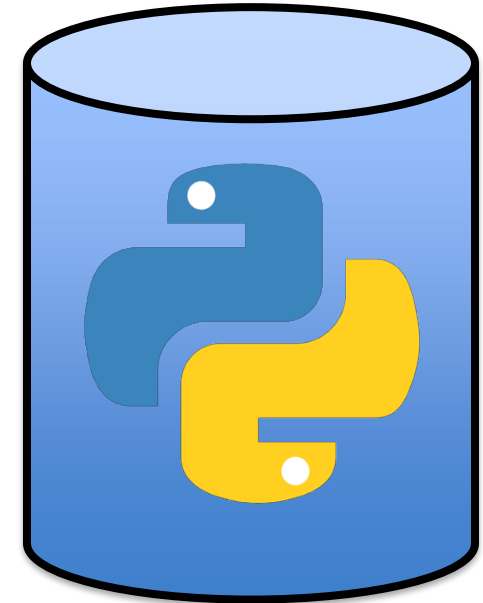
```
{  
  "name": "Taylor Swift",  
  "bio" : "Singing"  
}
```





Send the
user info for
taylorswift

Instagram Server

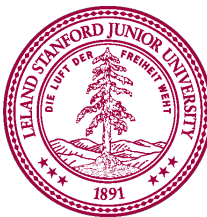


```
{  
  "name": "Taylor Swift",  
  "bio" : "Singing"  
}
```

Background: The Internet



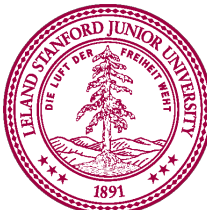
The internet is just many programs sending messages (as *Strings*)



Background: The Internet



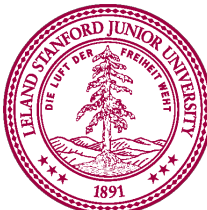
The internet is just many programs sending messages (as *Strings*)



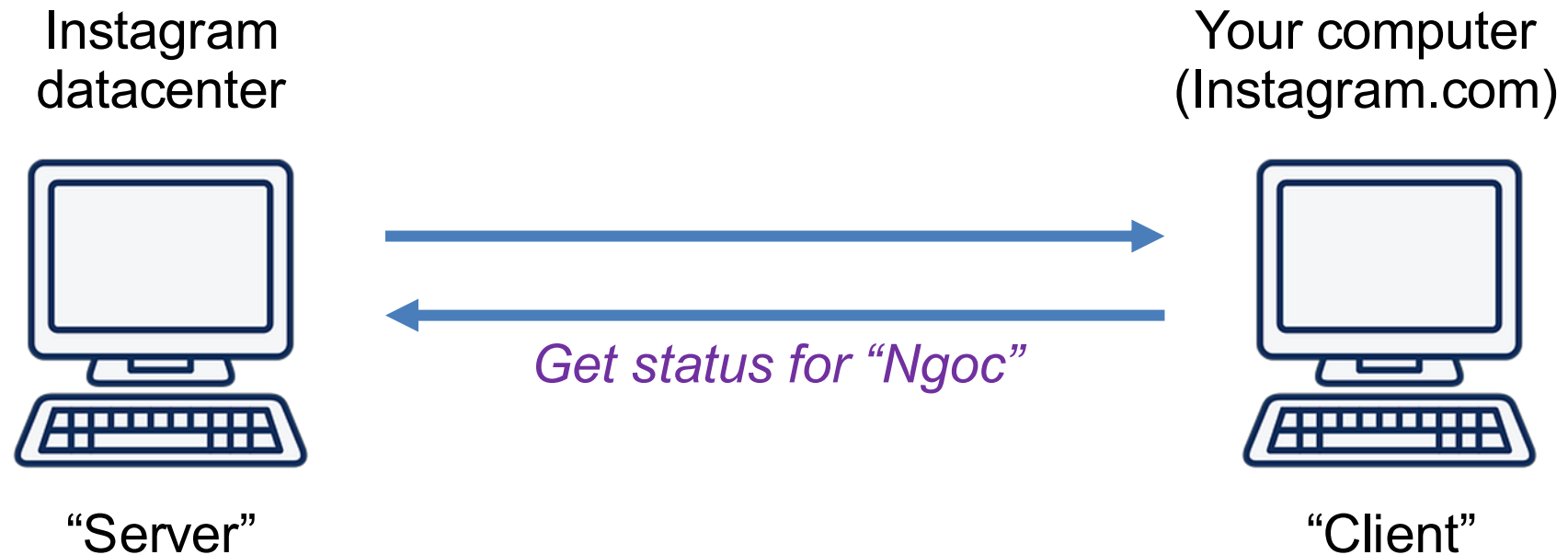
Background: The Internet



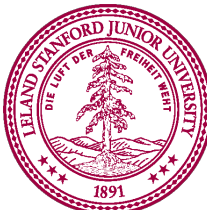
The internet is just many programs sending messages (as *Strings*)



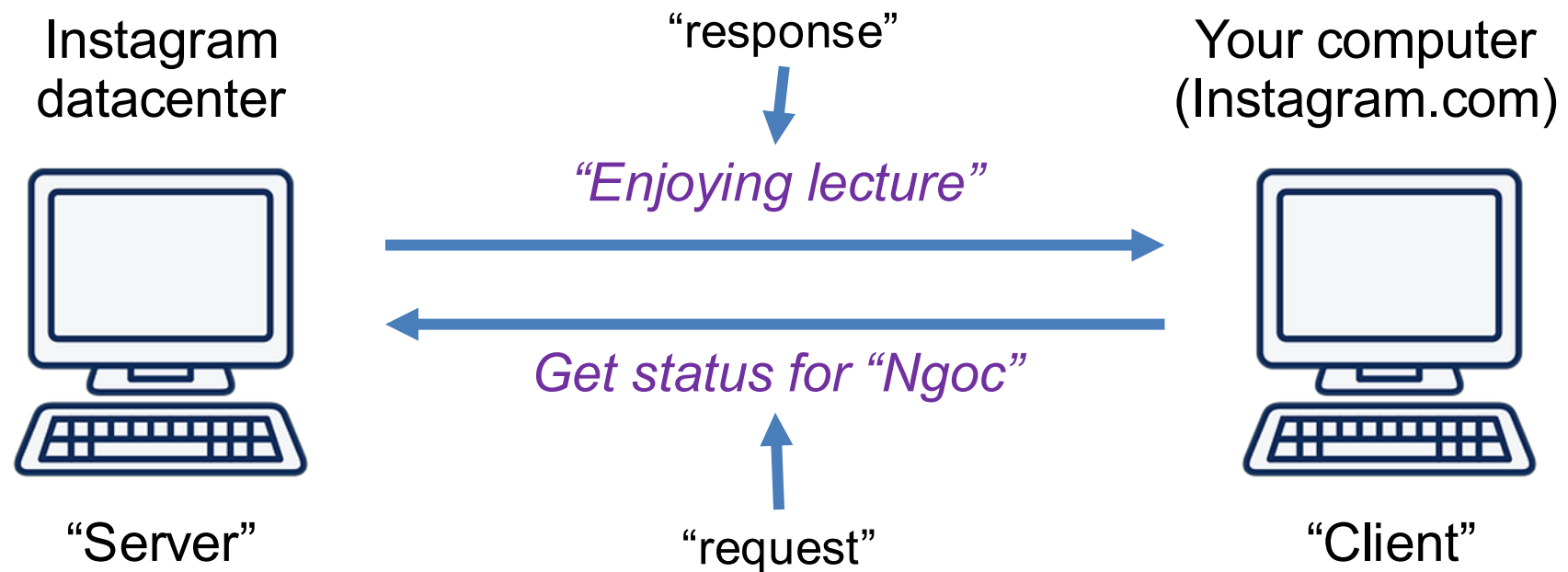
Background: The Internet



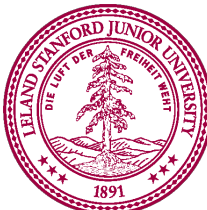
The internet is just many programs sending messages (as *Strings*)



Background: The Internet



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



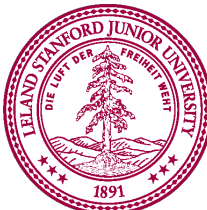
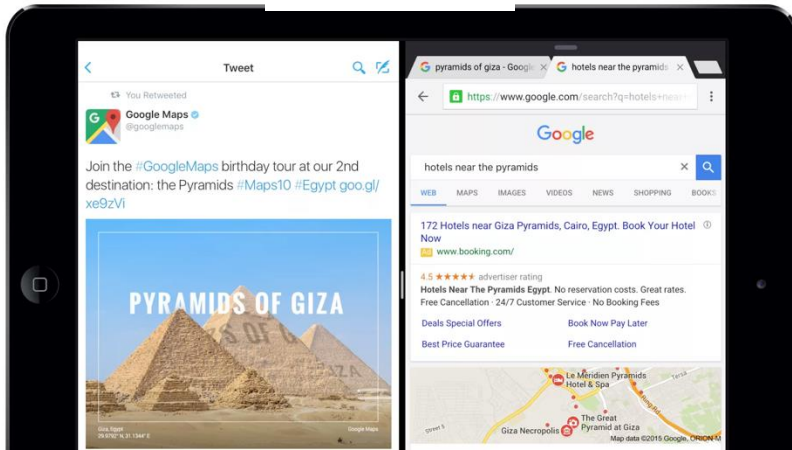


There are two types of
internet programs. Servers
and Clients



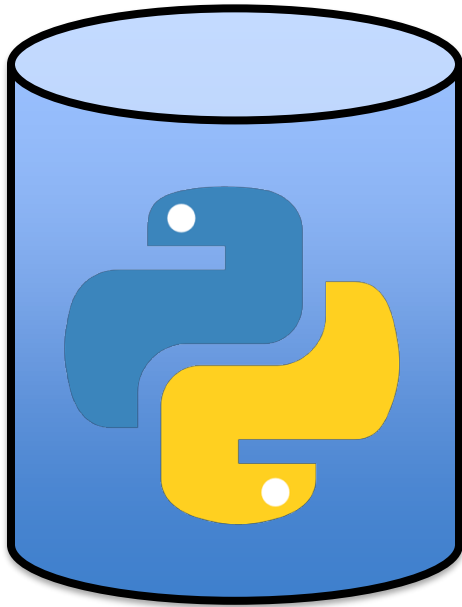
Internet 101

Computers on the internet



Servers are computers (running code)

Instagram Server



=

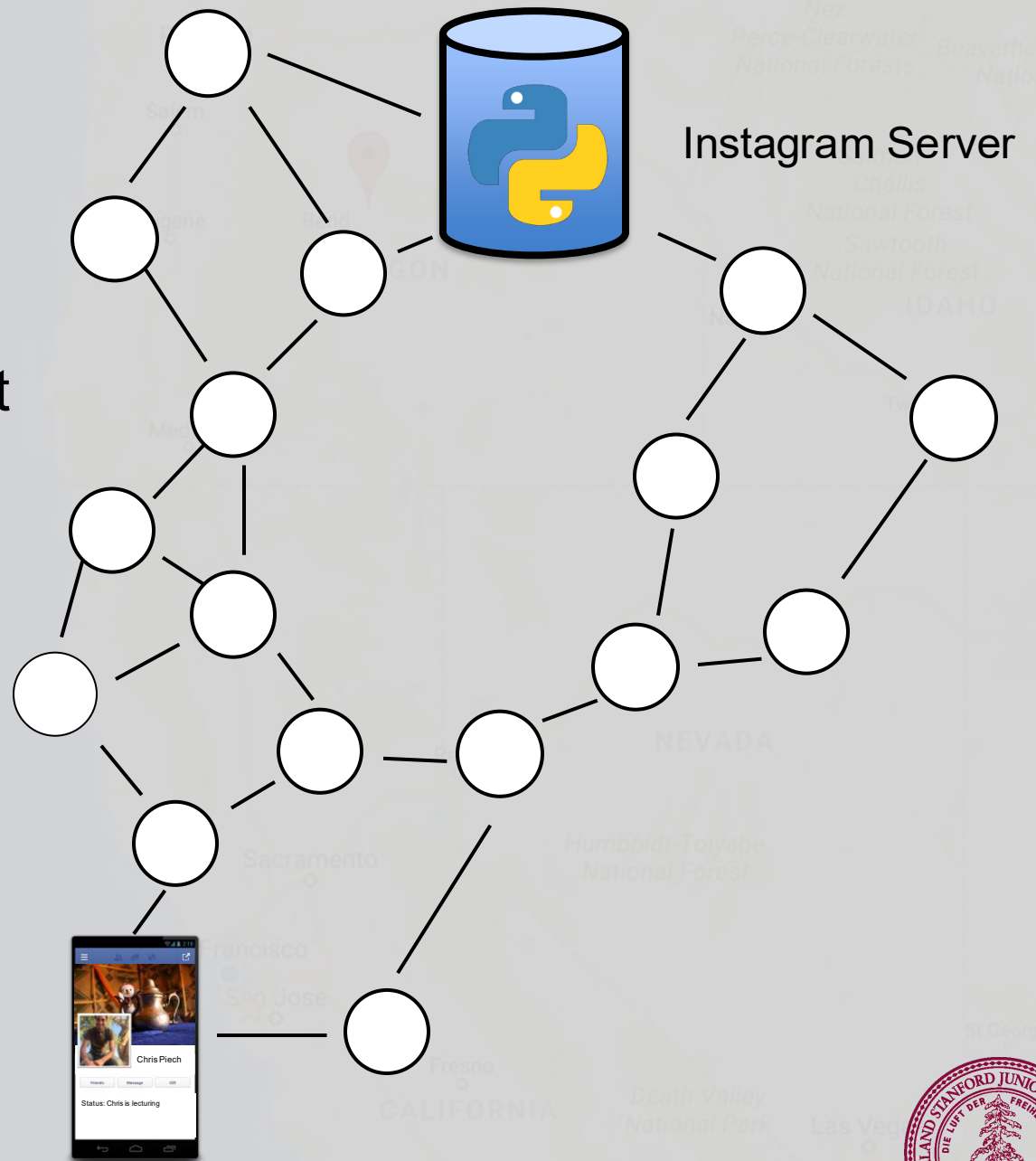


Instagram's closest
datacenter is here

I am here



The Internet



The Internet

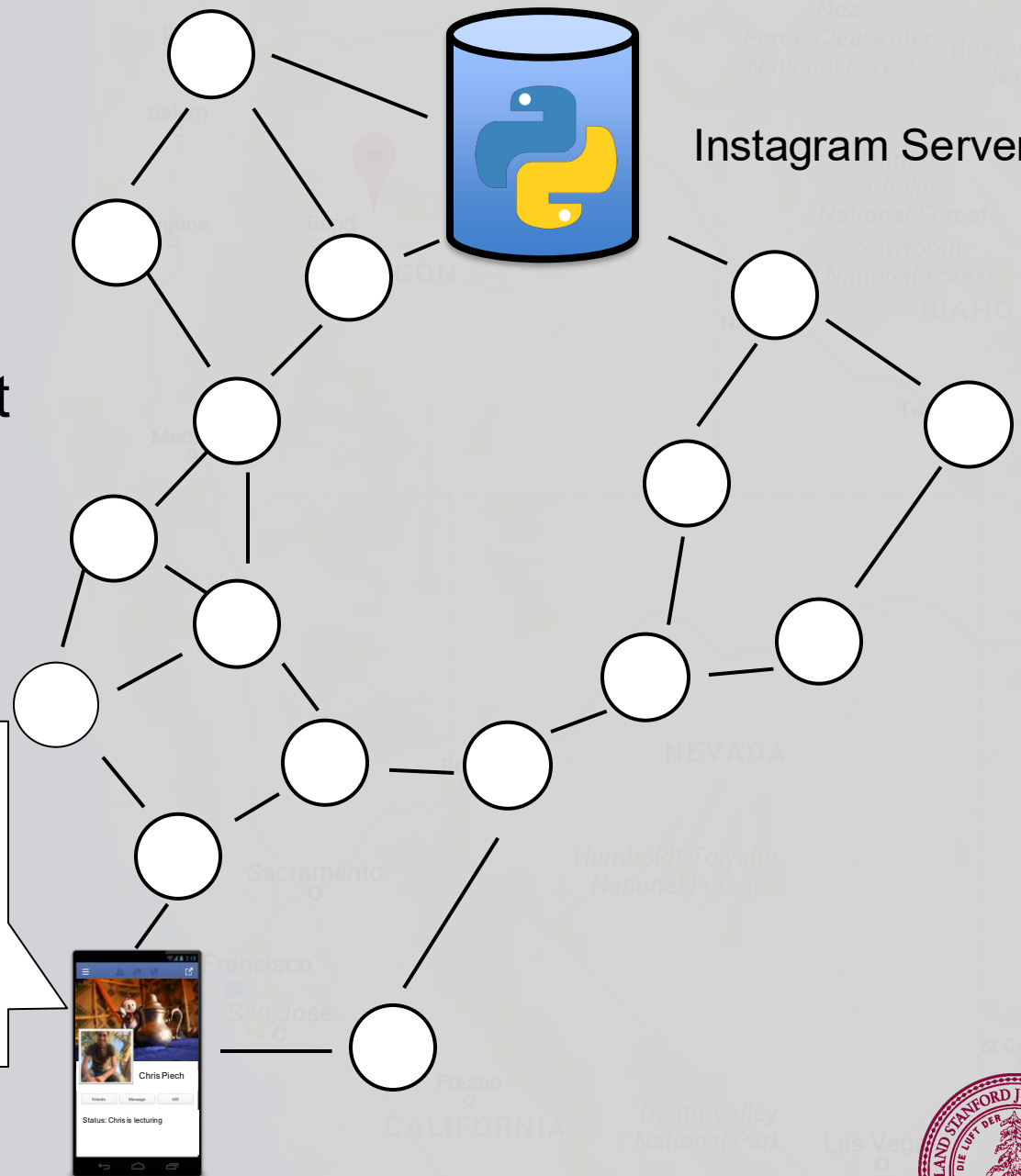
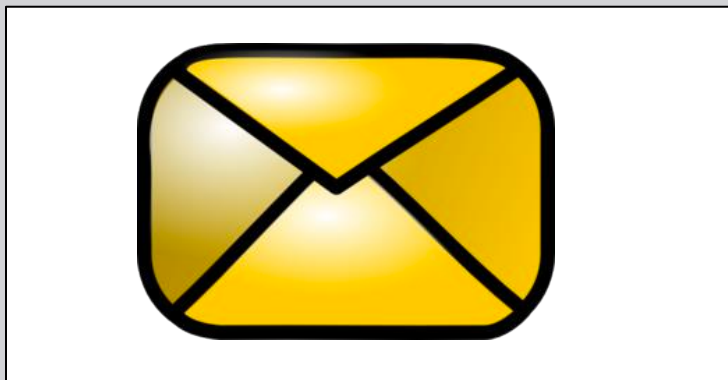
Instagram Server

Get bio for
taylorswift

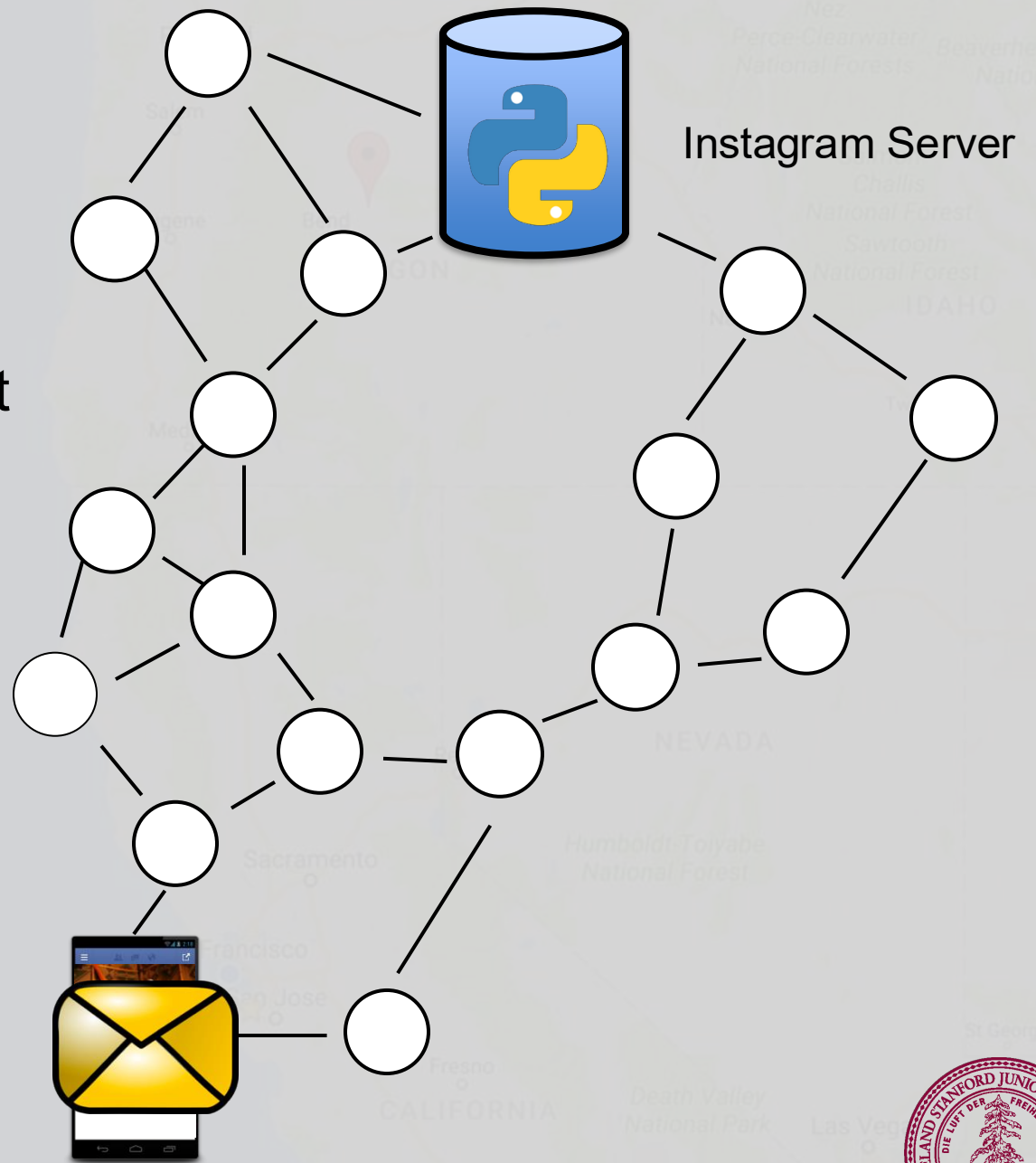


The Internet

Instagram Server



The Internet

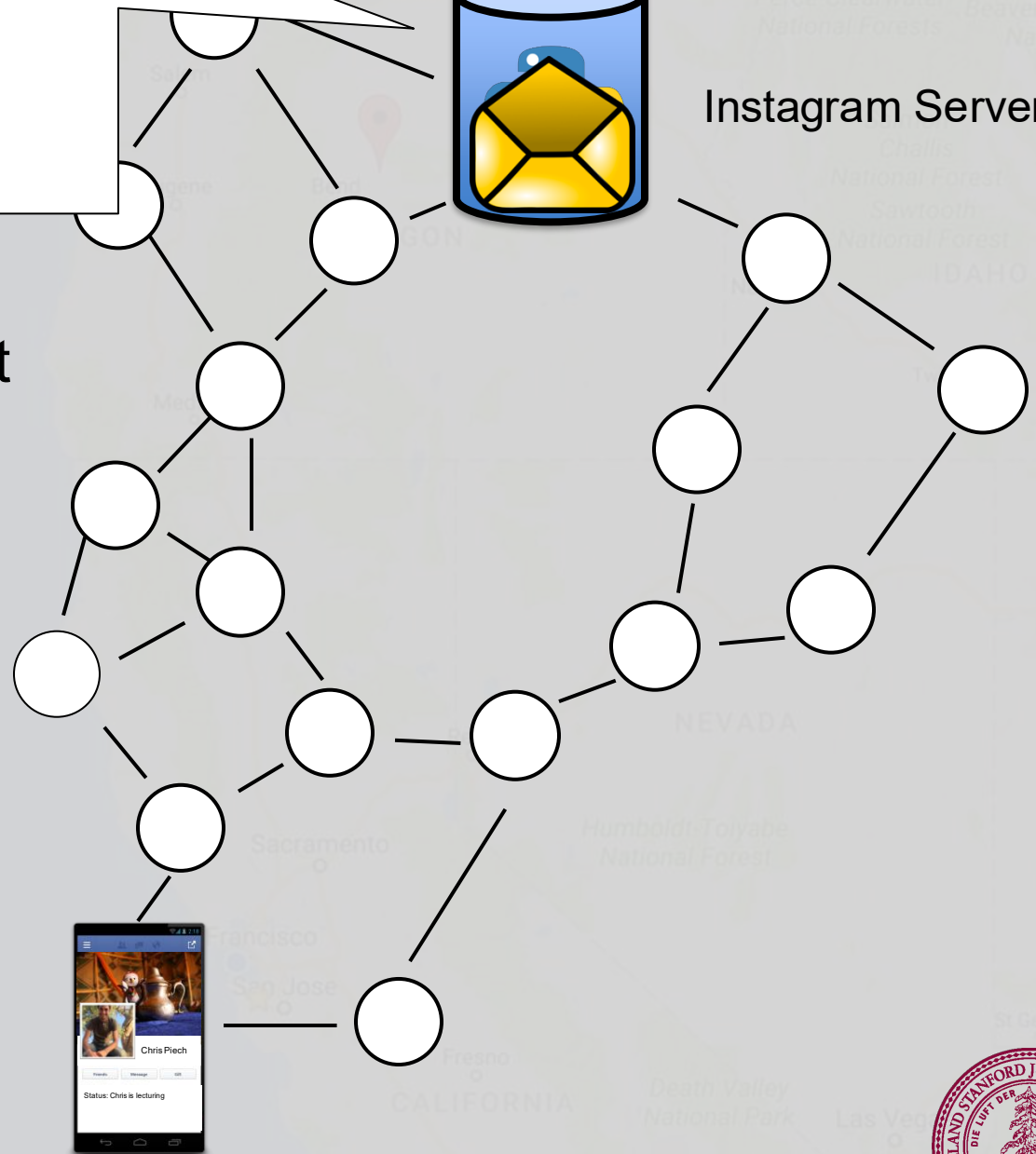


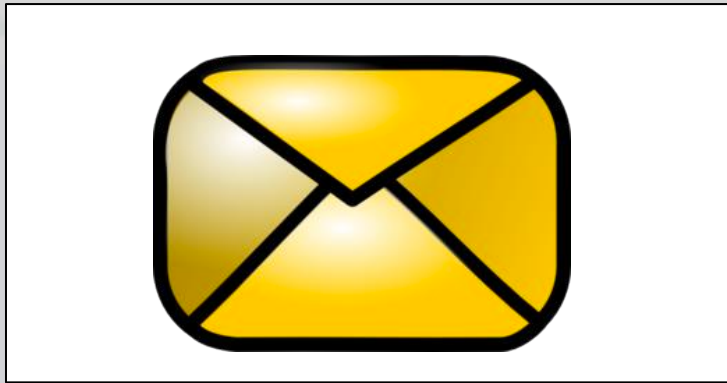
singing



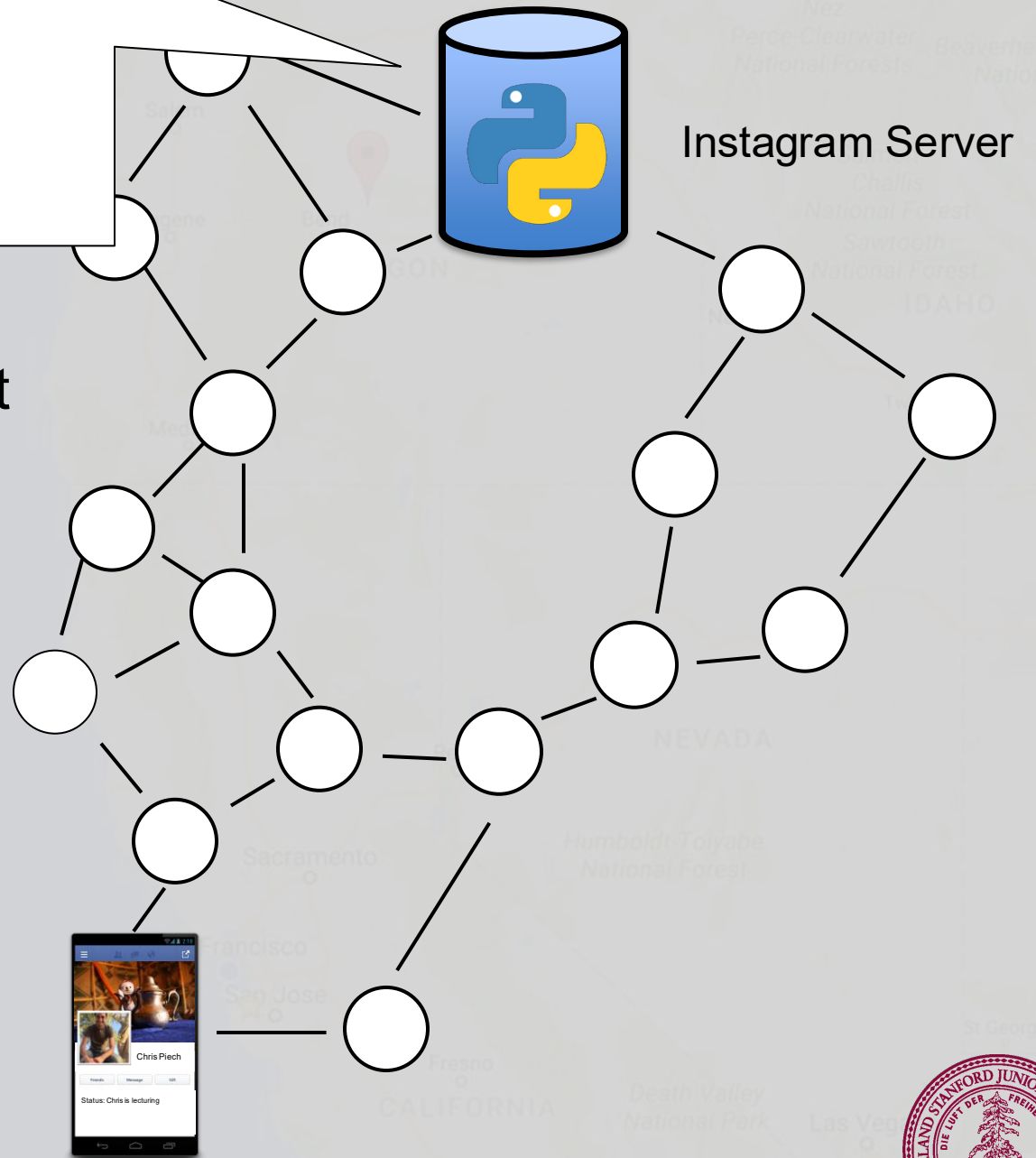
Instagram Server

The Internet

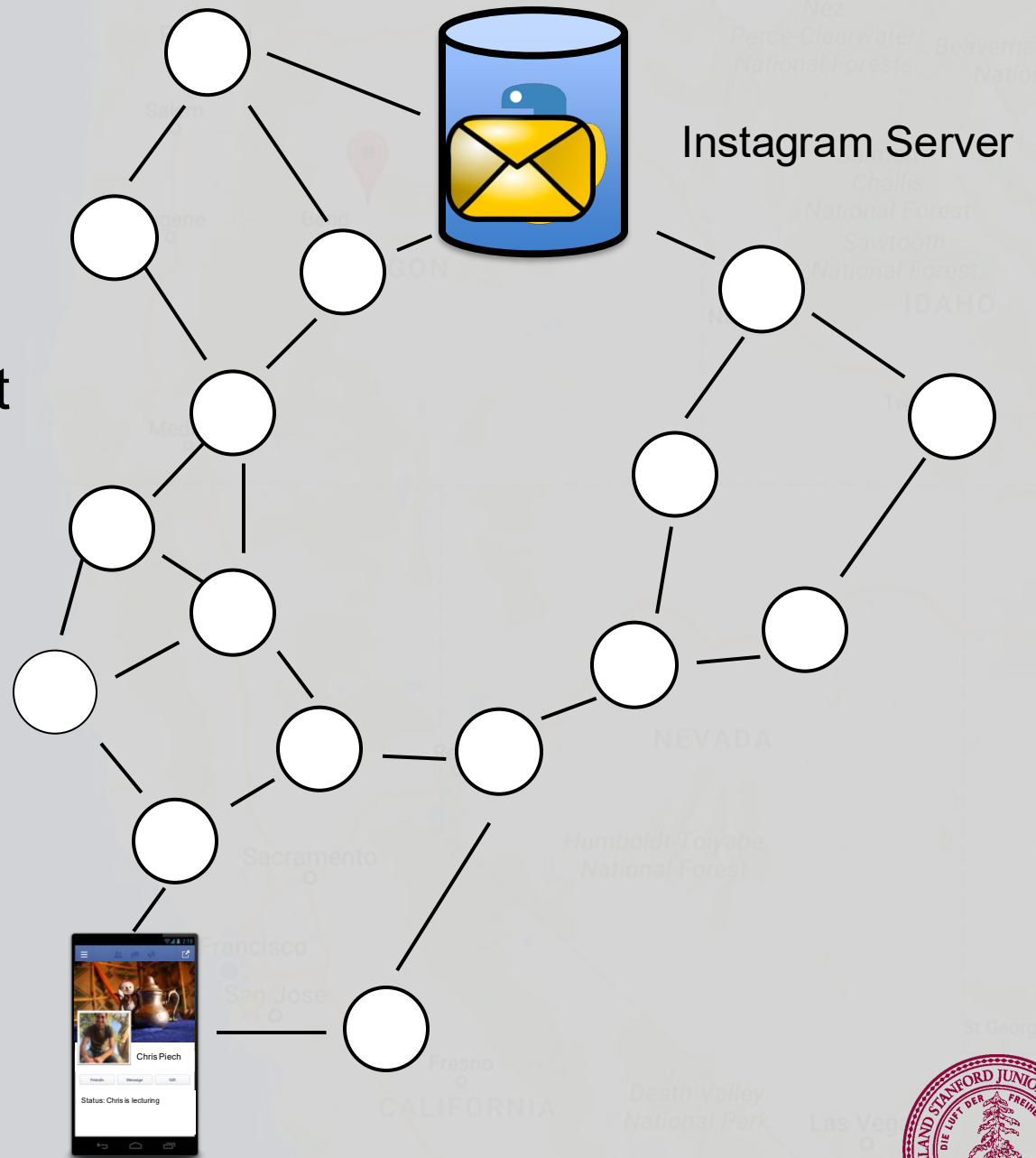




The Internet

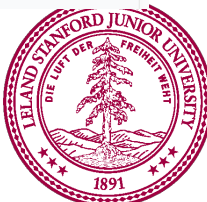
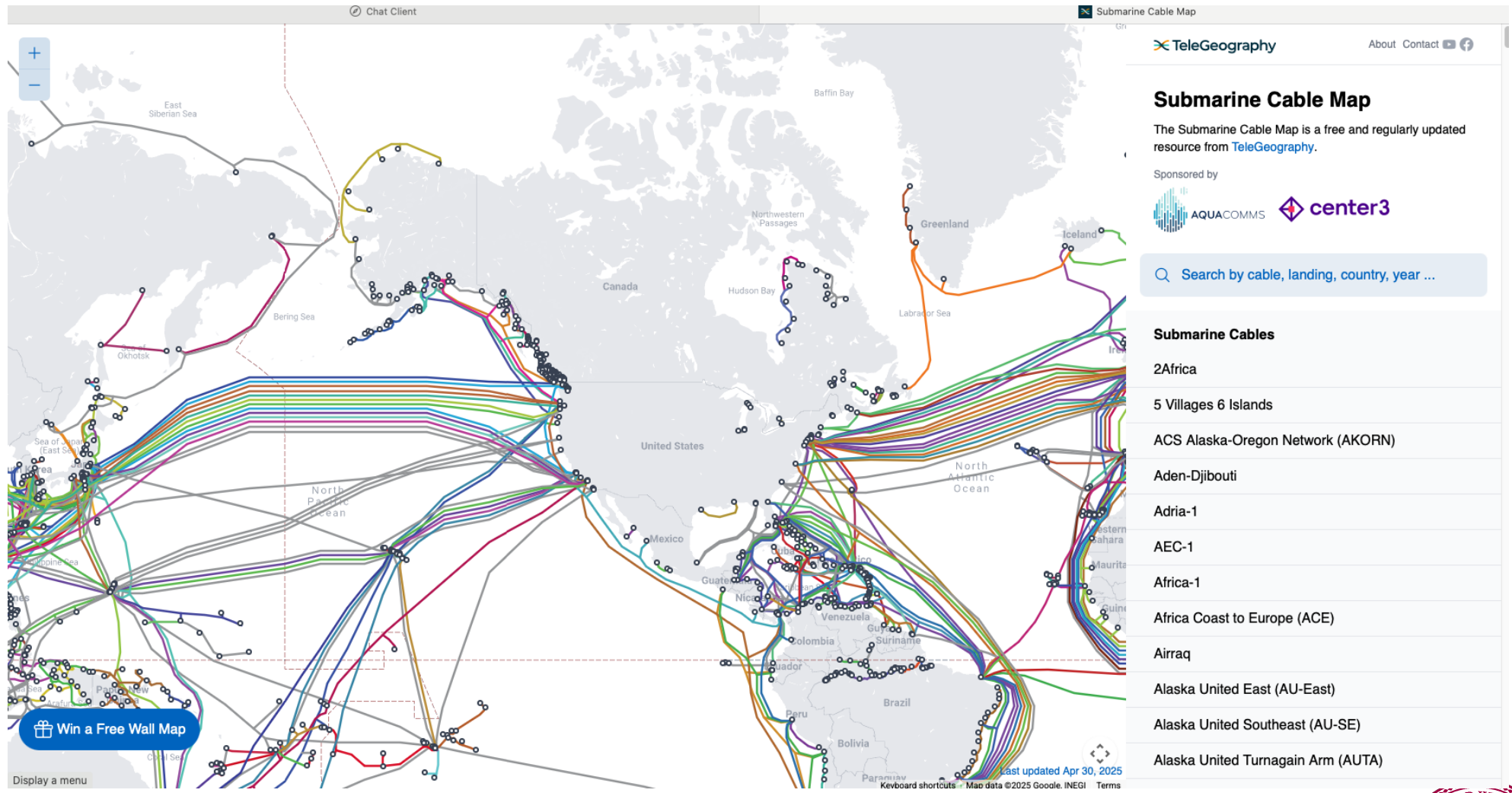


The Internet



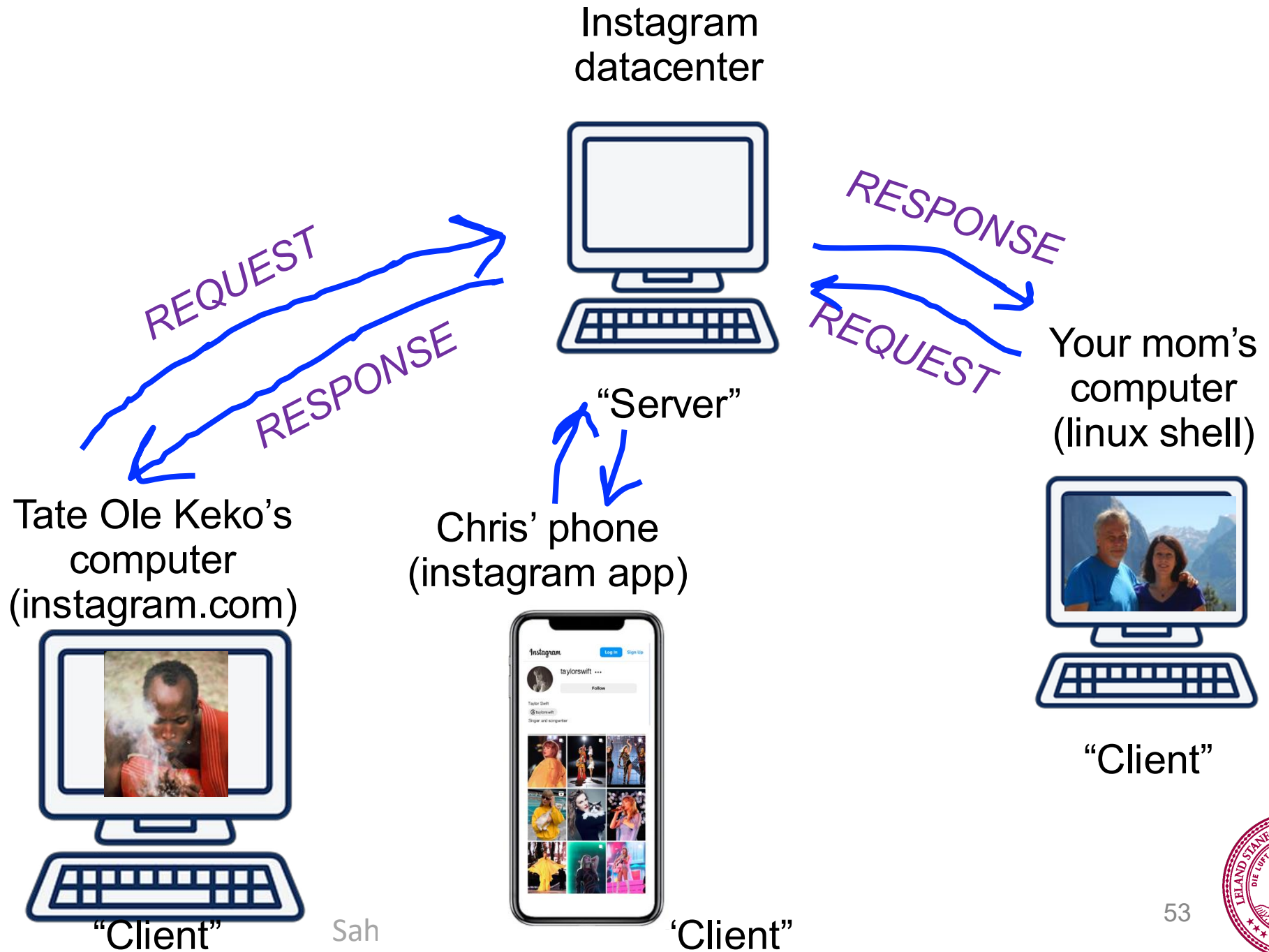
Aside: submarine cables:

<https://www.submarinecablemap.com>

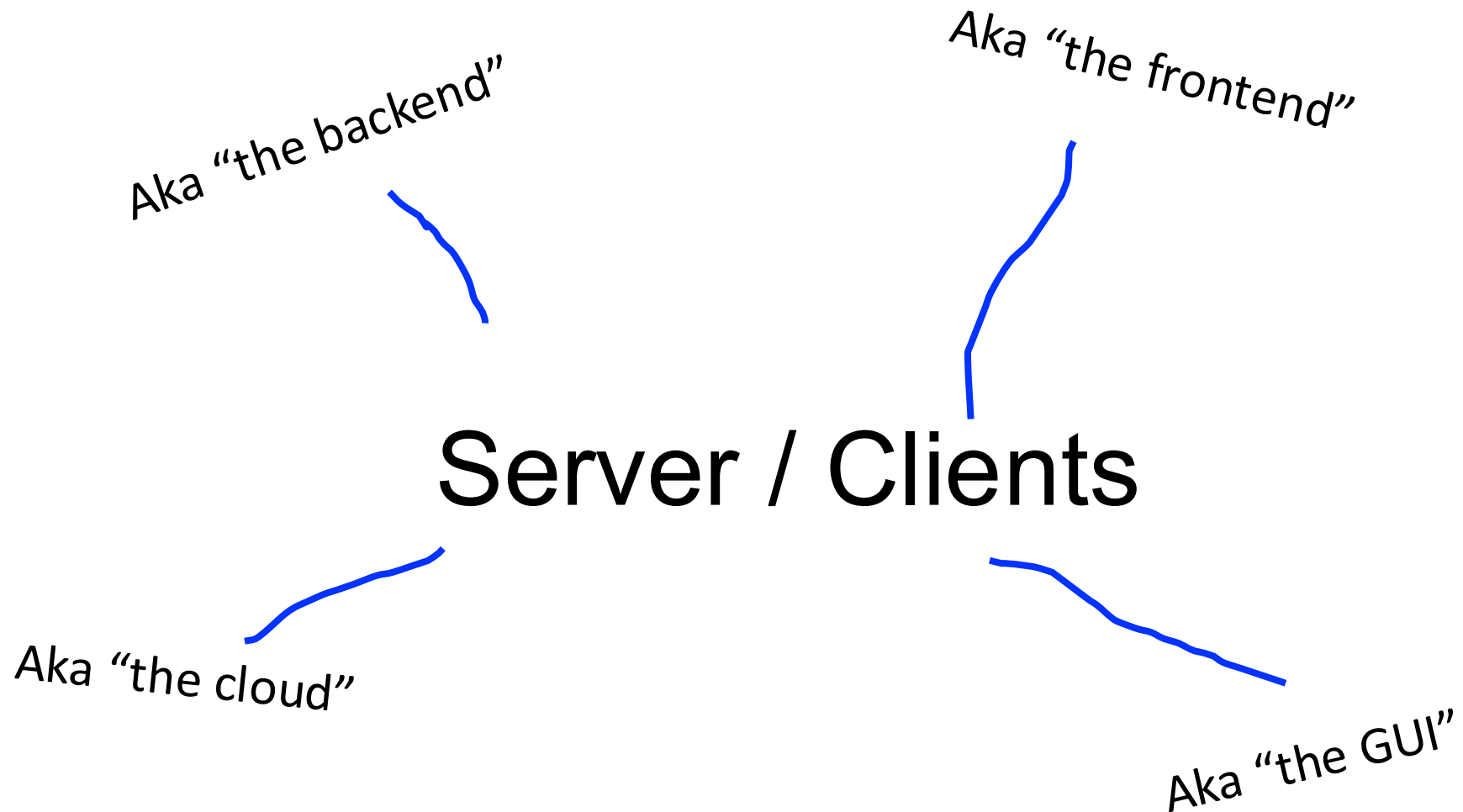


Many computers can connect
to the same server

The Internet



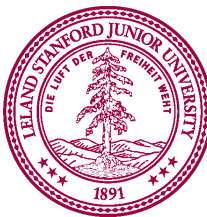
Most of the Internet



Today, the server



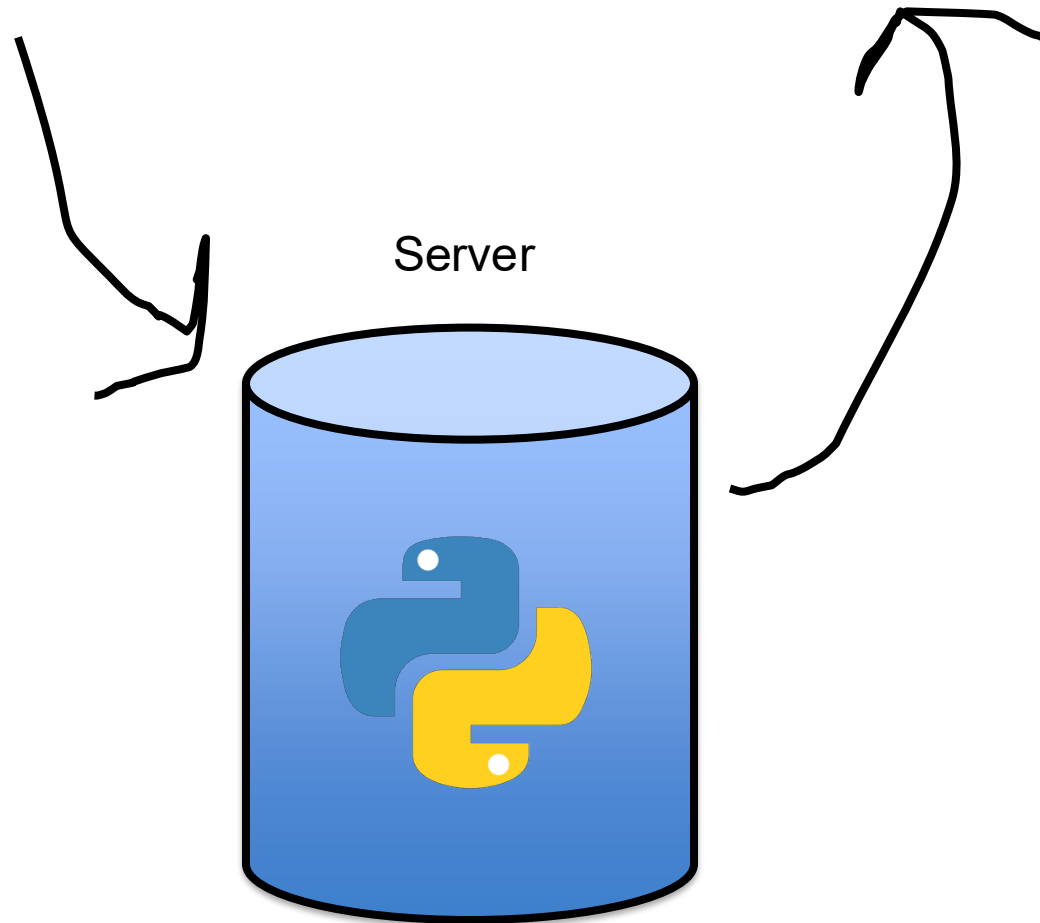
A server's main job is to
respond to requests



A Server's Simple Purpose

Request
From a client

Response
To the client



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
```



Servers on one slide

1

handle server requests (must be in a class)

```
class MyServer:
```

```
    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)
```

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(self, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

 handler = HitServer()

start the server!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(**self**, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

 handler = HitServer()

start the server!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class **MyServer**:

def **handle_request**(self, request):

return a string response!

2

turn on the server

def **main**():

make an instance of your server class

handler = HitServer()

start the server!

run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def *handle_request*(self, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

handler = HitServer()

start the server!

run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(self, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

 handler = HitServer()

start the server!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(self, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

handler = HitServer()

start the server!

run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(self, request):

return "hello world"

2

turn on the server

def main():

make an instance of your server class

 handler = HitServer()

start the server!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

def handle_request(self, request):

return a string response!

2

turn on the server

def main():

make an instance of your server class

 handler = HitServer()

start the server!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

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!

 run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

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!

run_server(handler, 8000)

3

enjoy



Servers on one slide

1

handle server requests (must be in a class)

class MyServer:

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!

run_server(**handler**, **8000**)

3

enjoy



What is a Port?



Servers on one slide

1

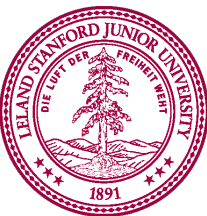
```
# handle server requests (must be in a class)
class MyServer:
    def handle_request(self, request):
        # return a string response!
```

2

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

3

```
# enjoy
```



Servers on one slide

1

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

2

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

3

```
# enjoy
```



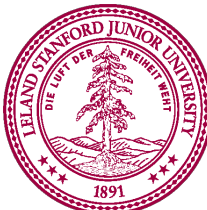
What is a Request?



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

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

```
// methods that the server calls on requests  
request.get_command()  
request.get_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 an 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 request  
    return 'command=\'\' + self.command + '\\' params=\'\' + str(self.params)
```

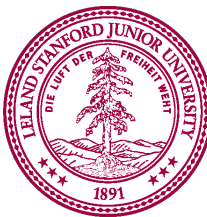
First Server Example!

```
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/')
```

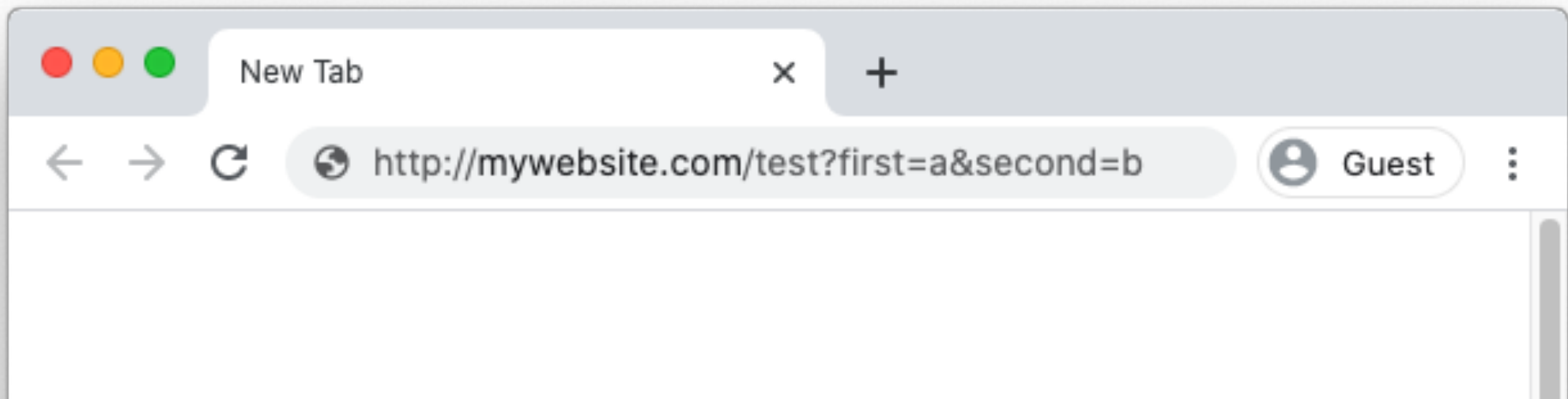
Who makes requests?

Other programs can send requests!

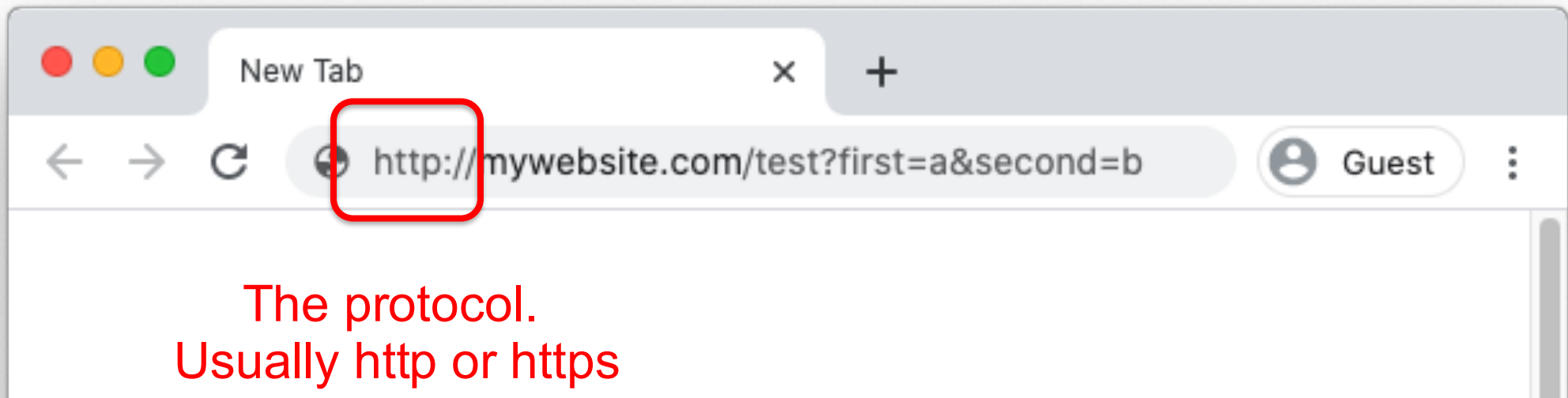
```
response = requests.get('https://xkcd.com/353/')
```

Web browsers can send requests!

Anatomy of a Browser Request



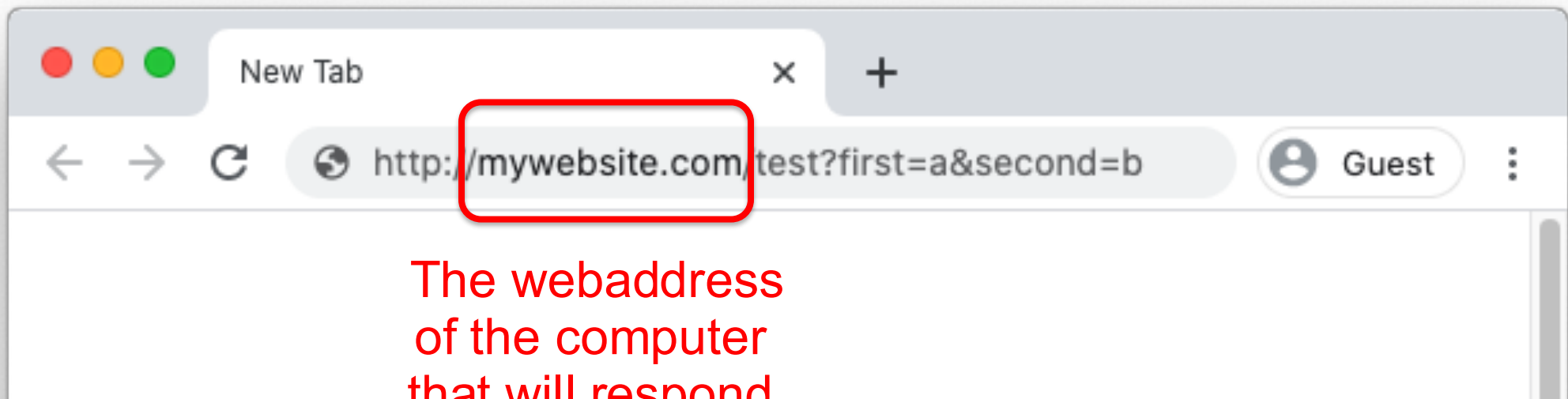
Anatomy of a Browser Request



The protocol.
Usually http or https



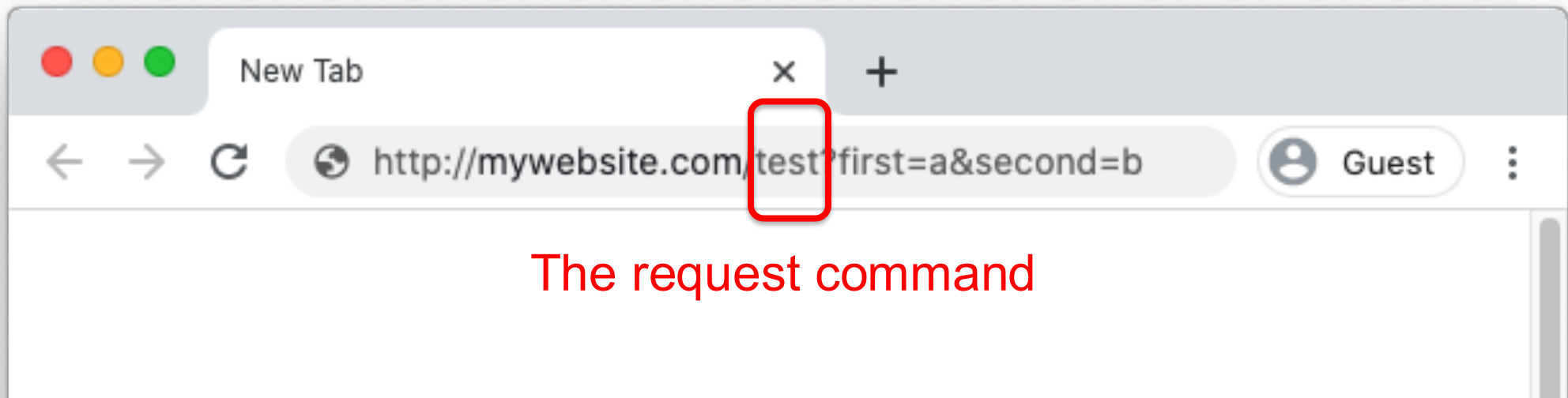
Anatomy of a Browser Request



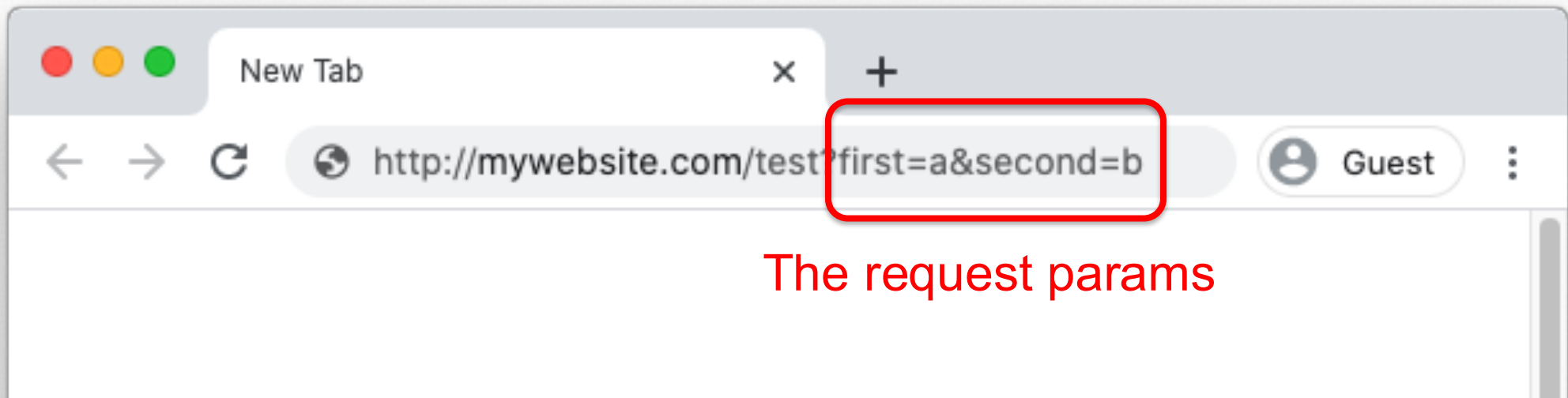
The webaddress
of the computer
that will respond
to the request



Anatomy of a Browser Request



Anatomy of a Browser Request



The request params



First Server Example!

```
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)
```



Housekeeping

- Assignment 5 (last assignment!) out now and due back next Wednesday at 11:59PM
- Final exam logistics + materials on course website
- Final exam: **Saturday August 16th from 7-10PM in Hewlett 200**
- Last class: Monday August 11th
- Neel's office hours likely shifted earlier than Friday



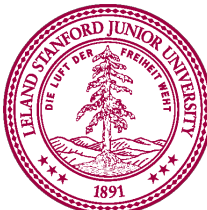
Recall Requests



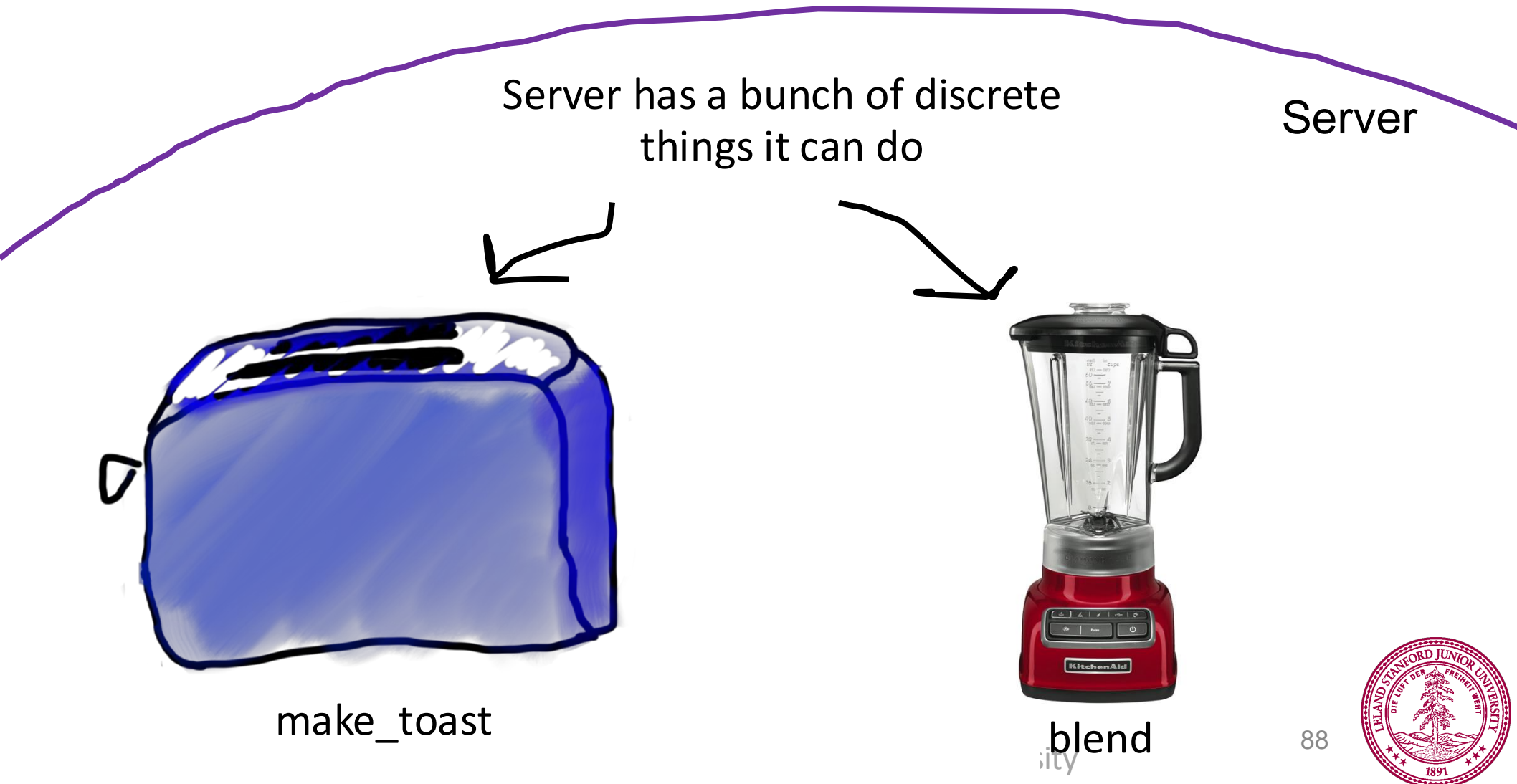
```
/* Request has a command */  
command (string)
```

```
/* Request has parameters */  
params (dict)
```

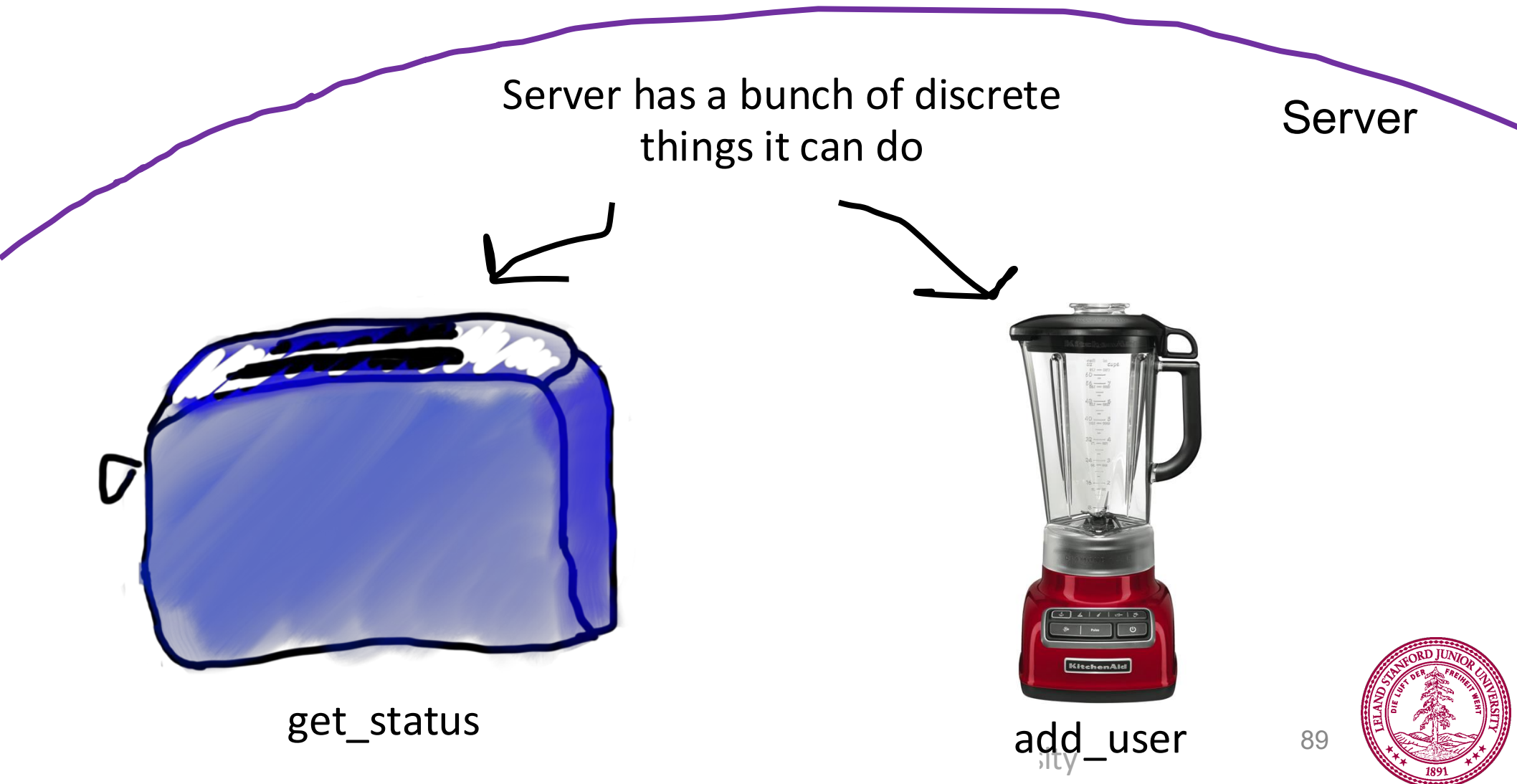
```
// methods that the server calls on requests  
request.get_command()  
request.get_params()
```



Requests are like Remote Method Calls

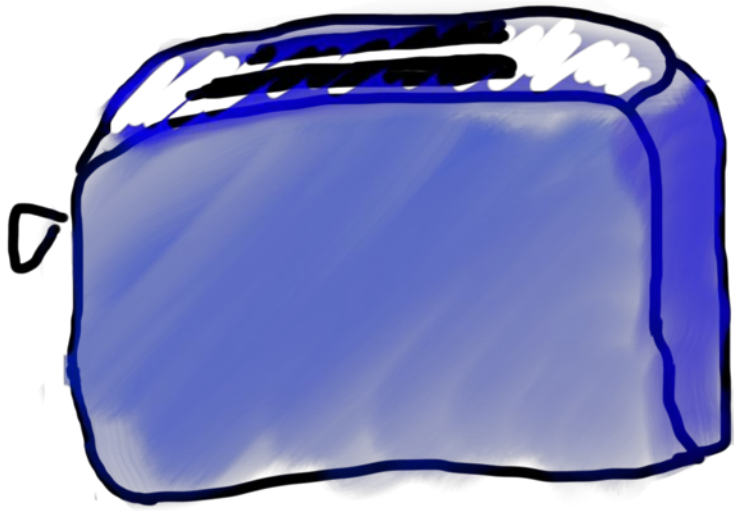


Requests are like Remote Method Calls



Requests are like Remote Method Calls

Server

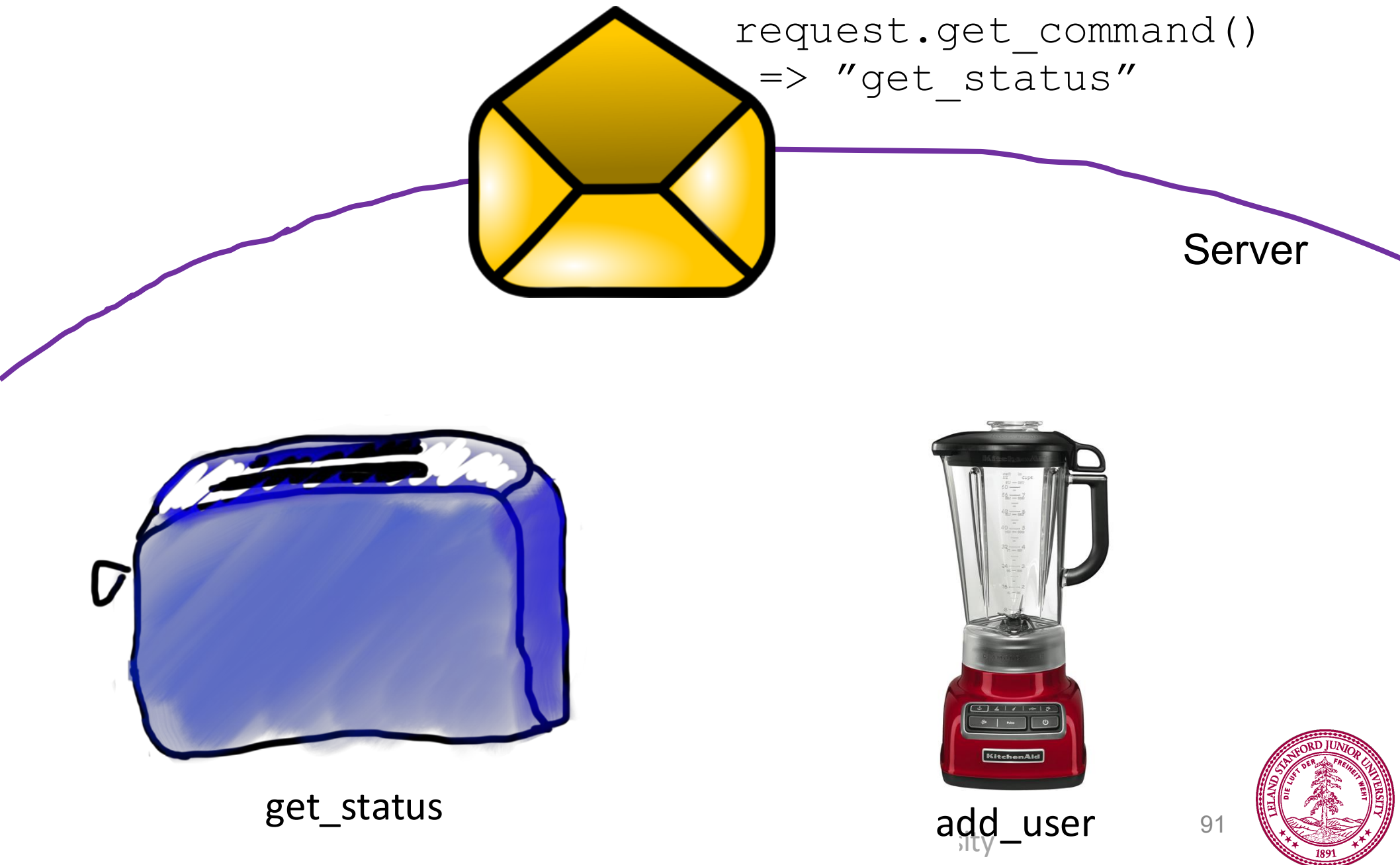


get_status



add_user

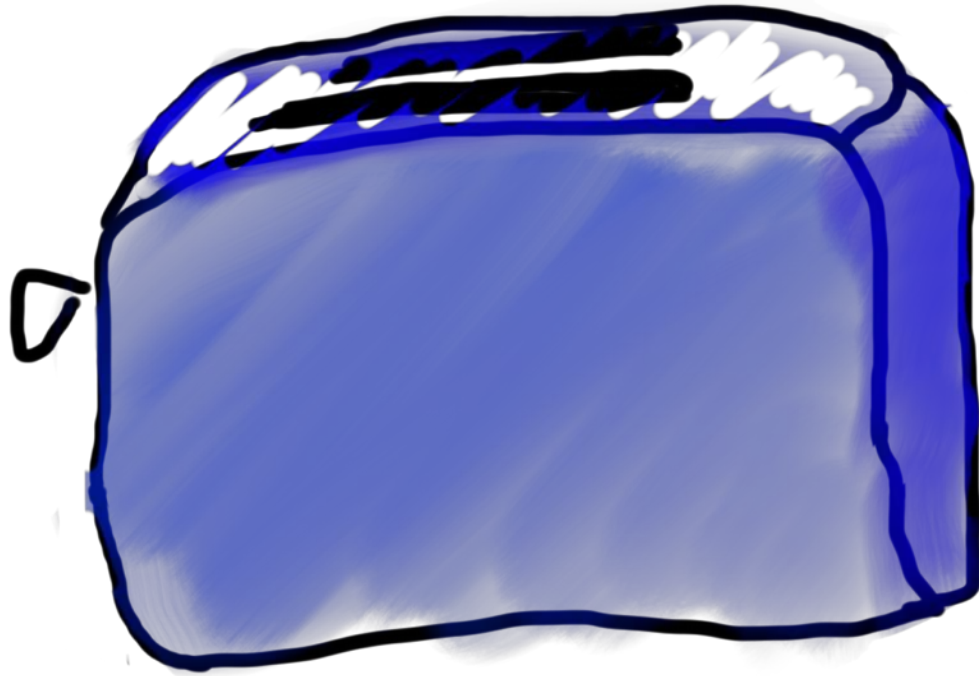
Requests are like Remote Method Calls



Requests are like Remote Method Calls



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

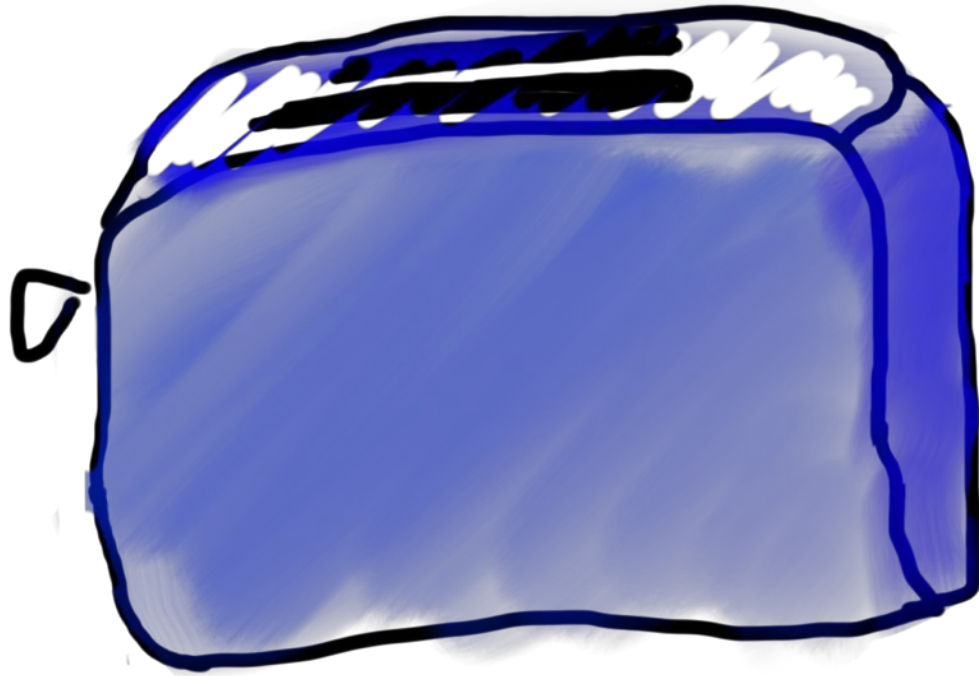


get_status

Requests are like Remote Method Calls



I was given a parameter!

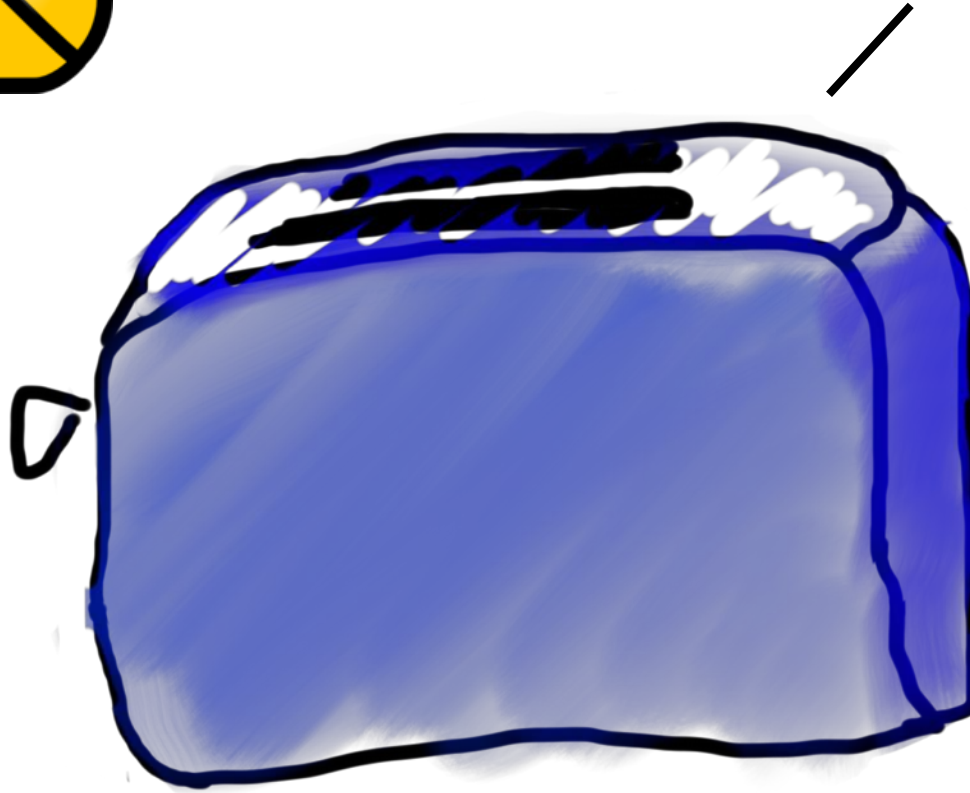


get_status

Requests are like Remote Method Calls

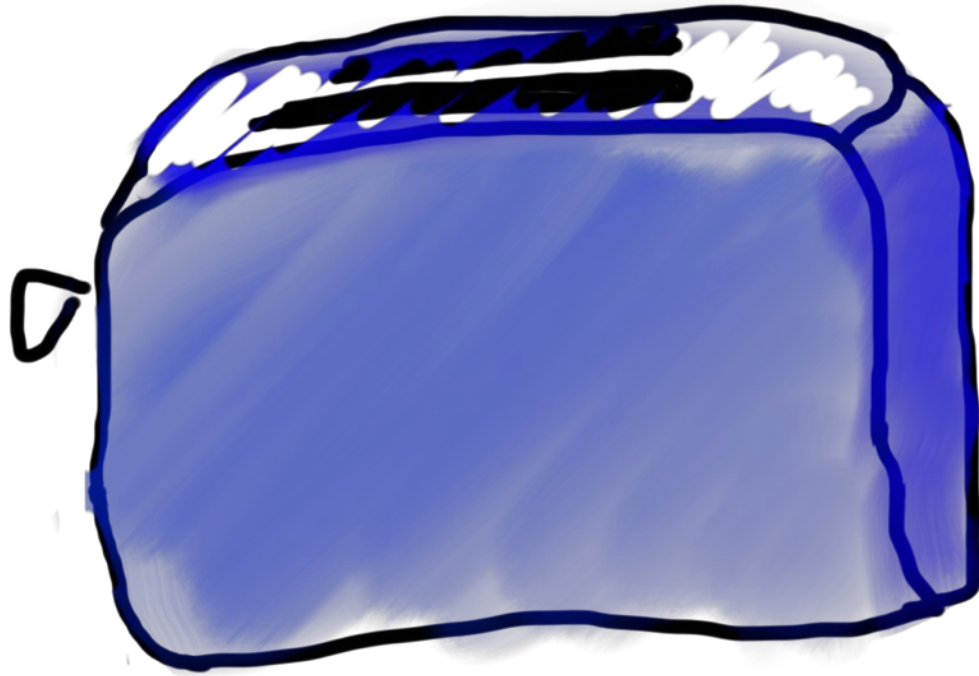


`request.params["userName"]`



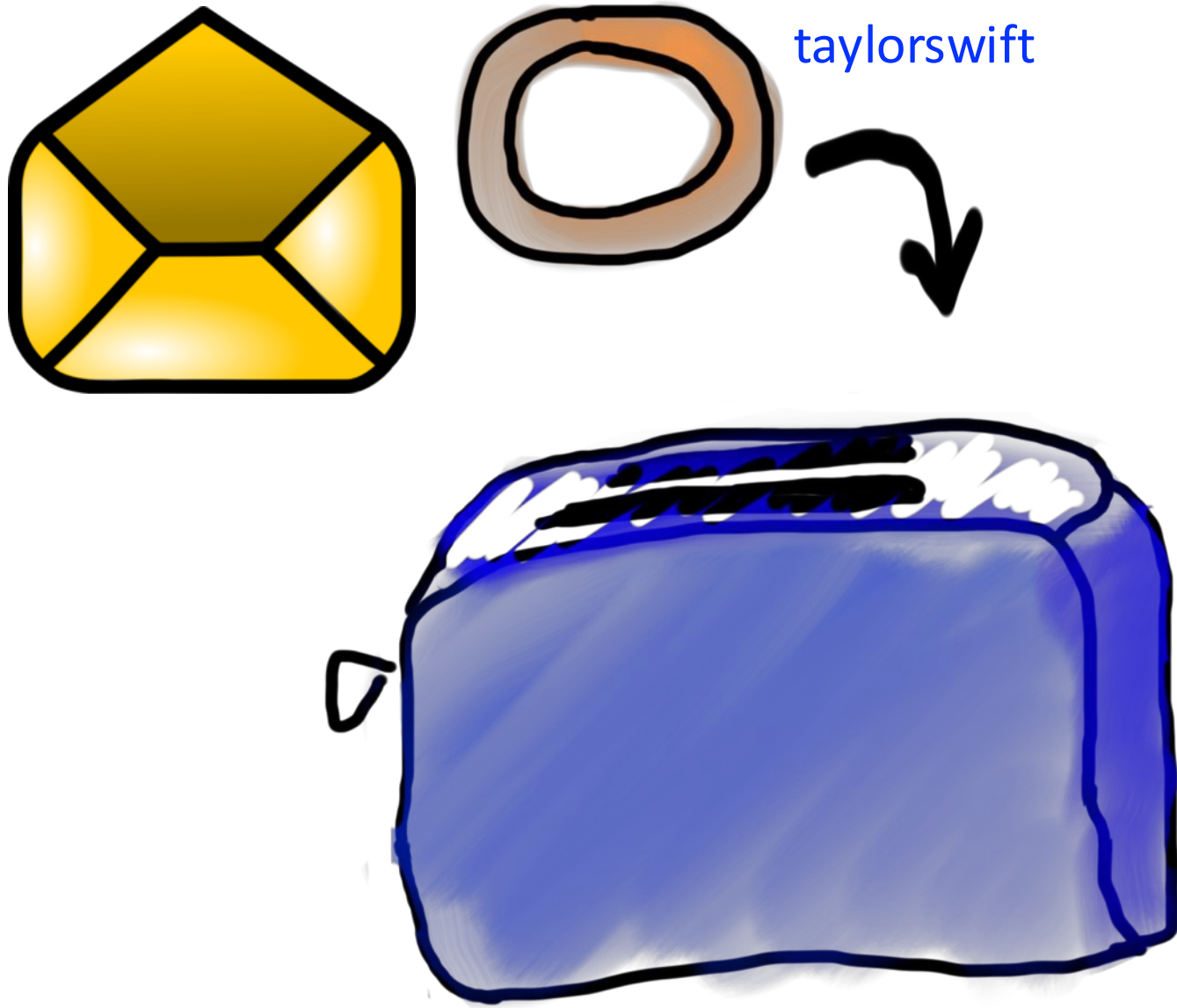
`get_status`

Requests are like Remote Method Calls



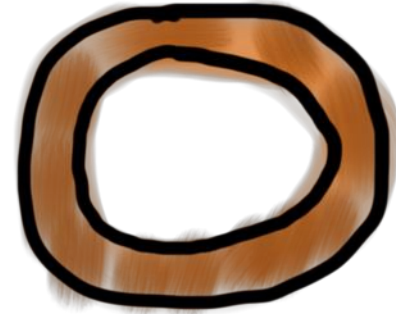
get_status

Requests are like Remote Method Calls

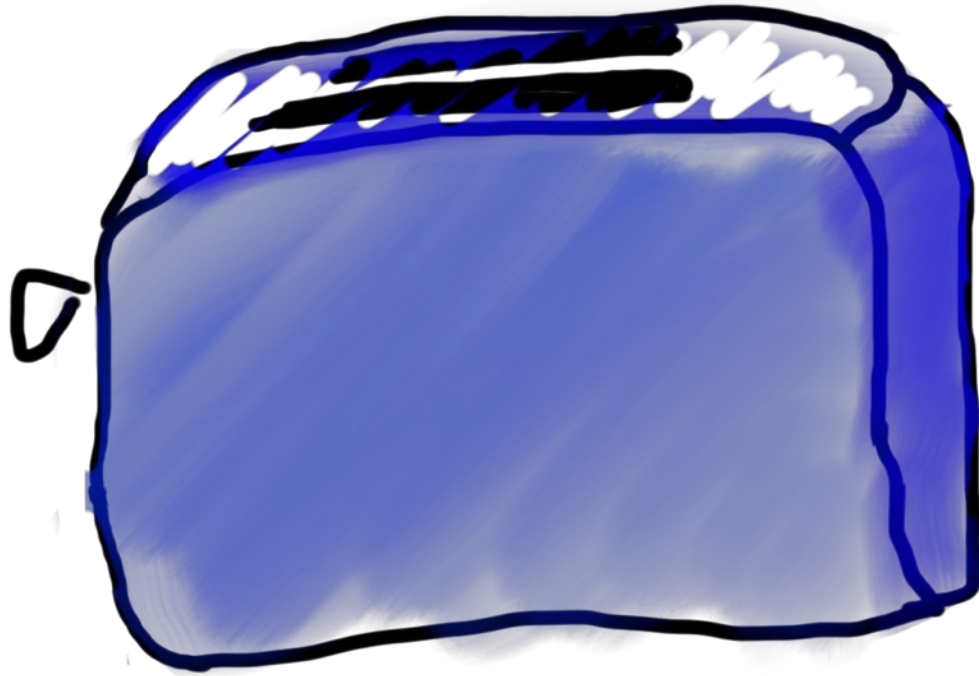


get_status

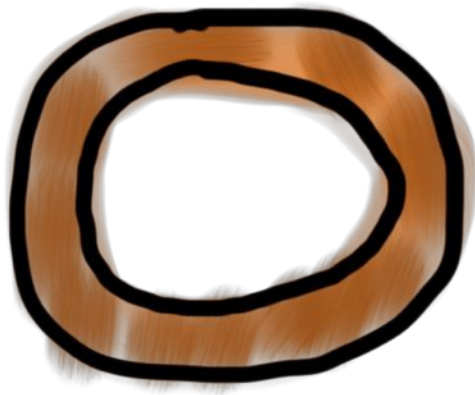
Requests are like Remote Method Calls



singing



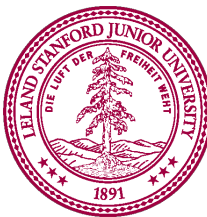
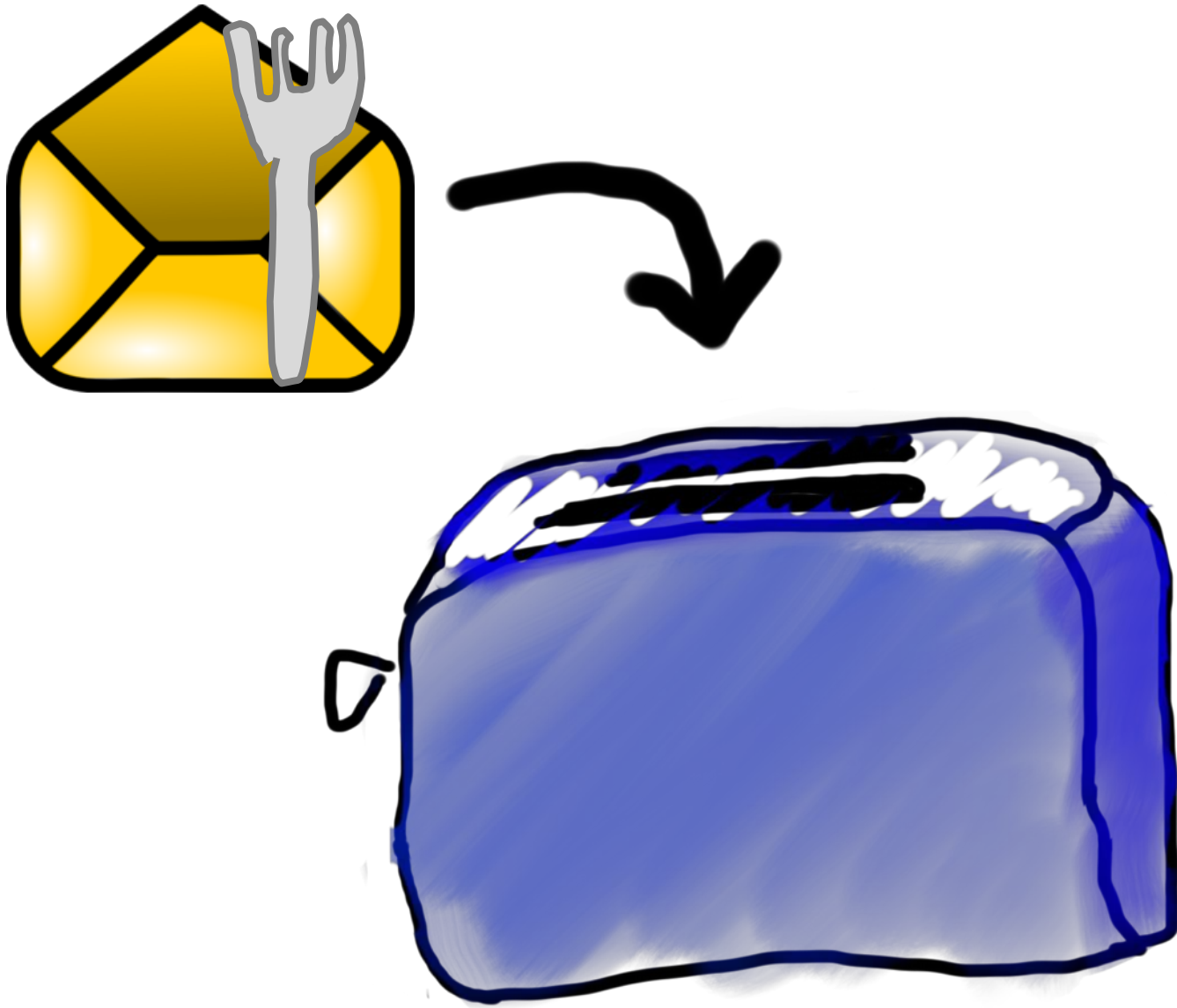
```
def handle_request(self, request):  
    cmd = request.command  
    if cmd == 'get_status':  
        user = request.params['userName']  
        status = self.get_status(user)  
        return status
```



Must be a string!



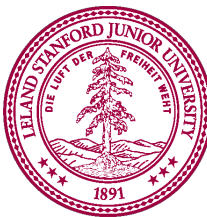
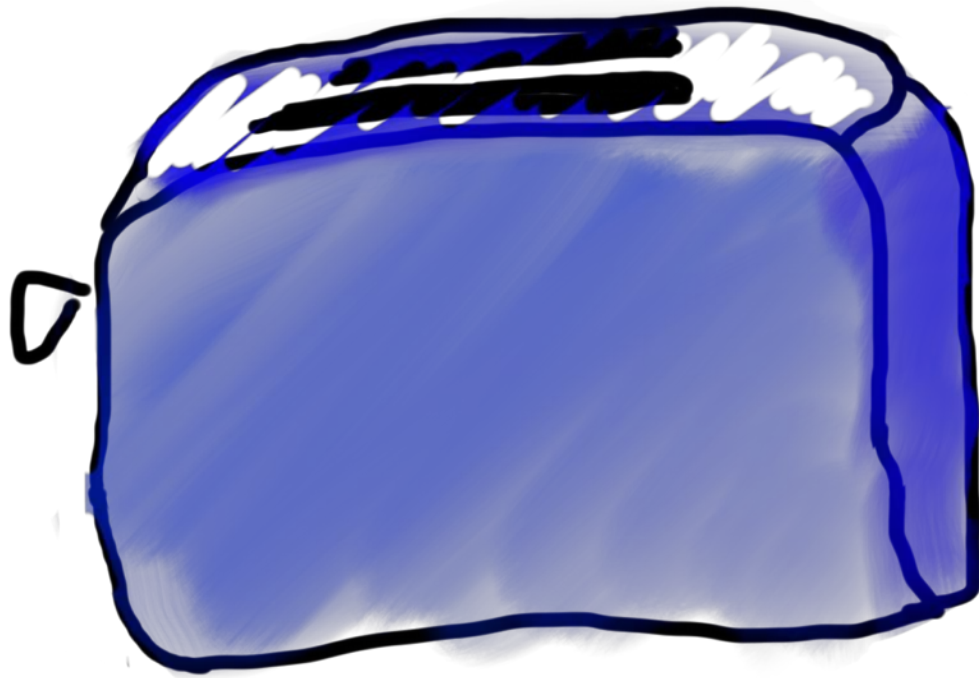
Requests are like Remote Method Calls



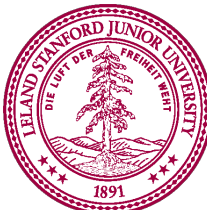
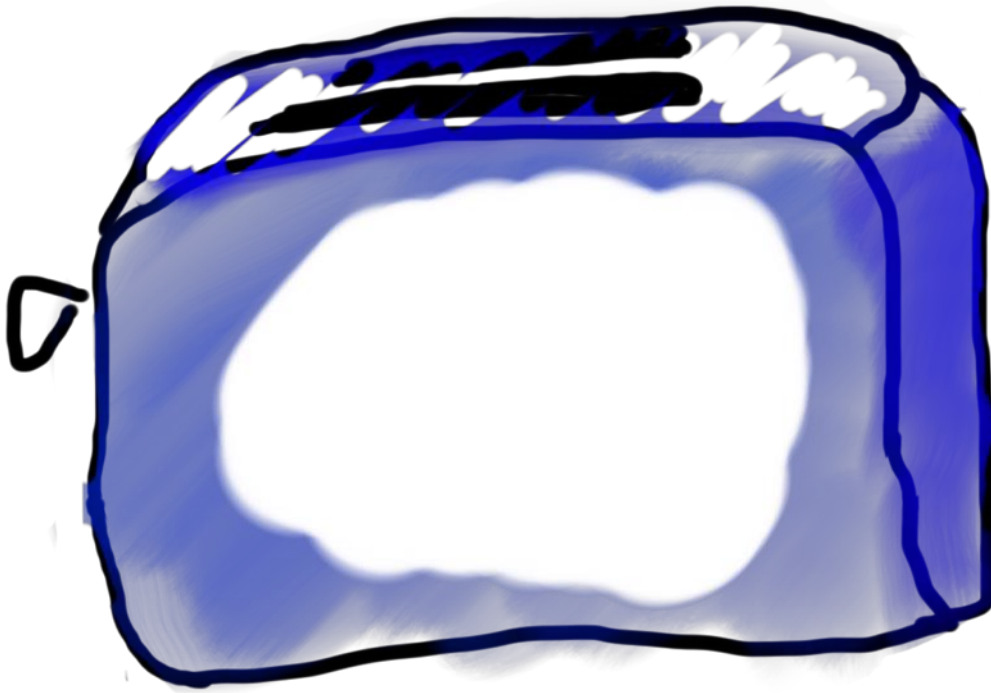
Requests are like Remote Method Calls



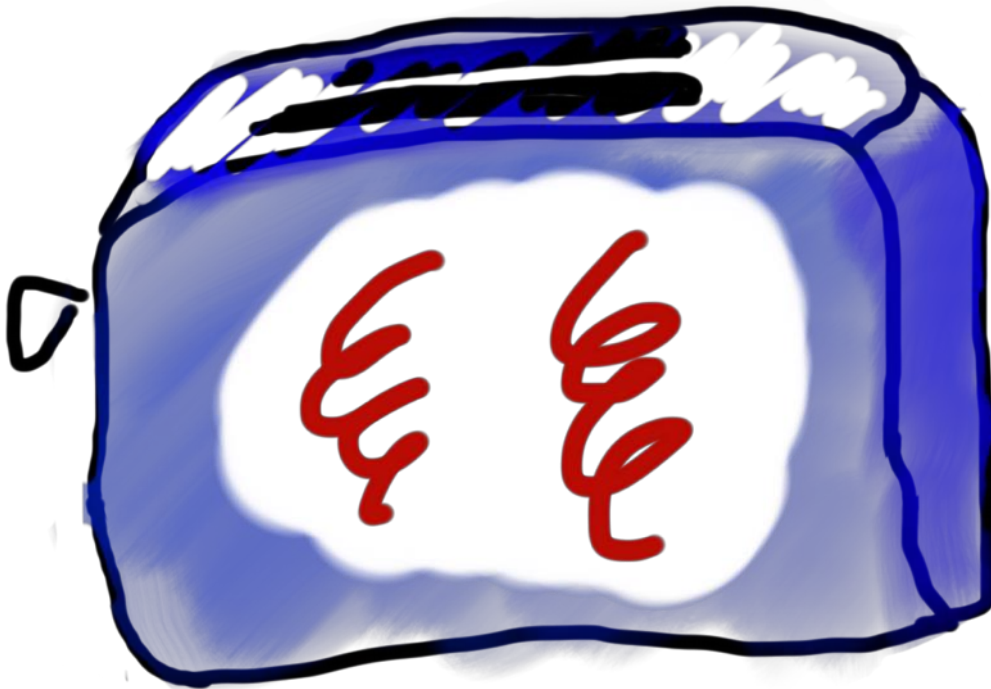
Requests are like Remote Method Calls



Requests are like Remote Method Calls



Requests are like Remote Method Calls



Internet sends data as strings...

How do you send a list or a dictionary?



Requests responses are
strings, often encoded
using JSON



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
import json
```

```
# load data
```

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

```
# save data
```

```
json.dump(data, open('ages.json'))
```



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
import json
```

```
# load data
```

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

```
# save data
```

```
json.dump(data, open('ages.json'))
```



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
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

index.html

Chat Client

Send The internet is a wild place...

Messages

Refresh

> [Chris] Hello world?

> [Laura] Here I am!!

> [Laura] This is fun!

> [Chris] Wahooooo :-)

> [Chris] We are on the internet...

> [Chris] This is like low-budget WhatsApp

> [Laura] But we made it, which is cool.

> [Terry] Hi everyone! Terry here too

> [Laura] Hi Terry!

> [Terry] The internet is a wild place...

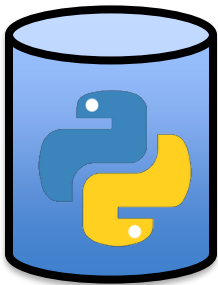
```
backend — Python chat_server.py — 93x34
...6a/2020-Spring/Lectures/Lecture22/Sandbox — Python
...tures/Lecture22/chat/backend — Python chat_server.py
~/Documents/Utilities — ngrok http 8000
~/Documents/Utilities — Python

Chris@ndoto backend % python chat_server.py
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': '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 Server

Chat Server



```
newMsg  
msg = text  
user = user
```



```
getMsgs  
index = start_index
```



```
history = [
]
```



```
newMsg
{
  'msg' : Hello world,
  'user' : 'C'
}
```

Chat Client

Hello world

Send

Chat Client

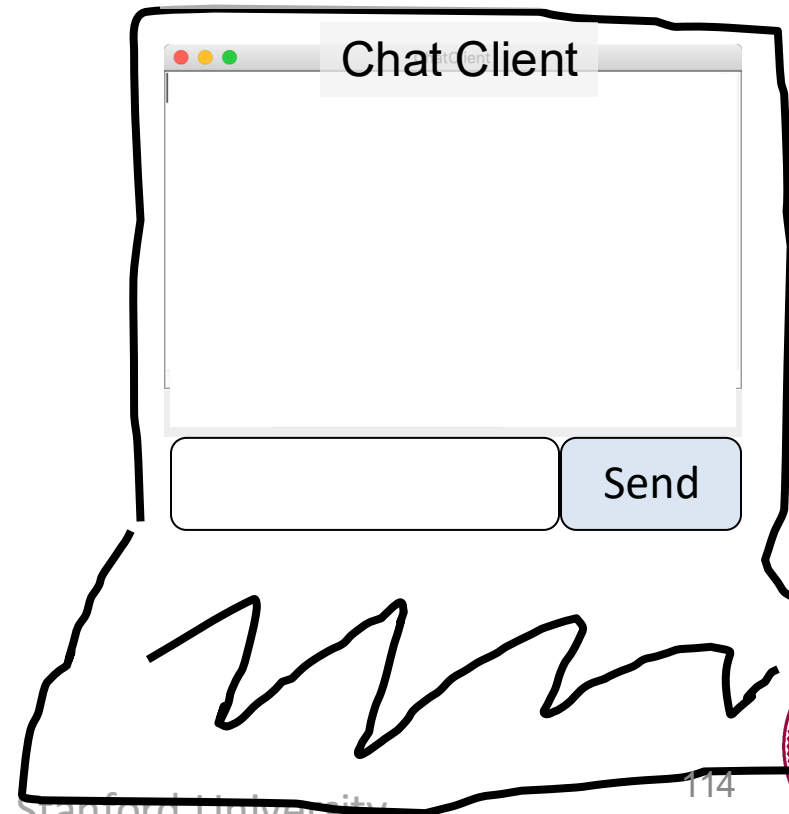
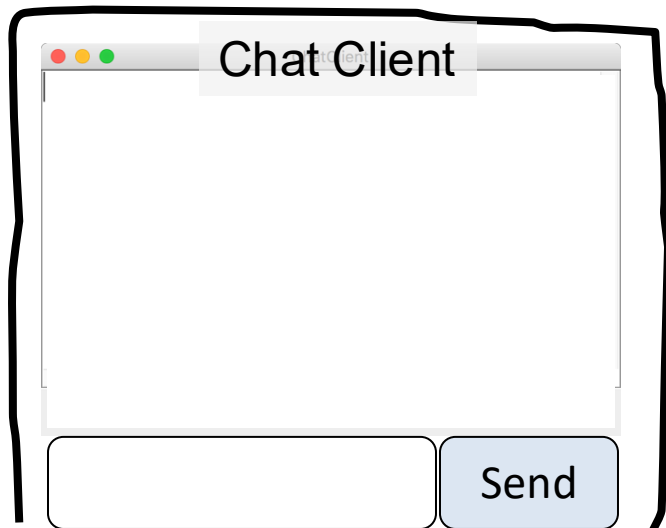
Send





```
history = [  
    '[C] Hello world'  
]
```

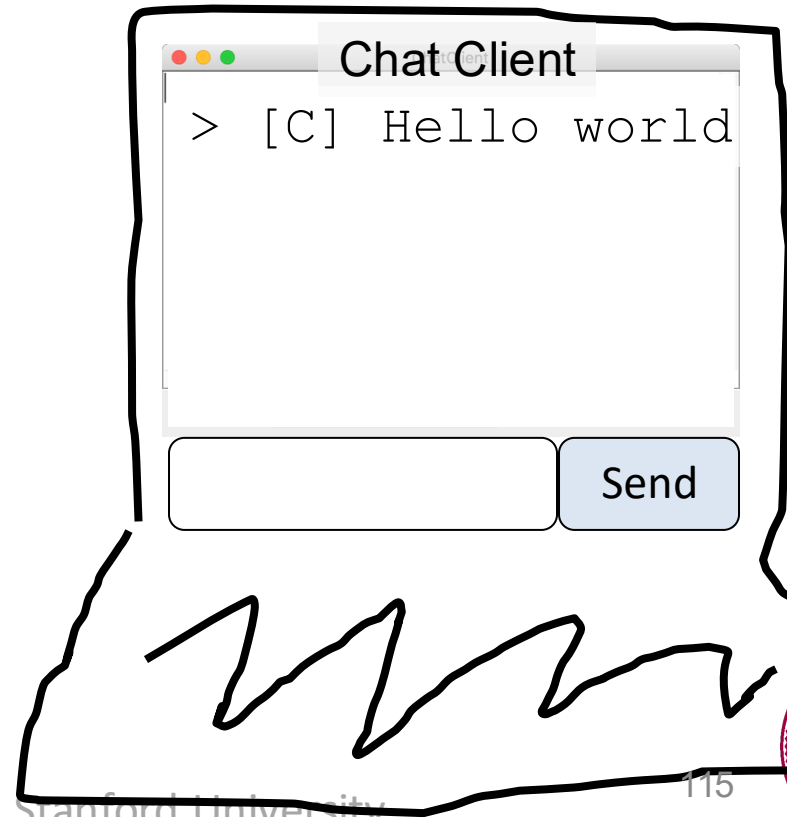
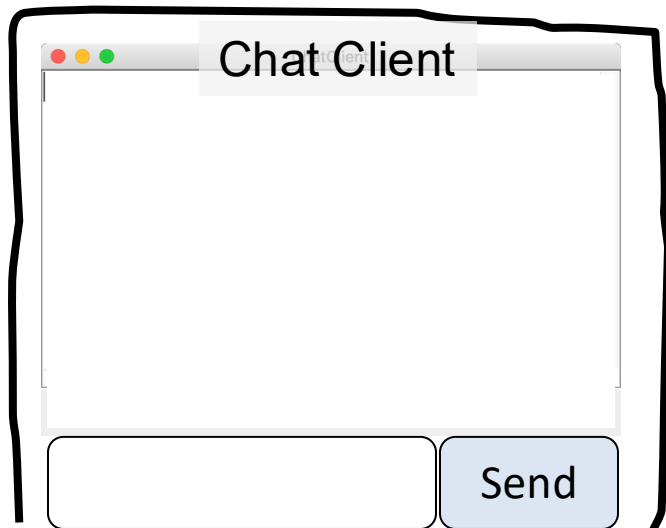
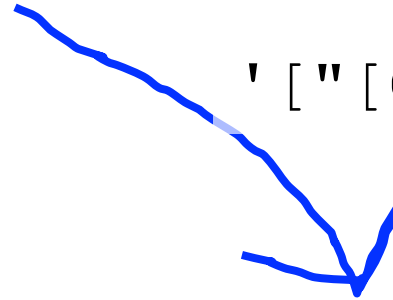
```
getMsgs  
{  
    'index' : 0  
}
```





```
history = [  
    '[C] Hello world'  
]
```

`'["[C] Hello world"]'`





```
history = [  
    '[C] Hello world'  
]
```

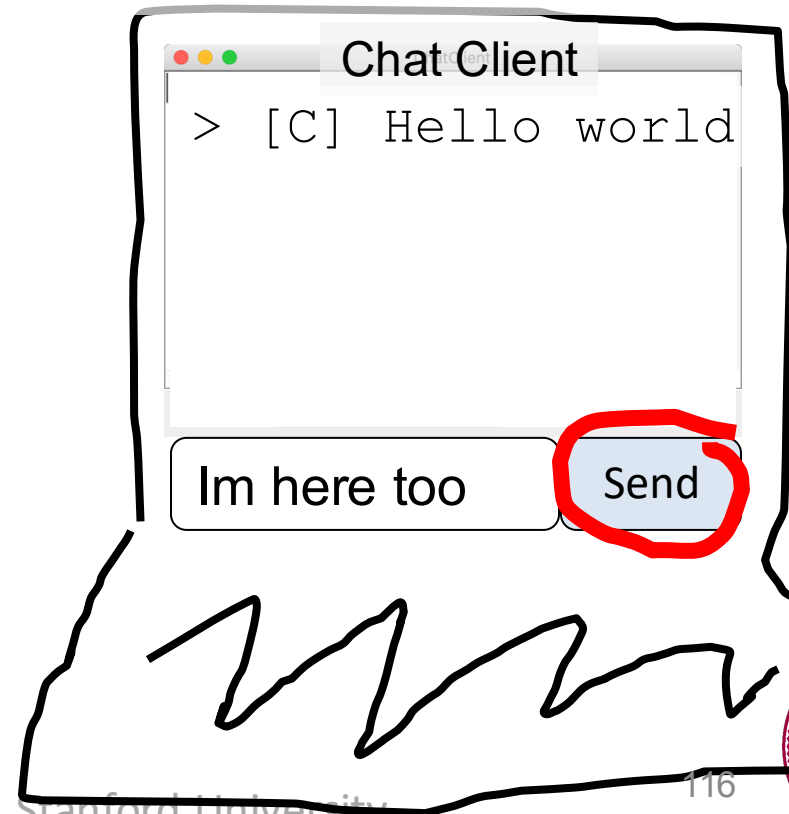
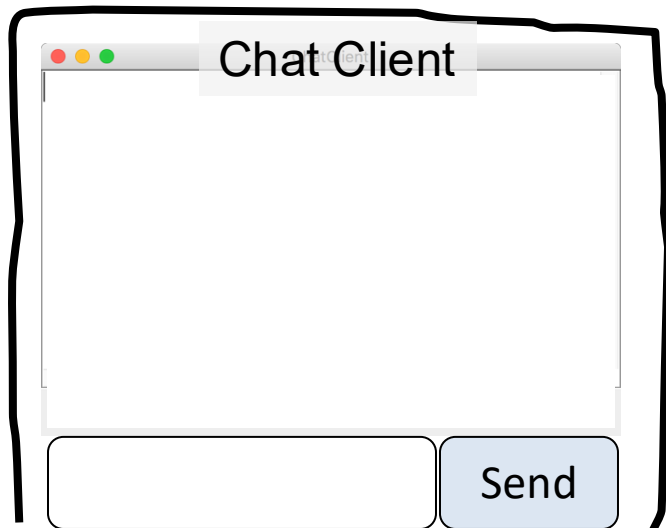
```
newMsg
```

```
{
```

```
    'msg' : 'Im here too'
```

```
    'user' : 'B'
```

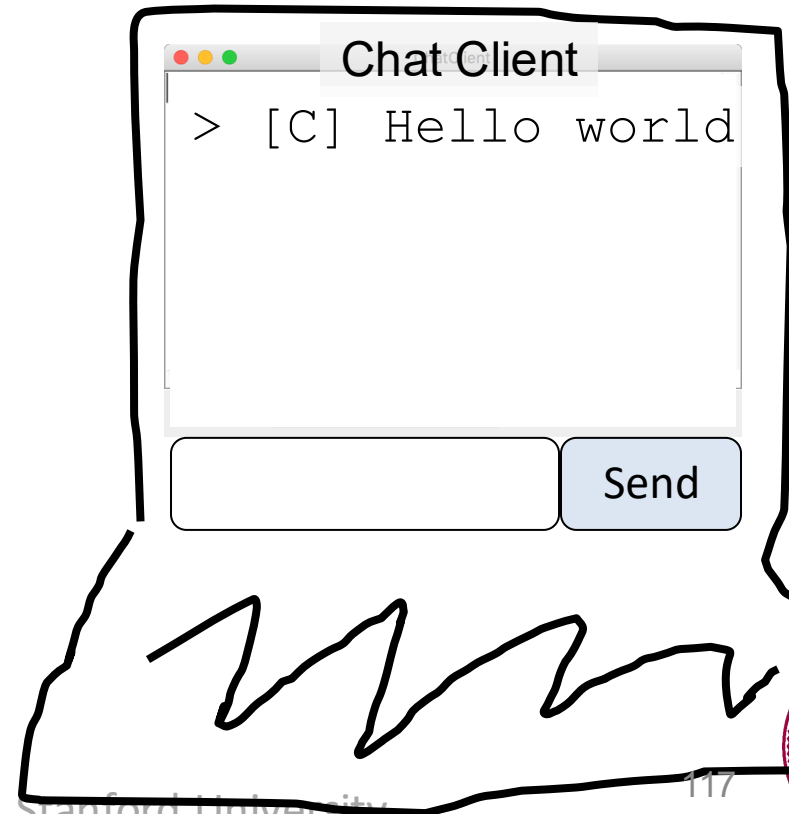
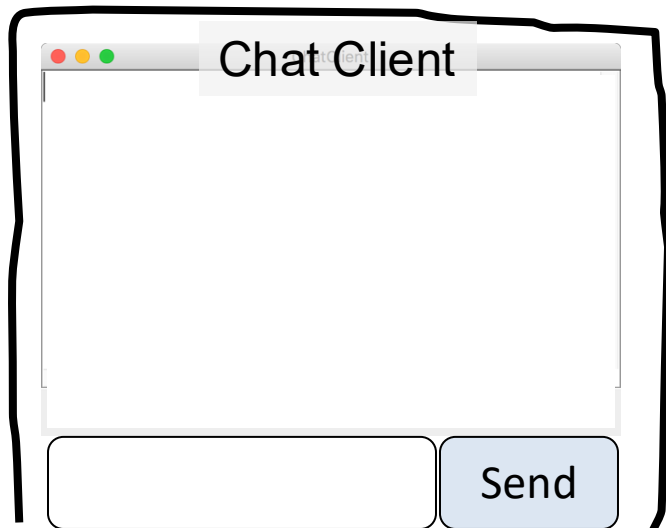
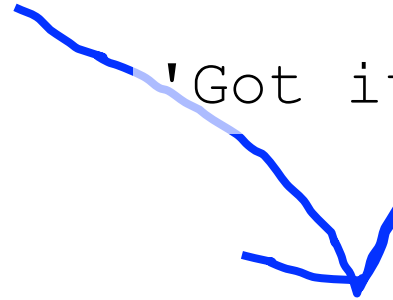
```
}
```





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

'Got it'

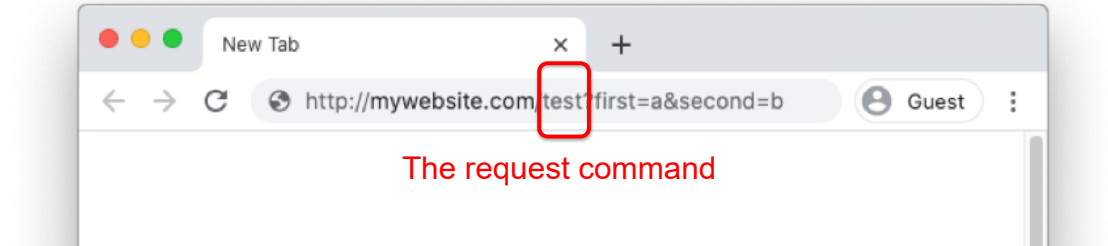
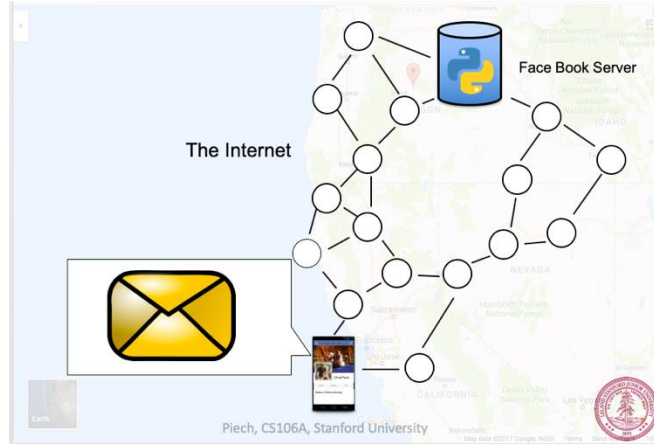


Learning Goals

1. Write a program that can respond to internet requests



Things we saw along the way

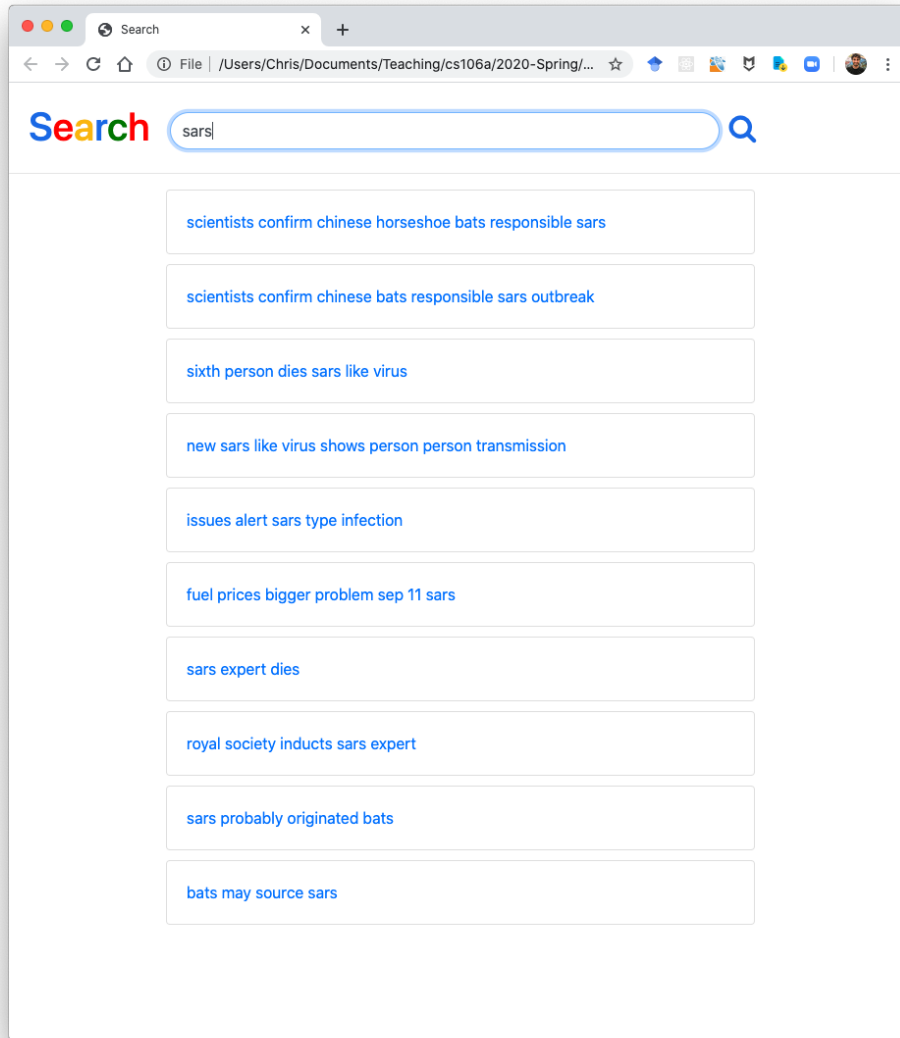


`response = requests.get(url)`

`data_str = json.dumps(data)`



Bajillion Extension



Search Engine

