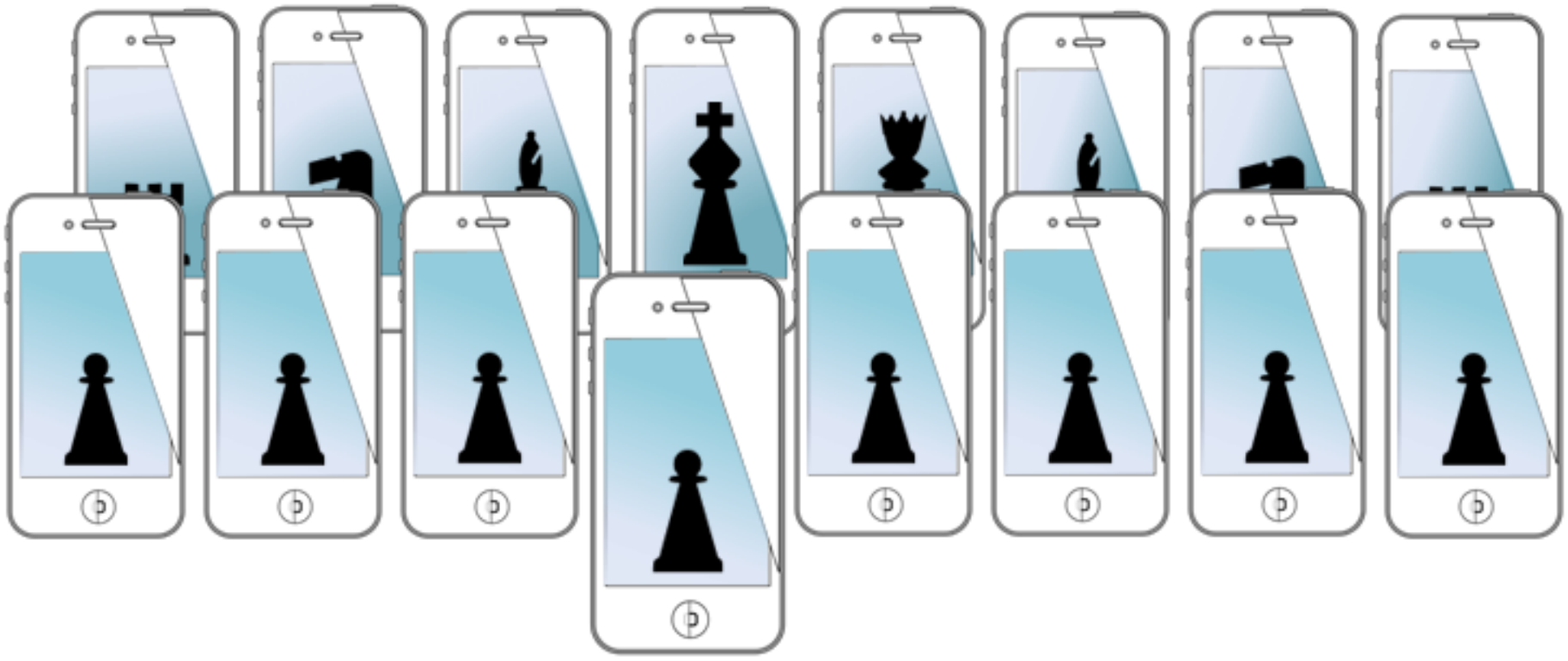
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

python crash-course, tornado

Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University

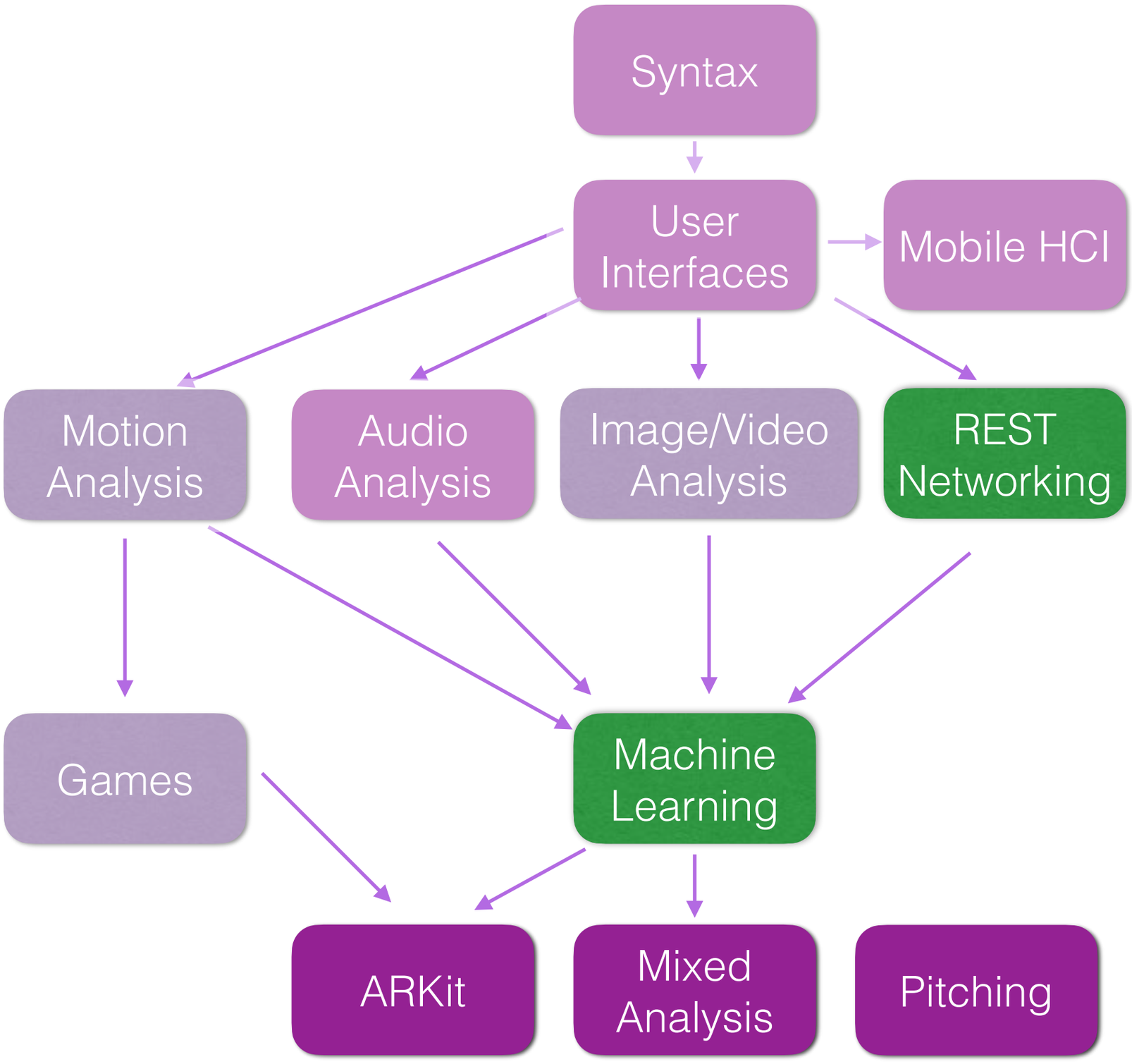
course logistics

- end of next week
  - **lab four due**: images
- two weeks following that
  - **lab five due**: machine learning as a service
  - **final project proposal due**

agenda

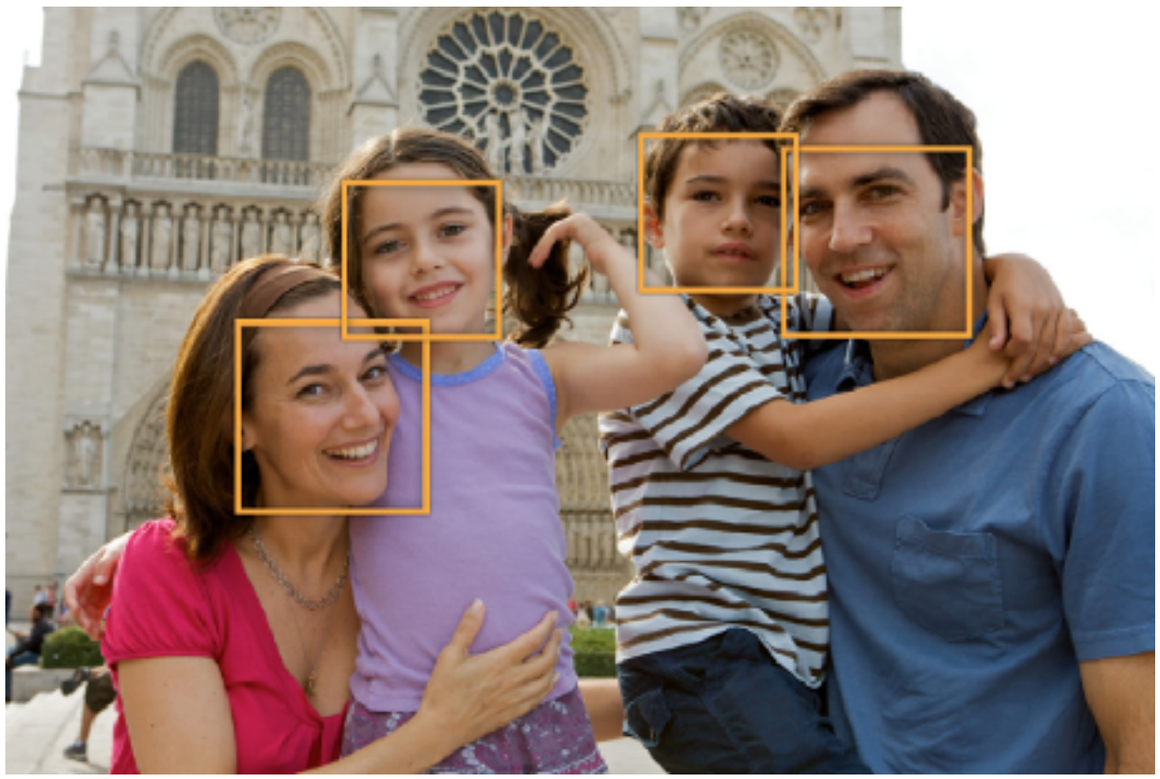
- last time: **OpenCV**
- image lab explanation
- history of python
- syntax
  - pythonic conventions
- web handling with tornado
- document databases

class overview

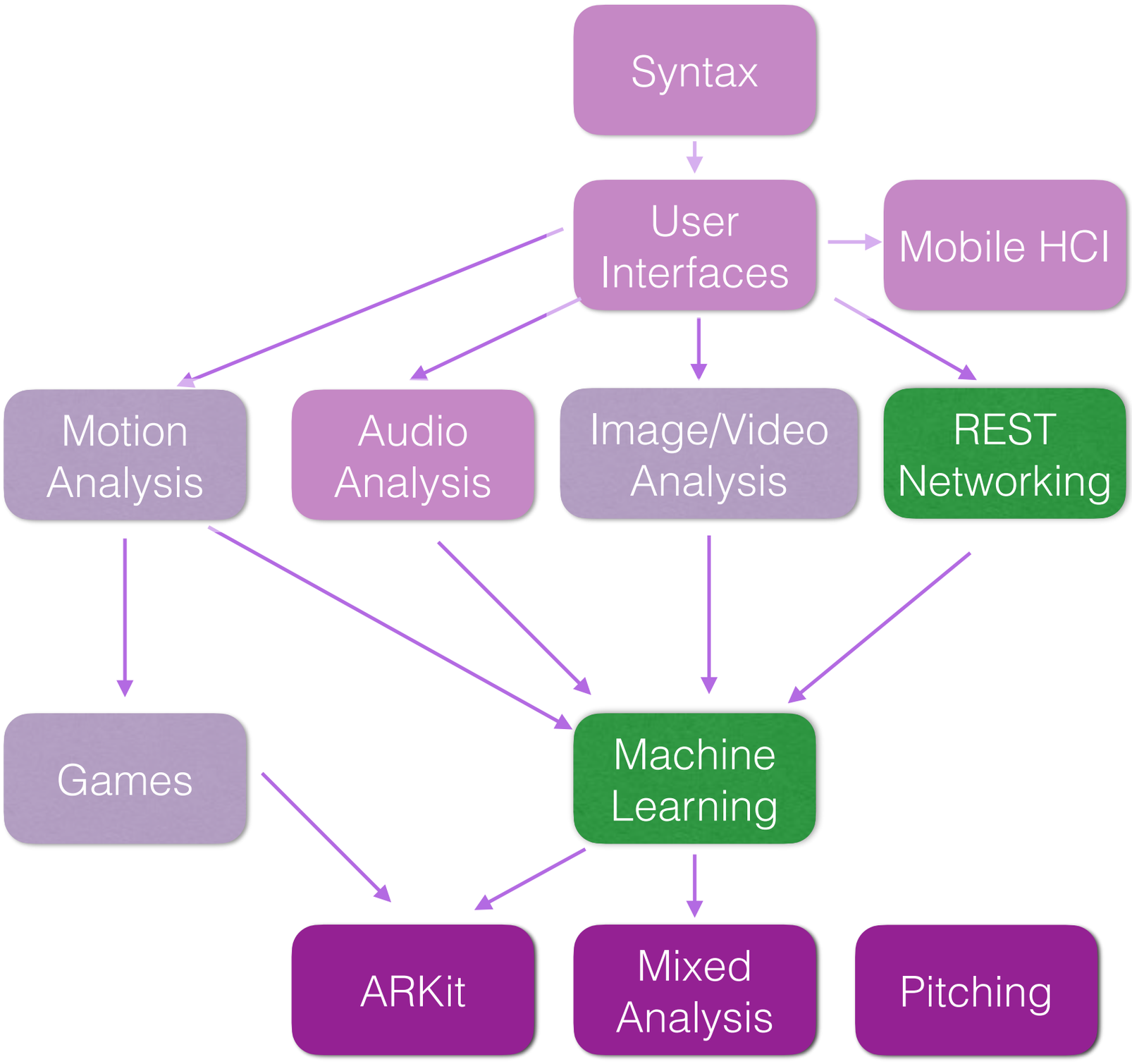


Lab Four Explanation

Lab Four Due: Core Image and OpenCV



class overview



python

- Guido van Rossum

From wikipedia:

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

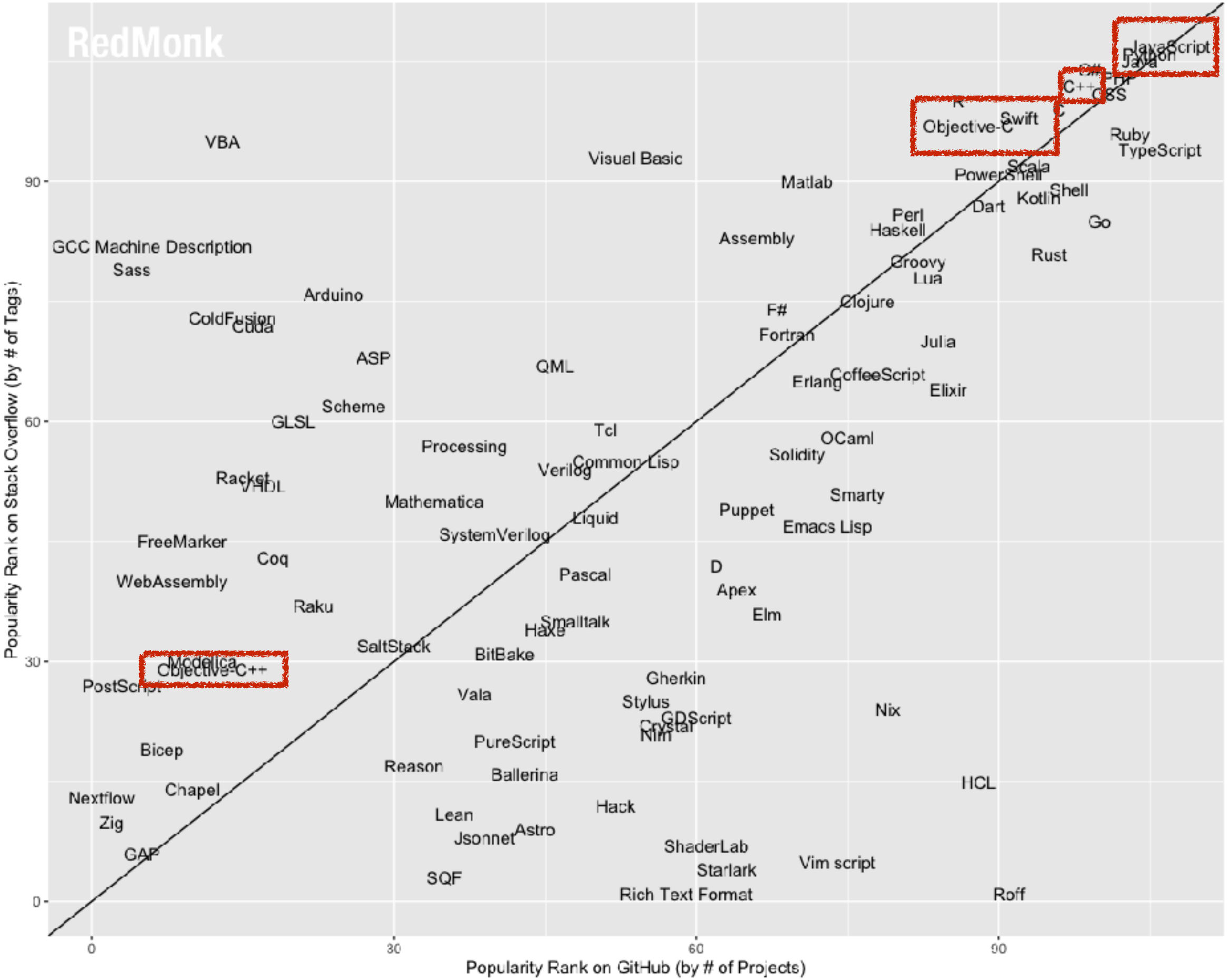
-Guido van Rossum in 1996

python adoption

- appears in every programming top three list
- 2019: Tops every list, beating out Java and Javascript
  - IEEE Spectrum, ACM, Others
- 2022: Python and Javascript swap intermittently

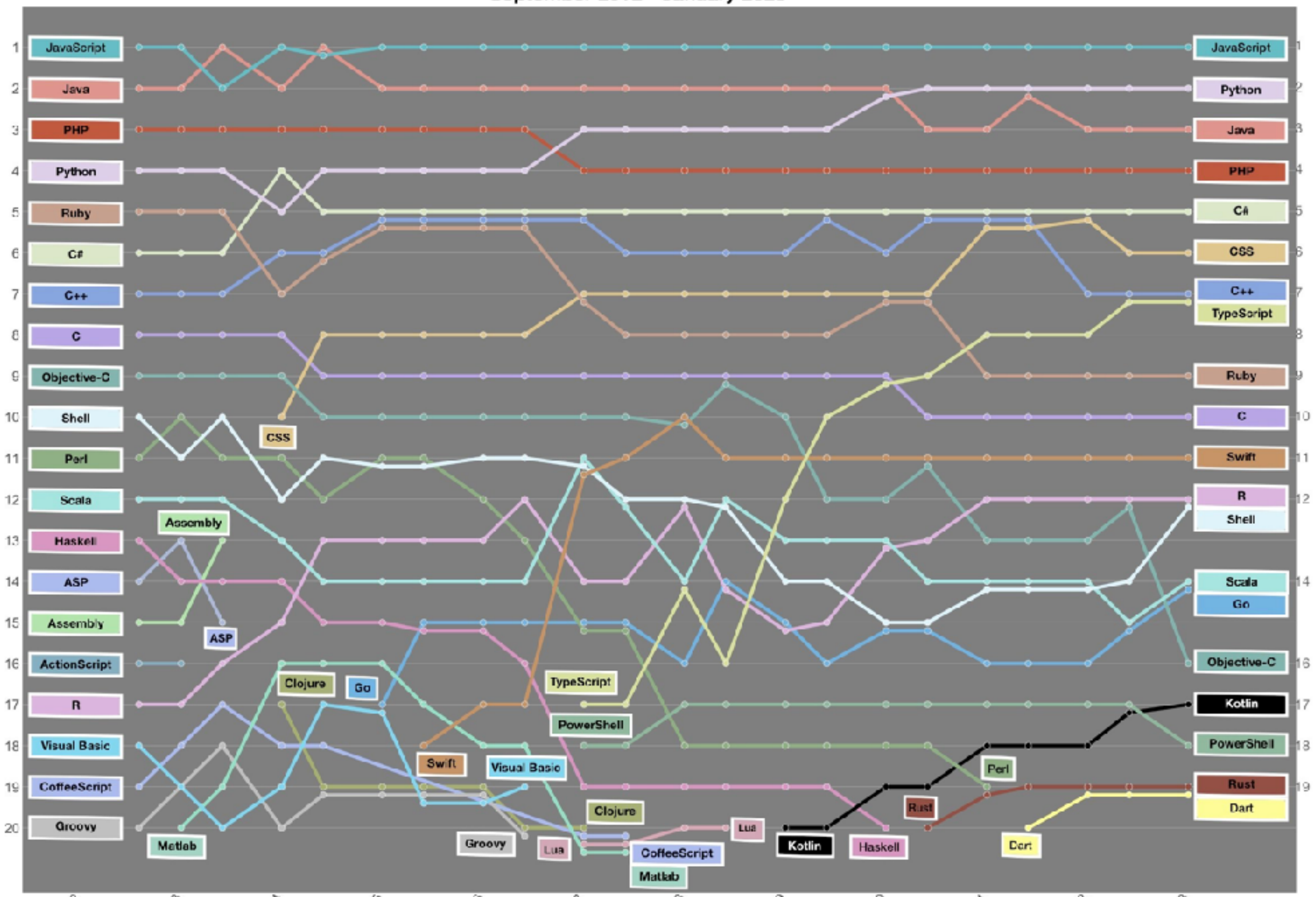
Top Programming Languages 2022 ›
Python's still No. 1, but employers love to see SQL skills

RedMonk Q123 Programming Language Rankings



RedMonk Language Rankings

September 2012 - January 2023



installation

- install anaconda and use pip3
  - **use python 3.8** (for compatibility with later package)
  - use **conda environments** or virtual environment
  - see the InstallPythonEnvironment.txt on the tornado repository
    - there are some packages we will use that require older versions of other packages, like numpy, Turi, etc.
- pick the IDE you want
  - Jupyter or sublime (not an IDE, but good for editing)
  - PyCharm, very good, supports breakpoints and watch
  - XCode can also be used, but is more limited

going through syntax…

```
if reference_slide:
    print(“make me skip that slide”)
```

syntax, python 3

reference slide

- numbers
  - int or float
- complex numbers

```
>>> 7*5
35
>>> 5/7
0.7142857142
>>> 7/5
1.4
>>> 7.0/5
1.4
```

```
>>> tmpVar = 4
>>> print (tmpVar)
4
>>> tmpVar/8
0.5
>>> tmpVar/8.0
0.5
```

```
>>> 1+1j
(1+1j)
>>> (1+1j)*5
(5+5j)
>>> 1+1j + 4
(5+1j)
```

syntax

reference slide

- strings
  - immutable

```
>>> 'single quotes'
'single quotes'
>>> "double quotes"
'double quotes'
>>> 'here is "double quotes"'
'here is "double quotes"'
>>> 'here is \'single quotes\''
"here is 'single quotes'"
>>> "here are also \"double quotes\""
'here are also "double quotes"'
```

```
>>> someString = 'MobileSensingAndLearning'
>>> someString[:5]
'Mobil'
>>> someString[5:]
'eSensingAndLearning'
>>> someString+'AndControl'
'MobileSensingAndLearningAndControl'
>>> someString*3
'MobileSensingAndLearningMobileSensingAndLearningMobileSensingAndLearning'
>>> someString[-5:]
'rning'
>>> someString[:-5]
'MobileSensingAndLea'
>>> someString[5]
'e'
>>> someString[-1]
'g'
>>> someString[-2]
'n'
```

```
>>> someString[5] = 'r'

Traceback (most recent call last):
    File "<pyshell#32>", line 1, in <module>
        someString[5] = 'r'
TypeError: 'str' object does not support item assignment
```

syntax

reference slide

- tuples
- lists
  - highly versatile and mutable
  - containers for more abstract data and objects

```
>>> aTuple = 45, 67, "not a number"
>>> aTuple
(45, 67, 'not a number')
```

immutable

```
>>> aList = ["a string",5.0,6,[4,3,2]]
>>> print(aList)
['a string', 5.0, 6, [4, 3, 2]]
>>> aList[0]
'a string'
>>> aList[2]
6
>>> aList[-1]
[4, 3, 2]
```

```
>>> anotherList = []
>>> i=0
>>> i+=1
>>> i
1
>>> while i<1000:
   anotherList.append(i)
   i+=i

>>> print anotherList
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

```
>>> len(aList)
4
>>> len(aList[-1])
3
>>> aList[0:1]=[]
>>> print(aList)
[5.0, 6, [4, 3, 2]]
>>> aList[0:2]=[]
>>> print(aList)
[[4, 3, 2]]
```

syntax loops

- for, while
  - indentation ~~matters~~ *is the only thing* that **matters**

```
i=0
while i<10:
    print (str(i) + ' is less than 10')
    i+=1
else:
    print (str(i) + ' is not less than 10')
```

```
0 is less than 10
1 is less than 10
2 is less than 10
3 is less than 10
4 is less than 10
5 is less than 10
6 is less than 10
7 is less than 10
8 is less than 10
9 is less than 10
10 is not less than 10
```

```
classTeams = ['Team','Monkey','CHC',
              'ThatGuyInTheBack',42]

for team in classTeams:
    print (team * 4)
else:
    print ('ended for loop without break')
```

```
TeamTeamTeamTeam
MonkeyMonkeyMonkeyMonkey
CHCCHCCHCCHC
ThatGuyInTheBackThatGuyInTheBackThatGuyInTheBackThatGu
168
ended for loop without break
```

Aug 28, 2018 — Even Python's retired creator **Guido van Rossum** has stated that he would not include **loop-else** in Python if he had to do it over.

syntax loops

reference slide

- for, while
  - indentation ~~matters~~ *is the only thing* that **matters**

```
for i in range(10):
    print (i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
for j in range(2,10,2):
   print (j)
```

```
2
4
6
8
```

data structures

- **lists** can be used as a stack

```
>>> classTeams = ['Team','Monkey','CHC','ThatGuyInTheBack',42]
>>> classTeams.pop()
42
>>> classTeams.pop()
'ThatGuyInTheBack'
>>> classTeams.sort()
>>> classTeams
['CHC', 'Monkey', 'Team']
```

- or import **queues**

```
>>> from queue import Queue
>>> q = Queue(maxsize=20)
>>> q.put('Team'), q.put('Eric'), q.put('Rocks')
>>> q.get()
'Team'
```

- **dictionaries**

```
>>> myDictionary = {"teamA":45,"teamB":77}
>>> myDictionary
{'teamA': 45, 'teamB': 77}
>>> myDictionary["teamA"]
45
```

lists and loops

- comprehensions

```
>>> timesFour = [x*x*x*x for x in range(10)]
>>> timesFour
[0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561]
```

```
from random import randint
grades = ['A','B','C','D','F']
teamgrades = [grades[randint(0,4)] for t in range(8)]
print (teamgrades)
```

```
['C', 'A', 'B', 'F', 'A', 'C', 'A', 'D']
```

can be nested as much as you like!

only **pythonic** if it makes the code **more readable**

```
>>> timesFour = {x:x*x*x*x for x in range(10)}
>>> timesFour
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256, 5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

can use comprehensions with dictionaries too!

lists and loops

reference slide

```
>>> timesFour = {x:x*x*x*x for x in range(10)}
>>> timesFour
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256, 5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

can use comprehensions with dictionaries too!

```
from random import randint

teams = ['CHC','Team','DoerrKing','MCVW','etc.']
grades = ['A','B','C','D','F']
teamgrades = {team:grades[randint(0,4)] for team in teams}
teamgrades

{'etc.': 'F', 'CHC': 'A', 'DoerrKing': 'B', 'MCVW': 'B', 'Team': 'A'}
```

pop quiz!

- add the numbers from 0 to 100, not including 100

```
sumValue = 0
for i in range(100):
    sumValue += i

print (sumValue)
print (sum(range(100)))
print (100*(100-1)/2)
```

more pythonic?

or use real math

now, print the **index** and **value** of elements in a list

```
list = [1,2,4,7,1,5,6,8]

for i in range(len(list)):
    print(str(list[i]) + " is at index " + str(i))

for i,element in enumerate(list):
    print(f"{element} is at index {i}")
```

```
1 is at index 0
2 is at index 1
4 is at index 2
7 is at index 3
1 is at index 4
5 is at index 5
6 is at index 6
8 is at index 7
```

more pythonic

conditionals

self test slide

- if, elif, else, None, is, or, and, not, ==

```
a=5
b=5

if a==b:
    print (“Everybody is a five!”)
else:
    print (“Wish we had fives…”)


a=327676
b=a

if a is b:
    print (“These are the same object!”)
else:
    print (“Wish we had the same objects…")

a=327676
b=327675+1

if a is b:
    print (“These are the same object!”)
else:
    print (“Wish we had the same objects…")

a=5
b=4+1

if a is b:
    print (“Everybody is a five!”)
else:
    print (“Wish we had fives…")
```

```
Everybody is a five!
```

```
These are the same object!
```

```
Wish we had the same objects
```

small integers are cached
strings behave the same

```
Everybody is a five!
```

conditionals

reference slide

```
teacher = "eric"

if teacher is not "Eric":
    print (“Go get the prof for this class!”)
else:
    print (“Welcome, Professor!”)
```

Go get the prof …

```
teachers = ["Eric","Paul","Ringo","John"]

if "Eric" not in teachers:
    print (“Go get the prof for this class!”)
else:
    print (“Welcome, Professor!”)
```

Welcome!

```
teachers = ["Eric","Paul","Ringo","John"]
shouldCheckForTeacher = True

if "Eric" not in teachers and shouldCheckForTeacher:
    print (“Go get the prof for this class!”)
elif shouldCheckForTeacher:
    print (“Welcome, Professor!”)
else:
    print (“Not checking”)
```

Welcome!

functions

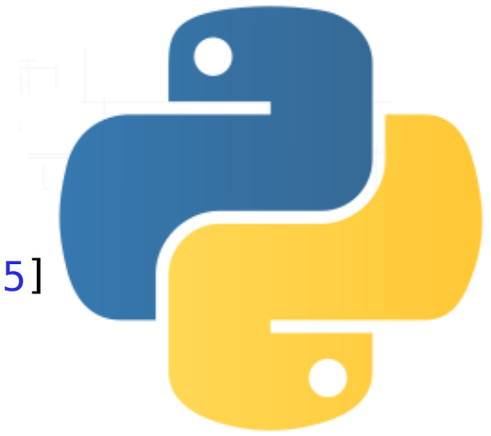
reference slide

- def keyword
  - like c, must be defined before use

```
def show_data(data):
    # print the data
    print (data)

some_data = [1,2,3,4,5]
show_data(some_data);
```

```
[1, 2, 3, 4, 5]
```



```
def show_data(data,x=None,y=None):
    # print the data
    print data
    if x is not None:
        print (x)
    if y is not None:
        print (y)

some_data = [1,2,3,4,5]
show_data(some_data);
show_data(some_data,x='a cool X value')
show_data(some_data,y='a cool Y value',x='a cool X value')

def get_square_and_tenth_power(x):
    return x**2,x**10

print (get_square_and_tenth_power(2))
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
a cool X value
[1, 2, 3, 4, 5]
a cool X value
a cool Y value

(4, 1024)
```

debugging

- the python debugger
  - http://docs.python.org/2/library/pdb.html
  - or use break points in pycharm
- `import pdb, pdb.set_trace()`
- command line arguments
  - `s`(tep), `c`(ontinue), `n`(ext), `w`(here), `l`(ist), `r`(eturn), `j`(ump)
  - and much more… like `print, p, pp`
  - can set numbered break points by running from python window
    - `python -m pdb your_function.py`

classes

reference slide

- multiple inheritance
- “self” is always passed as first argument

```
class BodyPart(object):
    def __init__(self,name):
        self.name = name;


class Heart(BodyPart):
    def __init__(self,rate=60,units="minute"):
        self.rate = rate
        self.units= units
        super().__init__("Heart")

    def __str__(self):
        print (“name:" + str(self.name) + " has " + str(self.rate) + " beats per " + self.units)


myHeart = Heart(1,"second")
print(myHeart)
```

python generators

reference slide

- kinda like static variables
- used to create iterables
- lots more that you can do, like send in values

```
def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 1

total = 2
for next_prime in get_primes(3):
    if next_prime < 2000000:
        total += next_prime
    else:
        break
```

yield allows iteration

https://jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-generators-explained/

python syntax “with”

reference slide

- the “with” statement
- defines an “enter” and an “exit” protocol
- used commonly for opening files, where “open” adopts the “with” protocol

```
file = open("/some_file.txt")
try:
    data = file.read()
finally:
    file.close()



with open("/some_file.txt") as file:
    data = file.read()
```

python decorators

reference slide

- wrap your method inside another method
- the wrapper changes some functionality

```
from time import sleep

def sleep_decorator(function):
    def wrapper(*args, **kwargs):
        sleep(2)
        return function(*args, **kwargs)
    return wrapper

@sleep_decorator
def print_number(num):
    return num

print(print_number(222))

for num in range(1, 6):
    print(print_number(num))
```

used a **bunch** in web applications before python 3.5

python async/await

reference slide

- introduced in python 3.5: awaitable objects

co-routine:
awaitable methods

```
import asyncio

async def nested():
    return 42

async def main():
    # Nothing happens if we just call "nested()".
    # A coroutine object is created but not awaited,
    # so it *won't run at all*.
    nested()

    # Let's do it differently now and await it:
    print(await nested())  # will print "42".

asyncio.run(main())
```

tasks:
awaitable objects

```
import asyncio

async def nested():
    return 42

async def main():
    # Schedule nested() to run soon concurrently
    # with "main()".
    task = asyncio.create_task(nested())

    # "task" can now be used to cancel "nested()", or
    # can simply be awaited to wait until it is complete
    await task

asyncio.run(main())
```

futures
gathering awaitable routines

```
async def main():
    await function_that_returns_a_future_object()

    # this is also valid:
    await asyncio.gather(
        function_that_returns_a_future_object(),
        some_python_coroutine()
    )
```

why are we learning python?

- its the glue for:
  - tornado and mongodb for managing http requests in iOS
  - machine learning (blackbox approaches)
  - saving data for use in CreateML (via swift)

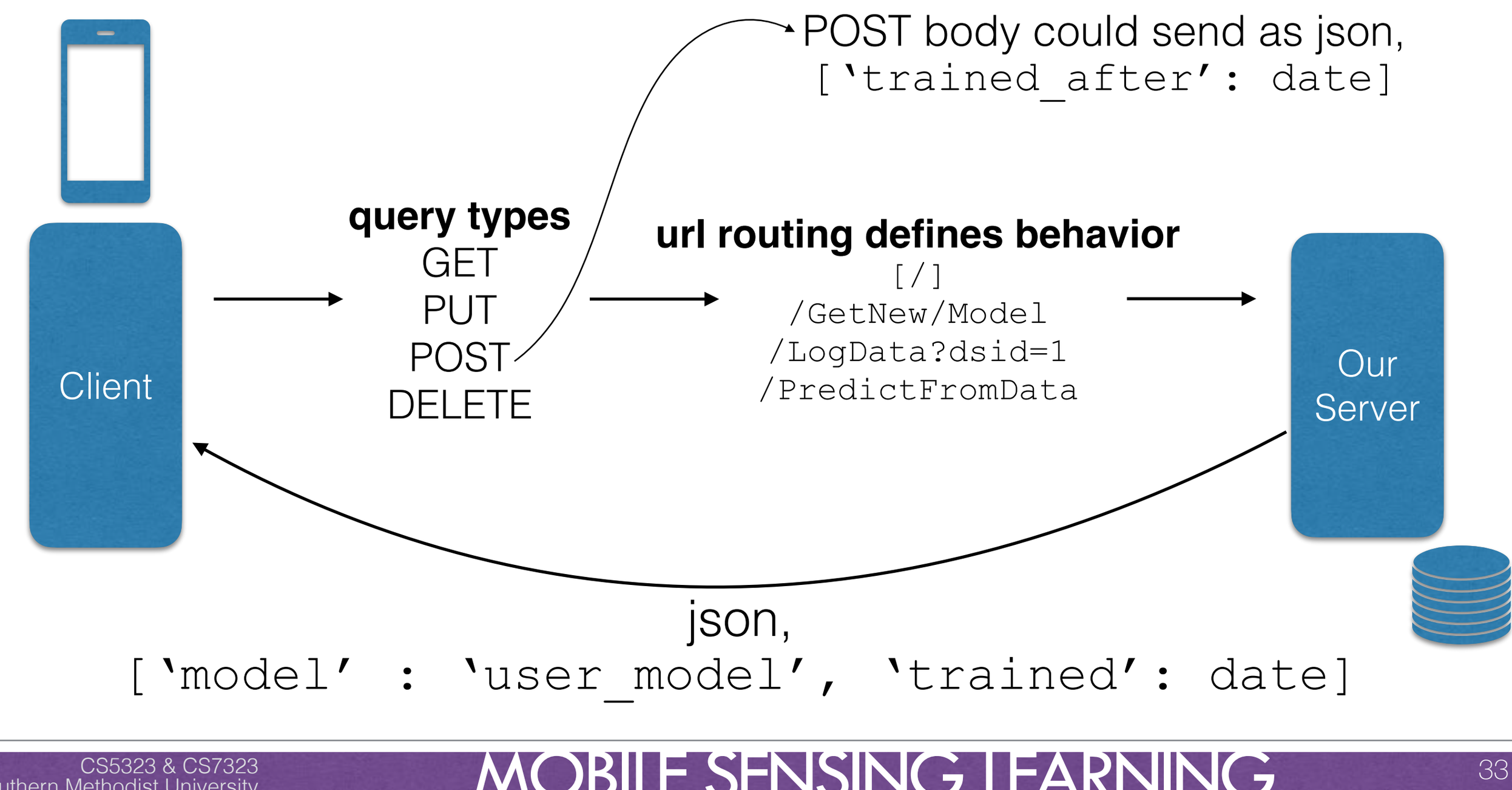
what are we doing?

- **preparing for A5**, need HTTP server that can:
  - accept (any) data
  - save it into a database
  - learn a (ML) model from that database
  - mediate queries and training of the model
- tornado is the event-driven architecture for interpreting the commands, routing the data, etc.
- our focus is building a deployment server, not an advanced ML algorithm (take DM or ML courses for that)

REST API

representational state transfer architectural style

- Specifies a design for how we can interact remotely with a server to post and query data.



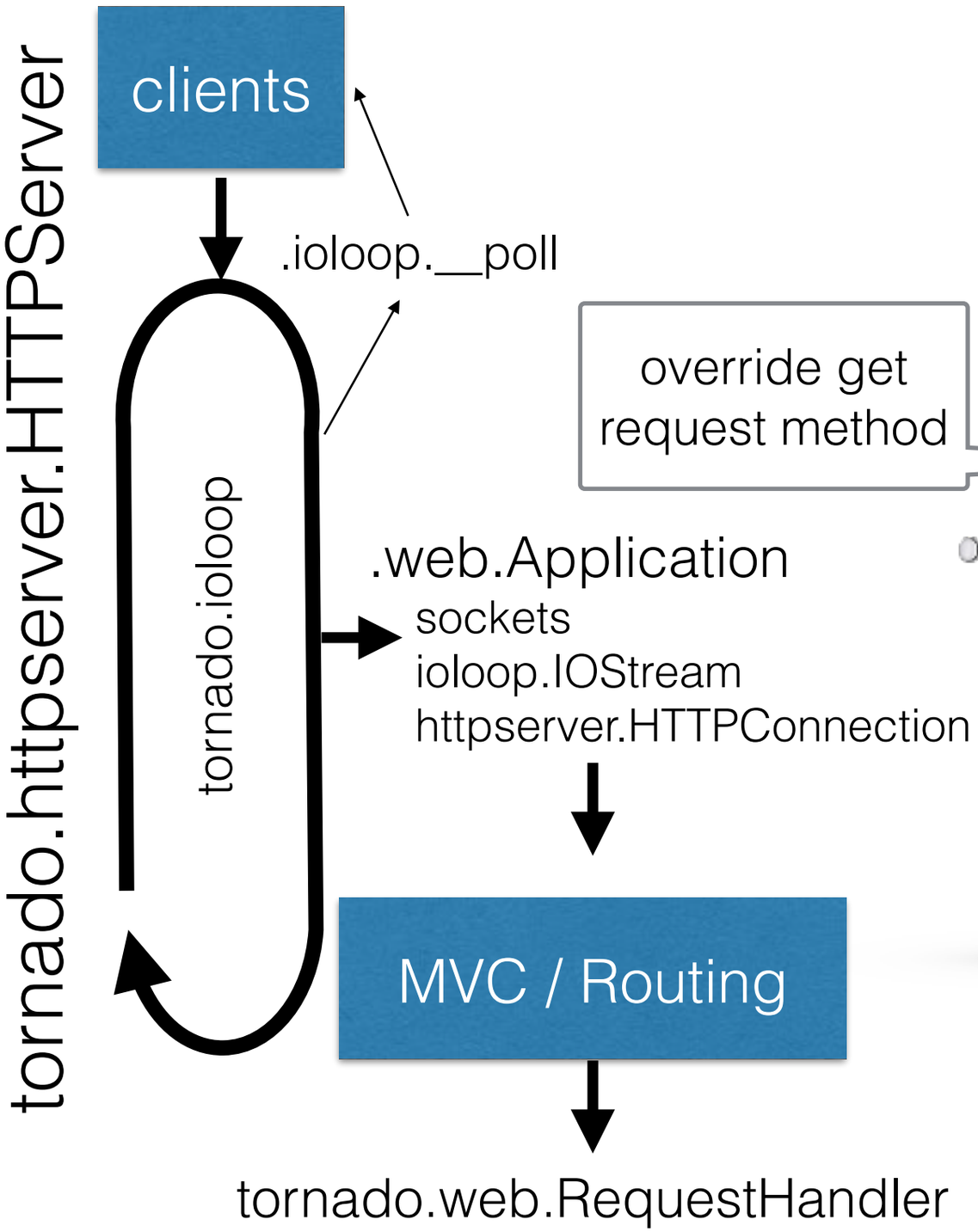
tornado web

https://www.tornadoweb.org/en/stable/

- non-blocking web server
  - built for short-lived requests (pipelined)
  - and long lived connections
- built to scale
  - an attempt to solve the 10k concurrent problem

- has a python wrapper implementation
  - open sourced by Facebook after acquiring friendfeed.com
  - originally developed by the developers of gmail and google maps (the original releases)
- uses IOLoop and callback model

tornado



override get request method

new class, inherit from RequestHandler

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, MSLC World")

application = tornado.web.Application([
    (r"/main", MainHandler),
])


application.listen(8888)
tornado.ioloop.IOLoop.instance().start()
```

URL routing and handler list

start the IO loop

listen on 8888

tornado: get request

- get requests with arguments

```
class GetExampleHandler(tornado.web.RequestHandler):
    def get(self):
        arg = self.get_argument("arg", None, True) # get arg
        if arg is None:
            self.write("No 'arg' in query")
        else:
            self.write(str(arg)) # spit back out the argument
```

- how many connections?
  - one front end of Tornado~3,000 in 1 second
  - with nginx and four instances of tornado
    - anywhere from 9,000-17,000
  - caveat: as long as you do not block the thread!

sub-classing application

```
# tornado imports
import tornado.web
from tornado.web import HTTPError
from tornado.httpserver import HTTPServer
from tornado.ioloop import IOLoop
from tornado.options import define, options

# Setup information for tornado class
define("port", default=8000,
       help="run on the given port", type=int)

CUSTOM CLASSES AND DEFINITIONS

def main():
    '''Create server, begin IOLoop
    '''
    tornado.options.parse_command_line()
    http_server = HTTPServer(Application(), xheaders=True)
    http_server.listen(options.port)
    IOLoop.instance().start()

if __name__ == "__main__":
    main()
```

We need to write the Application class to meet desired functionality

sub-classing application

```
# custom imports
from basehandler import BaseHandler
import examplehandlers

# Utility to be used when creating the Tornado server
# Contains the handlers and the database connection
class Application(tornado.web.Application):
    def __init__(self):
        '''Store necessary handlers,
           connect to database
        '''

        handlers = [(r"/[/]?",
                     BaseHandler),
                    (r"/Test[/]?",
                     examplehandlers.PostHandler),
                    (r"/DoPost[/]?",
                    MORE HANDLERS AND URL PATHS
                    ]

        settings = {'debug':True}
        tornado.web.Application.__init__(self, handlers, **settings)

        SETUP DATABASE

    def __exit__(self):
        self.client.close()
```

I wrote the base handler for you

handlers should subclass it

call the super class init

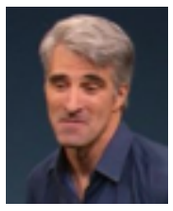
more to come in a moment

BaseHandler

- check out what it does
  - built for analyzing and writing back json
  - implements both get and post requests
  - put this in the main python file to access these:

```
# custom imports
from basehandler import BaseHandler
import examplehandlers
```

we will explore this more in the demo to come!



post requests

- identical handling code in python
- in our implementation, return json

```
class PostHandler(BaseHandler):
  def get(self):
   '''respond with arg1*2
   '''
   arg1 = self.get_float_arg(“arg1",default=0.0);
   self.write("Get from Post Handler? " + str(arg1*2));

  def post(self):
   '''Respond with arg1 and arg1*4
   '''
   arg1 = self.get_float_arg("arg1",default=1.0);
   self.write_json({"arg1":arg1,"arg2":4*arg1});
```

client sent GET to URL

client sent POST to URL

what to install for this class?

- look at installation packages list in tornado branch
- https://github.com/SMU-MSLC/tornado_bare/blob/turi_create_example/InstallPythonEnvironment.txt

From the Rosetta terminals:
CONDA_SUBDIR=osx-64 conda create "python38env" python=3.8

conda create -n "python38env" python=3.8

Instructions for all Macs:
Note that numpy must be an older version to be compatible with Turi ...

conda activate python38env
python3 -m pip install --upgrade pip
pip3 install numpy==1.23.1
pip3 install pandas
pip3 install matplotlib
pip3 install scikit-learn
pip3 install seaborn
pip3 install jupyter
pip3 install coremltools
pip3 install turicreate
pip3 install pymongo