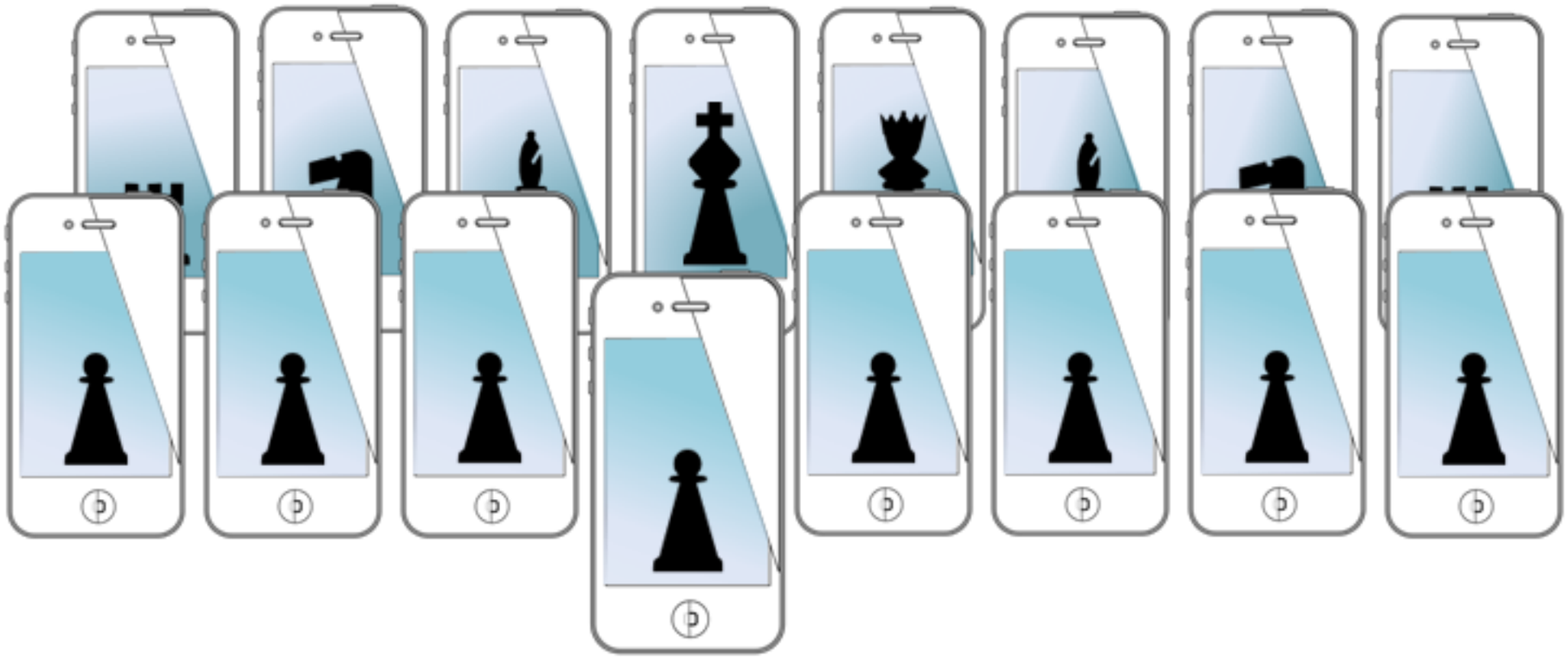
MOBILE SENSING LEARNING & CONTROL



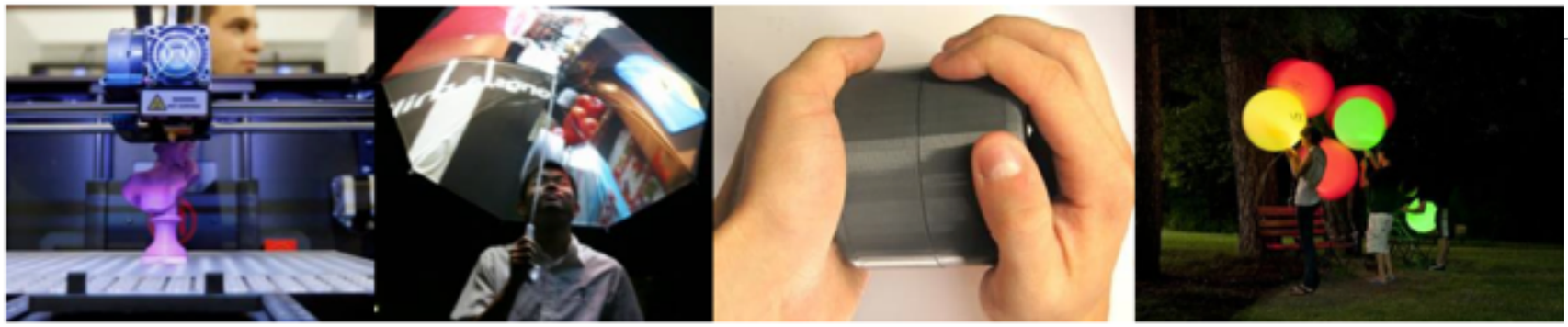
CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture seventeen: python crash-course

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

UBICOMP



- offering fall semester 2014
- MW 2:00-3:20

Ubiquitous Computing

CSE 5390 & CSE 7390

Eric C. Larson

eclarson@lyle.smu.edu

- History of UbiComp
- Human Computer Interaction
- Advanced Prototyping
  - 3D printing
  - software radio
  - scikit-learn
- Sensing and machine learning
- Wireless technologies
- Input methods for UbiComp
- Activity sensing
- Location tracking
- Wearable computing
- Assistive technology
- mobile health
- Sustainability and feedback
- Evaluation techniques
- Privacy and Security

course logistics

- A4 grades will be posted sometime next week
- A5 is due Friday after this
  - sensors? need them?

assignment 5

Assignment Five - Bluetooth Arduino Sensing

Create an iOS application using the Bluetooth template and the Arduino Bluetooth Shield that:

- Reads and displays sensor data from two or more sensors attached to the arduino
  - one sensor must use analog voltage
  - the other sensors can be analog, digital, or binary output
  - the display of the sensor data should be more than just a text label
  - use proper sampling rates to display the sensor output
  - use proper dynamic range and voltage references
- Sends two or more control commands to the microcontroller that change the behavior of the arduino
  - make the controls change something noticeable in the operation of the Arduino
  - the output must make use of a PWM signal (implemented in hardware)
- The Arduino sketch should:
  - use interrupts
    - for example, use a button as input
    - the interrupt must change something noticeable in the operation of the Arduino
  - use digital outputs
    - for example to control LED(s), pin 13 is already setup for this
  - use PWM
  - use the ADC

Use the tips from lecture for proper interfacing of sensors, buttons, timers, PWM, and interrupts.

agenda

- history of python
- syntax
  - pythonic conventions
  - examples

python

- Guido van Rossum

From wikipedia:

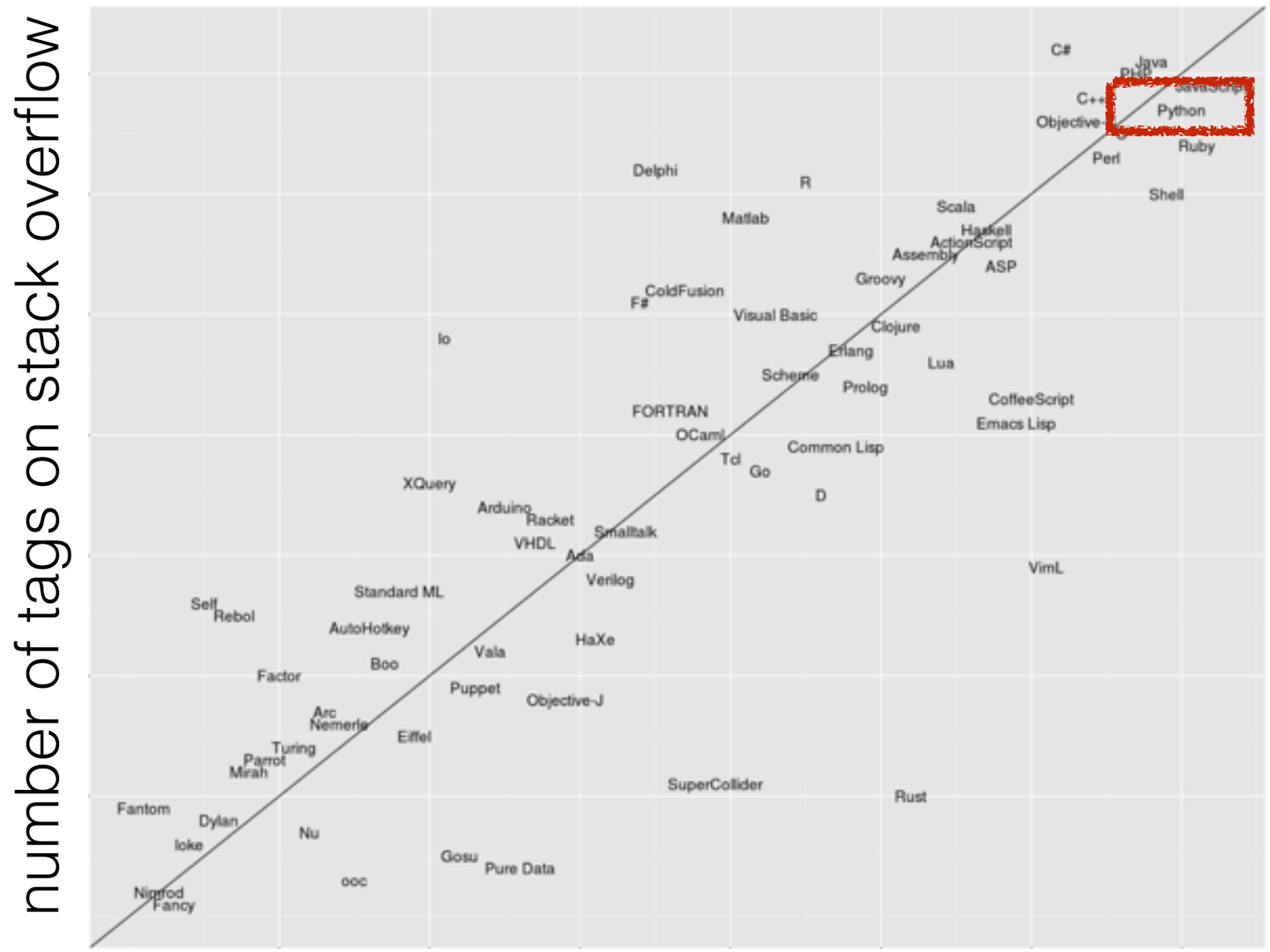
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language 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 ten list
  - hacker news #1
  - dice job list #8
  - book sales #8

you (probably) should know it!



python disclaimers

- weakly typed variables (dynamic)
- its an interpreter
  - loops are slow
  - until they are not (compile it)
- can't use parallel instructions natively
  - unless you use IPython
- more similar to objective c than you think
  - kinda
- can be the glue for your different codebases

python releases

- 1.0 (up to 1.6)
  - basic python, complex numbers, lambdas
- 2.0 (up to 2.7)
  - unified types, made completely object oriented
- 3.0 (up to 3.4, released two weeks ago)
  - eliminate multiple paradigms (kinda)
  - 2.x not necessarily compatible with 3.x

installation

- open a terminal
- do nothing
  - OSX ships with python versions 2.7 and 3.x
- but you will want access to the packages
  - something like macports
  - http://www.macports.org
  - allows you to install any python package
  - sudo port install <some package>

python

- hello world!
- from a script and from an interpreter

python

- many different coding “styles”
- “best” styles get the distinction of “pythonic”
  - ill formed definition
  - changes as the language matures
- pythonic code is:
  - simple and readable
  - uses dynamic typing when possible
- …or to quote Tim Peters…

python zen

type this

get this

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass
Unless explicitly silenc                                    ss.
In the face of ambiguity                                    s way to do it.
There should be one-- ar                                    ou're Dutch.
Although that way may no
Now is better than never.
Although never is often better tha  right now.
If the implementation is hard to e plain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

python is quirky
but, don't assume that means
it is not a **serious** tool

syntax

- numbers
  - int or float
- complex numbers

```
>>> 7*5
35
>>> 5/7
0
>>> 7/5
1
>>> 7.0/5
1.4
```

```
>>> tmpVar = 4
>>> print tmpVar
4
>>> tmpVar/8
0
>>> tmpVar/8.0
0.5
```

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

syntax

- 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

- tuples
- lists
  - highly versatile and mutable
  - containers for anything

```
>>> 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
ThatGuyInTheBackThatGuyInTheBackThatGuyInTheBackThatGuyInTheBack
168
ended for loop without break
```

syntax loops

- 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 can import queues
  - append(value)
  - pop_left(), deque first element
- 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

```
>>> 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}
print 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 str(element) + " is at index " + str(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

- 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…"
```

```
Everybody is a five!
```

```
a=327676
b=a

if a is b:
    print "These are the same object!"
else:
    print "Wish we had the same objects..."
```

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

```
a=327676
b=327675+1

if a is b:
    print "These are the same object!"
else:
    print "Wish we had the same objects..."
```

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

small integers are cached
strings behave the same

```
a=5
b=4+1

if a is b:
    print "Everybody is a five!"
else:
    print "Wish we had fives..."
```

```
Everybody is a five!
```

conditionals

```
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

- 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')
```

```
[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
```

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

print get_square_and_tenth_power(2)
```

```
(4, 1024)
```

classes

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

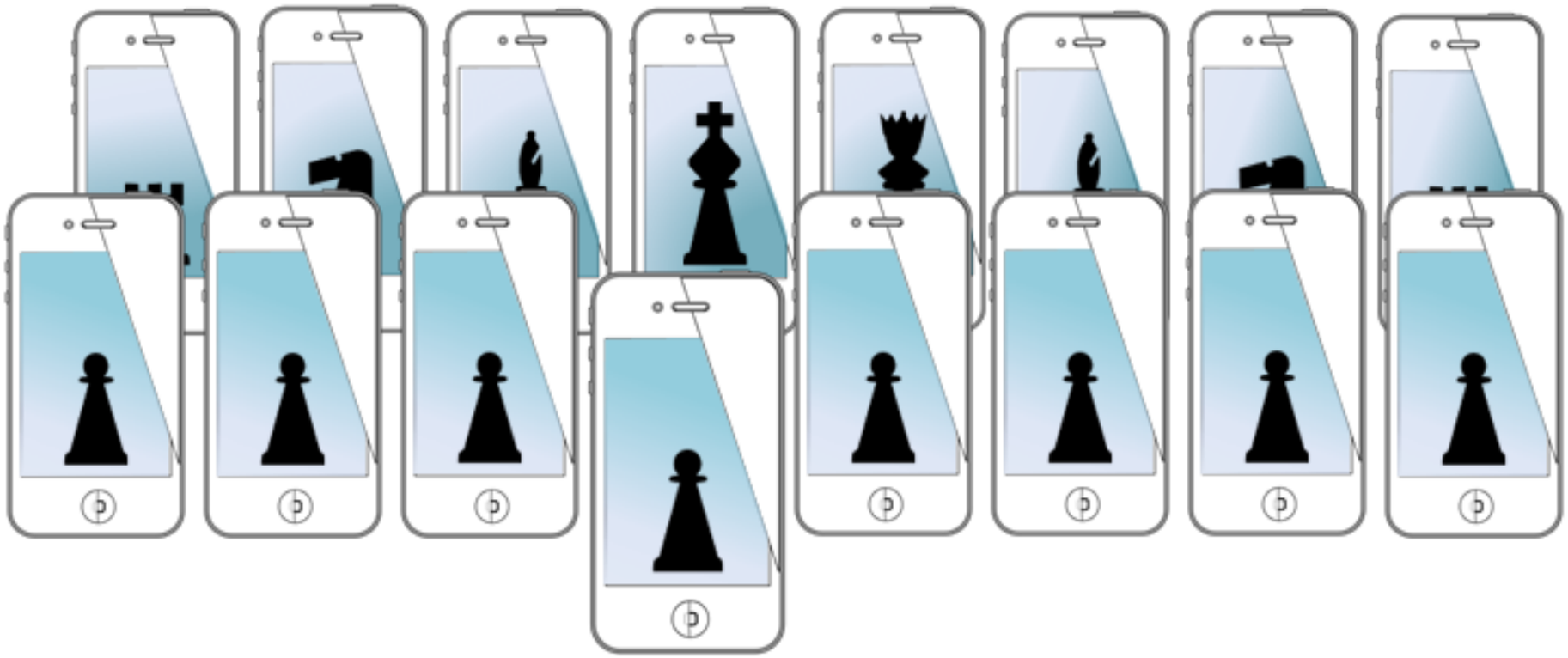
```
class BodyPart(object):
    def __init__(self,name):
        self.name = name;


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

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


myHeart = Heart(1,"second")
myHeart.print_rate()
```

for next time…

- more python examples
- tornado
- pymongo
- in preparation for
  - proper http requests in iOS
  - numpy
  - scikit-learn

install these on your mac!

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture seventeen: python crash-course

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