Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 1 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 1 - II. an introduction to python

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

python

- Guido van Rossum

From wikipedia:

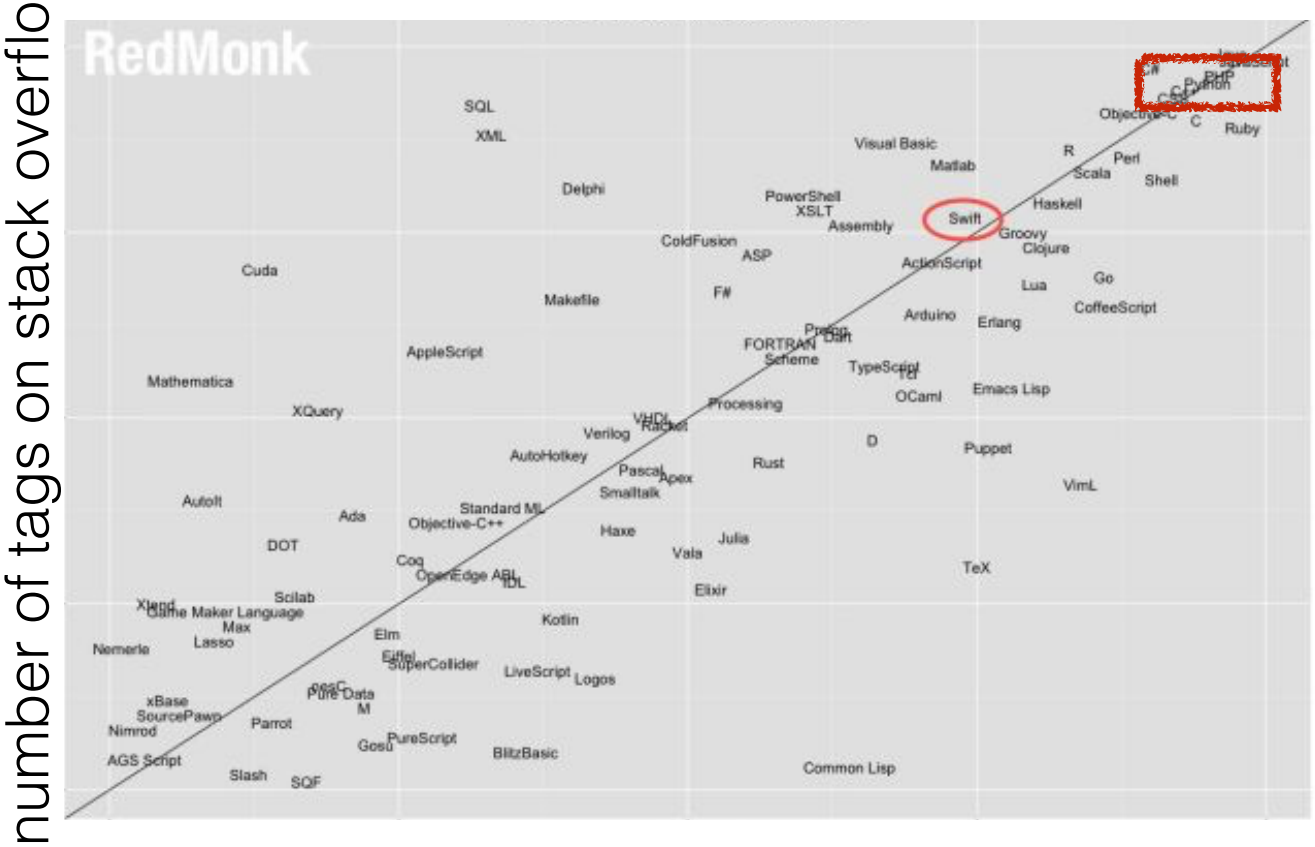
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language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

- appears in every programming top-ten list
  - hacker news
  - dice job list
  - book sales

you should know it!



python disclaimers

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

python releases

- 1.0 (up to 1.6)
  - basic python, complex numbers, lambdas
- 2.0 (still used, but it's the beginning of the end)
  - unified types, made completely object-oriented
- 3.0 (still actively developed)
  - eliminate multiple paradigms (kinda)
  - 2.x not necessarily compatible with 3.x

installation

- on **mac** or **linux**:
  - open a terminal
  - do nothing
    - OS X and linux ship with python
  - …but you probably want to install python 3
- on **windows**, **mac**, or **linux**
  - go to https://www.python.org and get python 3 on your system
- but you will want access to the **packages**
  - something like anaconda or pip
  - allows you to install most any python package that is registered

hello world

- from interpreter
- from script
- from jupyter

exercise

- install python 3 on your machine
  - try using anaconda first!
- run “hello world” examples

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

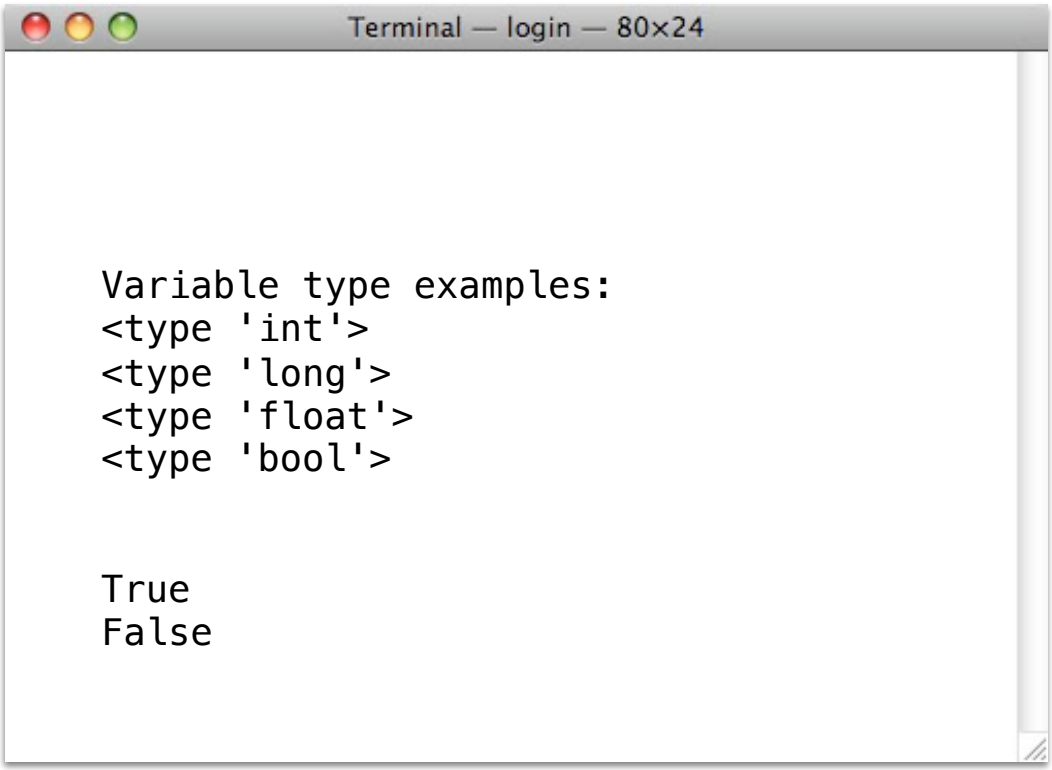
Unit 1 - III. python basics

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

comments, variables, types

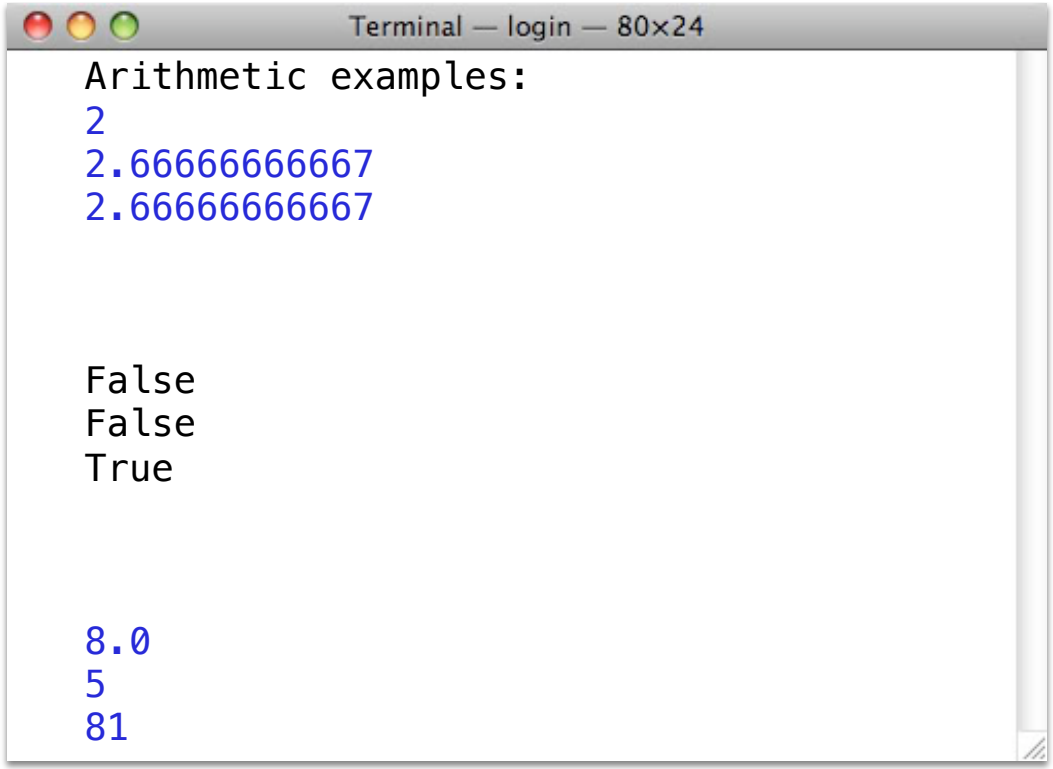
```
# this is a comment
# place this at the top of python file to enable running as >>./filename.py
! /usr/bin/python
# otherwise you can run with >>python filename.py
# this command can be run from a terminal/cmd window
```

```
int_val = 8
long_val = 23423423235L
float_val = 2.0
bool_val = True

print "Variable type examples:"
print type(int_val)
print type(long_val)
print type(float_val)
print type(bool_val)

# testing for the type of a variable
print isinstance(float_val,float)
print isinstance(float_val,int)
```



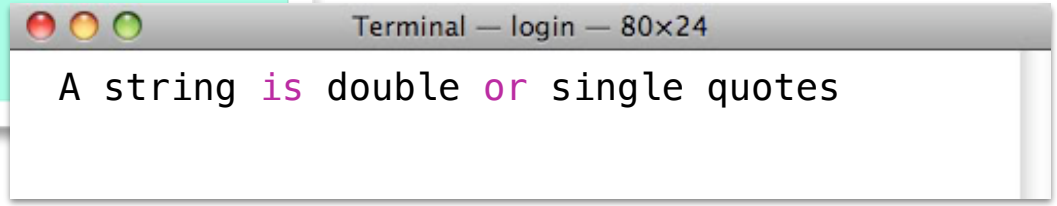
arithmetic and casting

```
print "\nArithmetic examples:"
print 8 / 3
print float(8) / 3
print float(8) / float(3)



print True and False  # logicals
print 8 == 3   # logical equallty
print 5 <= 6   # logical comparison



print 2.0*4.0  # multiplication
print 65%6    # remainder, modulus
print 3**4    # 3 to the fourth power
```

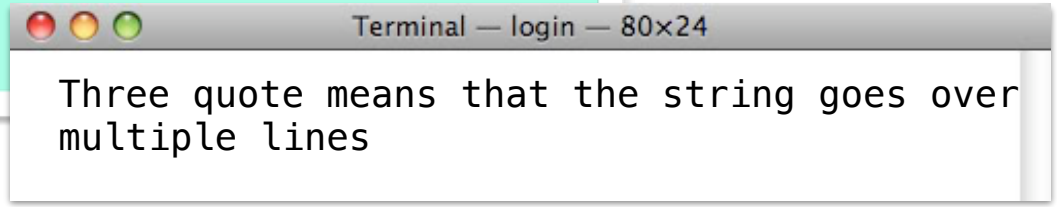


strings and string operations

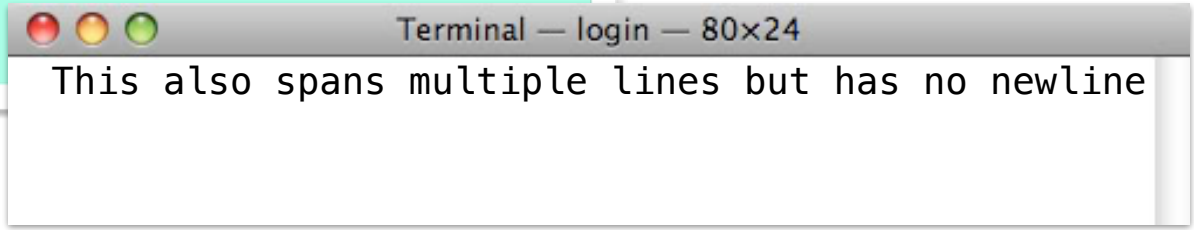
```
str_val = "A string is double or single quotes”
print str_val
```




```
str_val_long = '''Three quote means that the string goes over
    multiple lines’''
print str_val_long
```

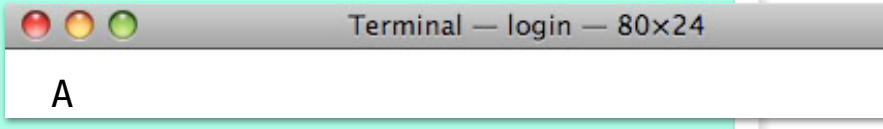


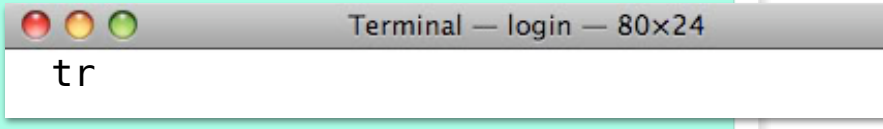

```
str_val_no_newline = '''This also spans multiple lines \
    but has no newline'''
print str_val_no_newline
```

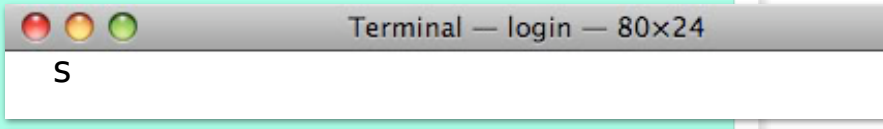


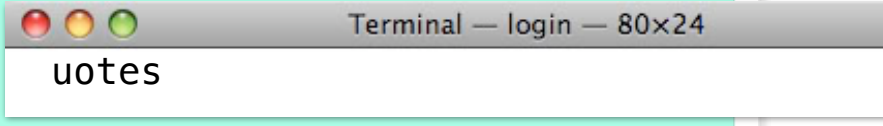
strings and string operations

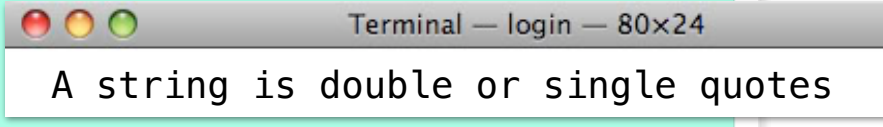
```
# string can be accessed in a variety of
different ways
print str_val[0] # initial element "0th" element


print str_val[3:5] # elements 3 and 4, but not 5


print str_val[-1] # the last element in the
string


print str_val[-5:] # the last five elements


print str_val[0:5] + str_val[5:] # print the
first five elements, then from the fifth and on


str_val[5] = 'G' # this is an error, strings are
immutable once they are set
```

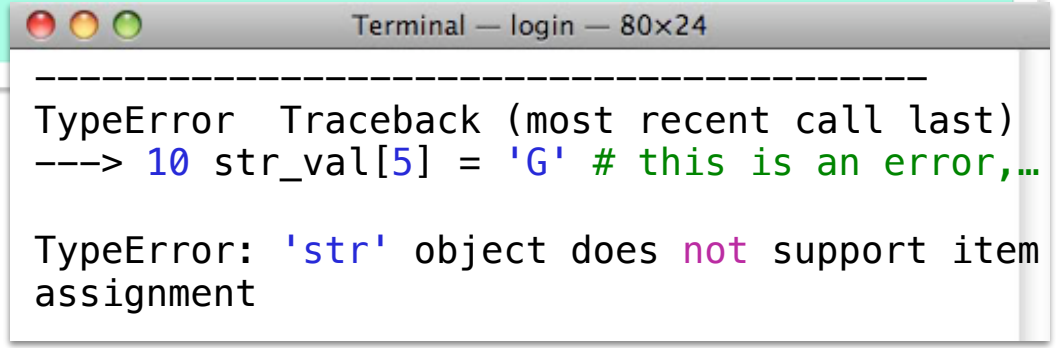






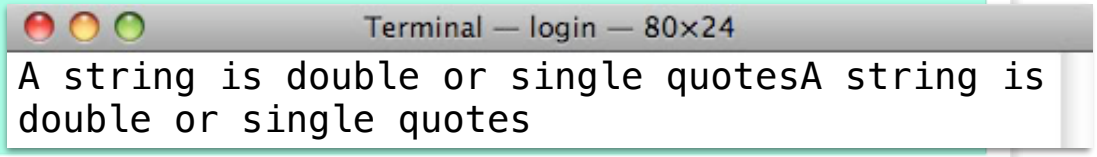




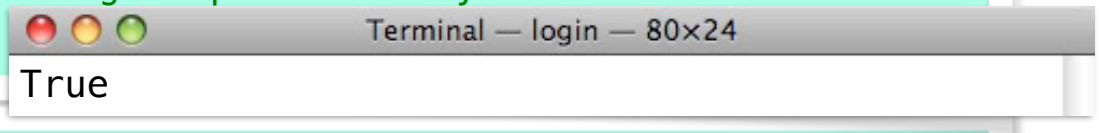


strings and string operations

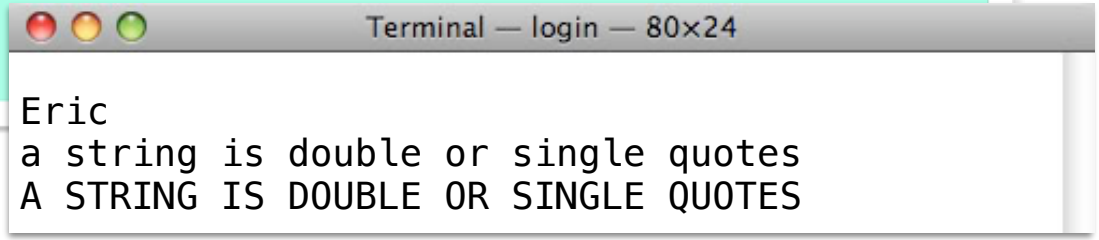
```
# some common operations for strings
print str_val*2 # mutliply is like adding many times, here it repeats the string
```




```
print 'Python' > 'Java' # compare the strings alphabetically
```




```
print "eric".capitalize() # the dot operator works like most other OOP languages
print str_val.lower()
print str_val.upper()
```



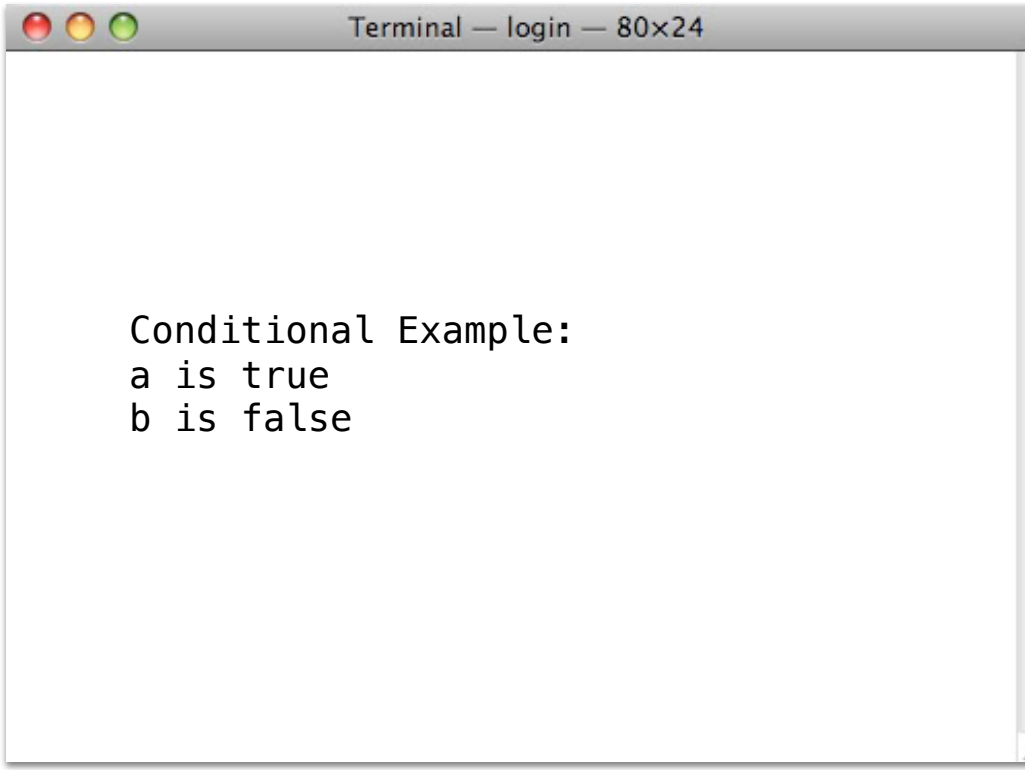

```
print "this, is, separated, by, commas".split(',') # this results is returned as a
list, which we need to talk about!
```

calculator example 1

- build the interpreter

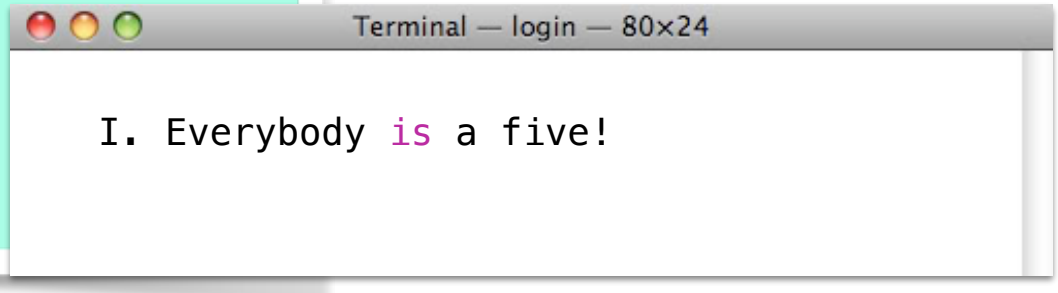
conditionals

```
# conditional example
print "\nConditional Example:"

a, b = True, False

if a:
    print "a is true"
elif a or b:
    print "b is true"
else:
    print "neither a or b are true"

# conditional assignment
val = "b is true" if b else "b is false"
print val
```

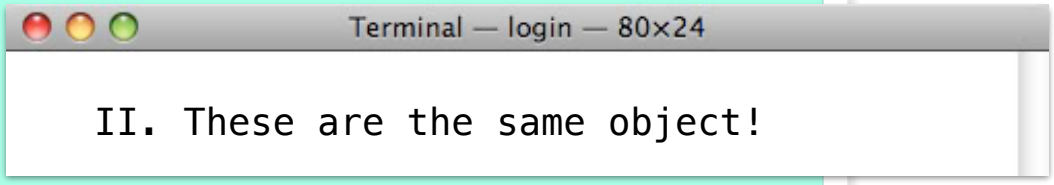


conditionals

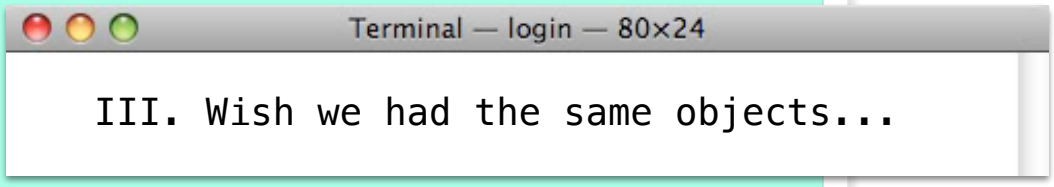
```
# I. the traditional == works as expected
a=5
b=5
if a==b:
    print "I. Everybody is a five!"
else:
    print "I. Wish we had fives…"
```




```
# II. the "is" function is for object comparison, much like comparing pointers
a=327676
b=a
if a is b:
    print "II. These are the same object!"
else:
    print "II. Wish we had the same objects..."
```

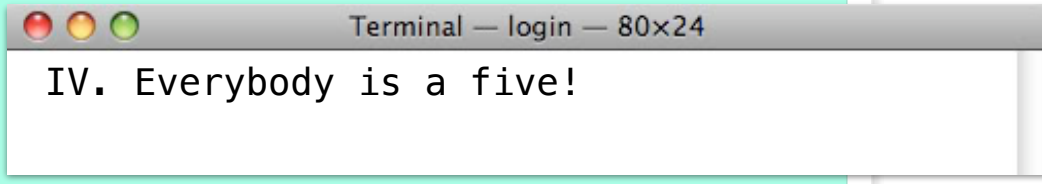



```
# III. while these have the same value, they are not the same memory
a=327676
b=327675+1
if a is b:
    print "III. These are the same object!"
else:
    print "III. Wish we had the same objects..."
```

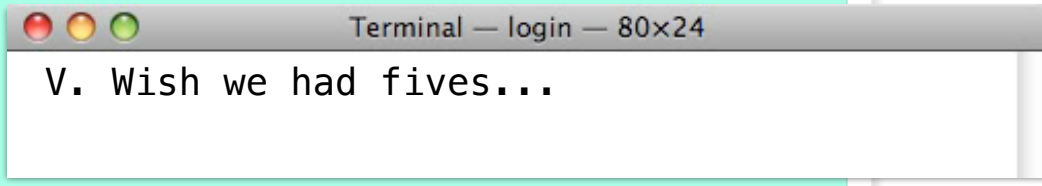


conditionals

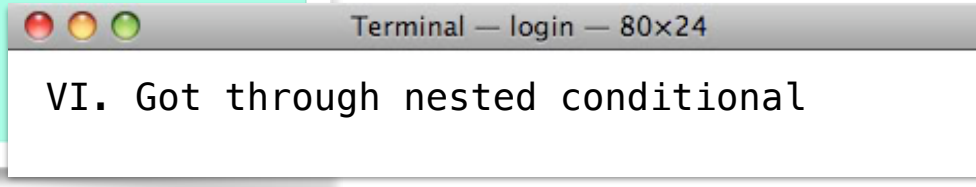
```
# IV. you would expect this to say wish we had fives,
# but small integers like this are cached so right now they do point to the same
memory
a=5
b=4+1
if a is b:
    print "IV. Everybody is a five!"
else:
    print "IV. Wish we had fives..."
```




```
# V. but if we change the memory, that caching gets released
b = b*2.0
b = b/2.0
if a is b:
    print "V. Everybody is a five!"
else:
    print "V. Wish we had fives..."
```



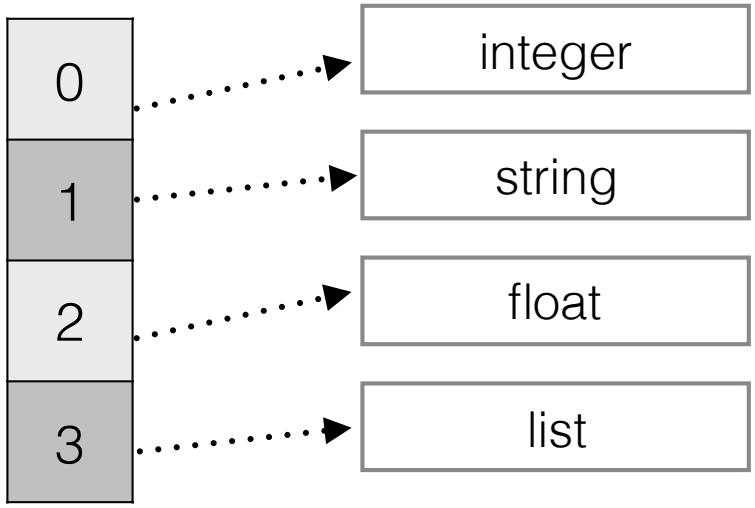

```
# you can also perform nested conditionals, like bounding
if 5 < 8 < 6: # not true because 8 is not less than 6
    print 'VI. How did we get here'
elif 4 < 18 < 22:
    print "VI. Got through nested conditional"
```



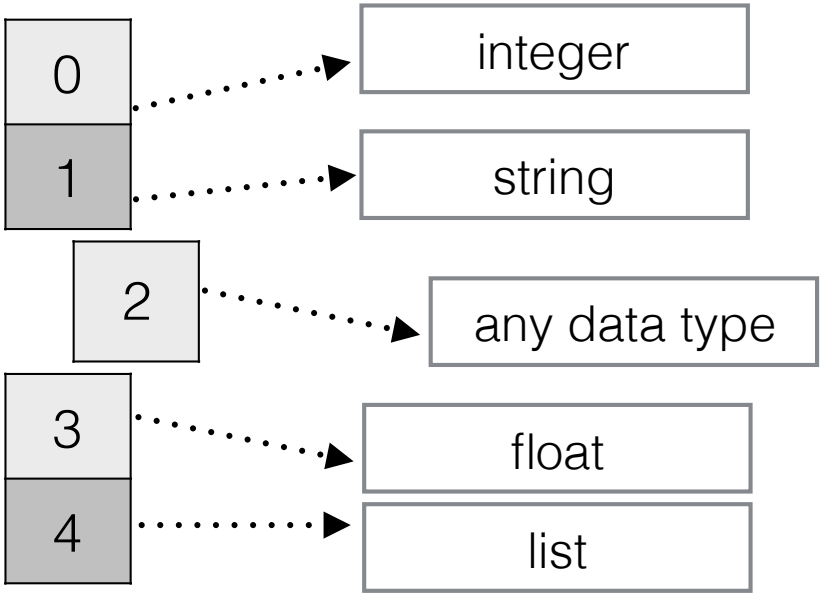
calculator example 2

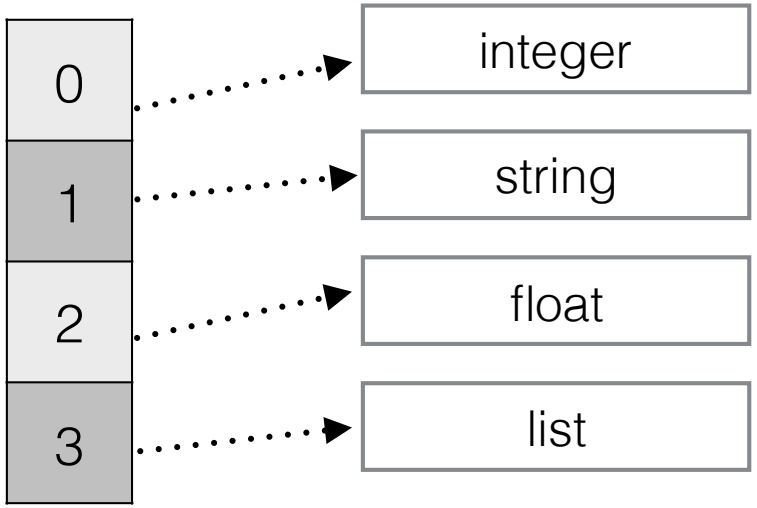
- build conditionals statements

python lists and tuples



a list can be changed

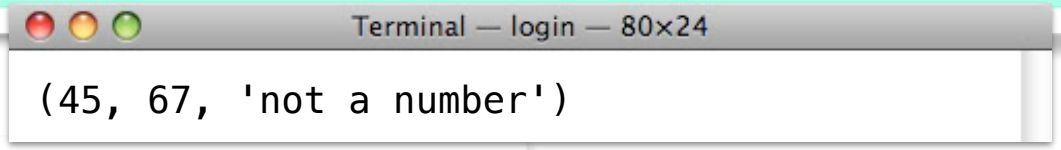




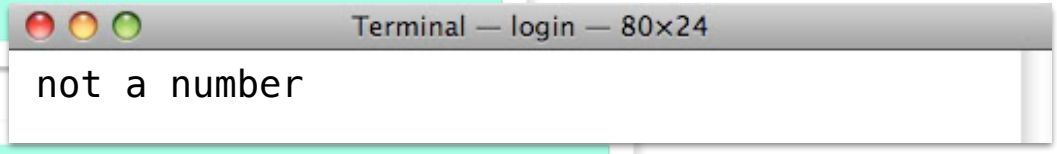
a tuple is immutable

python tuples

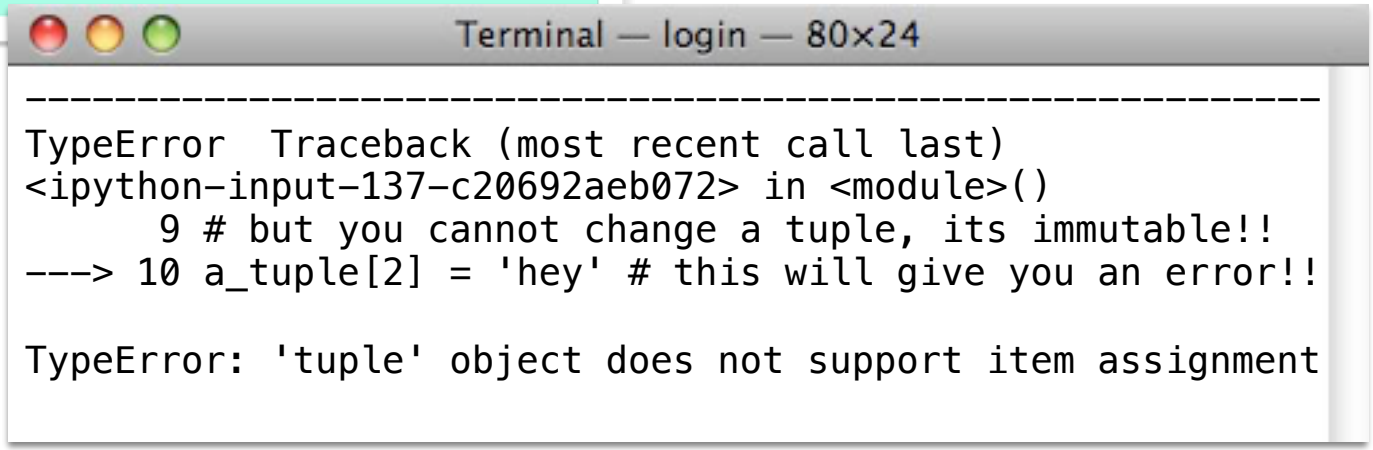
```
# tuples are immutable lists and are designated by commas
# you can store ANYTHING inside a tuple, its basically a complex object container
a_tuple = 45, 67, "not a number"
print a_tuple
```




```
# you can access a tuple with square brackets
print a_tuple[2]
```




```
# but you cannot change a tuple, it's immutable!!
a_tuple[2] = 'hey' # this will give you an error!!
```



python lists

```
# A List is one of the most powerful tools in python from which
# most abstract types get created and implemented
# a list is very much like the mutable version of a tuple
# it can hold any type of information
a_list = [45, 67, "not a number”]

# we can add to a list through the append function
a_list.append("A string, appended as a new element in the list")
print a_list
```

```
[45, 67, 'not a number', 'A string, appended as a new element in the list']
```

```
# Lists can have other lists in them
tmp_list = ["a list", "within another list", 442]
a_list.append(tmp_list)
print a_list
```

```
[45, 67, 'not a number', 'A string, appended as a new element in the list',
['a list', 'within another list', 442]]
```

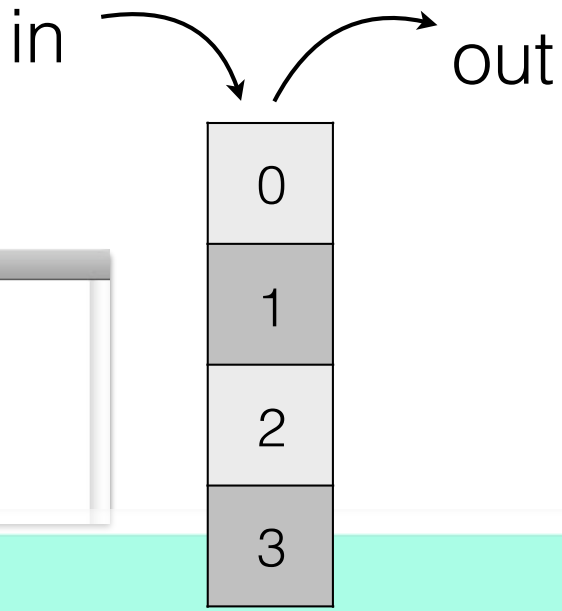
```
# all of the indexing we learned from before still works with lists
print a_list[-1]
print a_list[-2:]
```

```
['a list', 'within another list', 442]
['A string, appended as a new element in the list', ['a list', 'within another
list', 442]]
```

stacks and queues

```
# list as a stack
print "\nStack Example:"
list_example = []
list_example.append('LIFO')

for i in range(0, 5):
    list_example.append(i)

print list_example
val = list_example.pop()
print val
print list_example
```

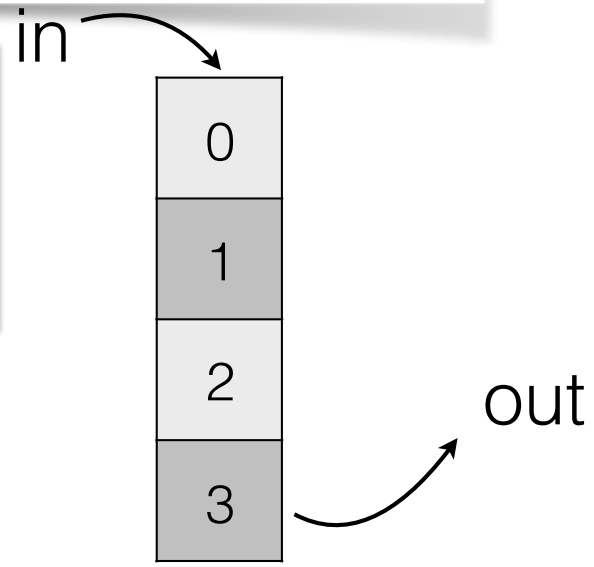
```
Terminal — login — 80×24
Stack Example:
['LIFO', 0, 1, 2, 3, 4]
4
['LIFO', 0, 1, 2, 3]
```




```
# list as a queue
print "\nQueue Example:"
from collections import deque # this is an import, we will get back to that later
```

```
q_example = deque()
q_example.appendleft("FIFO")
for i in range(5, 10):
    q_example.appendleft(i)

print q_example
val = q_example.pop()
print val
print q_example
```

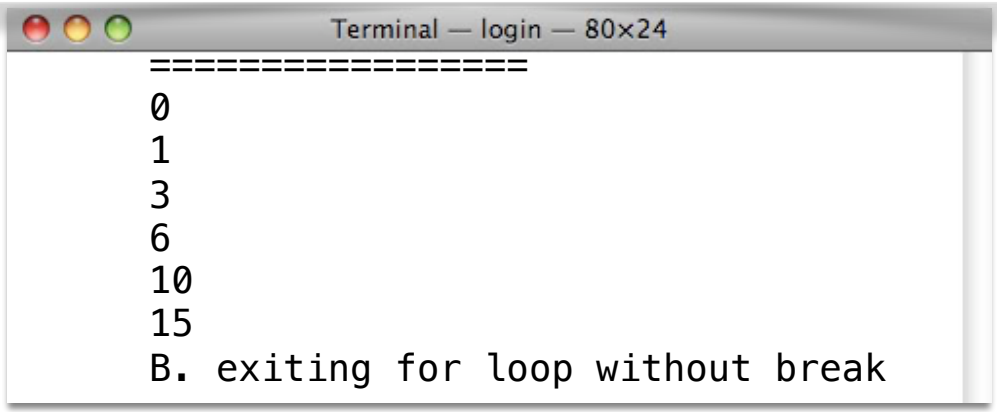
```
Terminal — login — 80×24
Queue Example:
deque([9, 8, 7, 6, 5, 'FIFO'])
FIFO
deque([9, 8, 7, 6, 5])
```

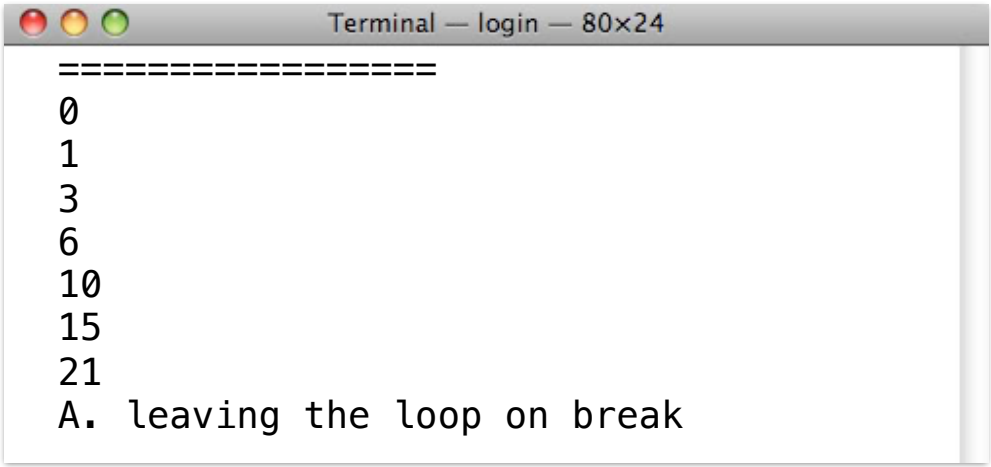


python loops

```
import random
print '================='
val = 0
for i in range(0, random.randint(1, 10) ):
    val += i
    print val
    if val>20:
        print ' A. leaving the loop on break'
        break # break out of loop
else: # this else belongs to the for loop
    print 'B. exiting for loop without break'
```

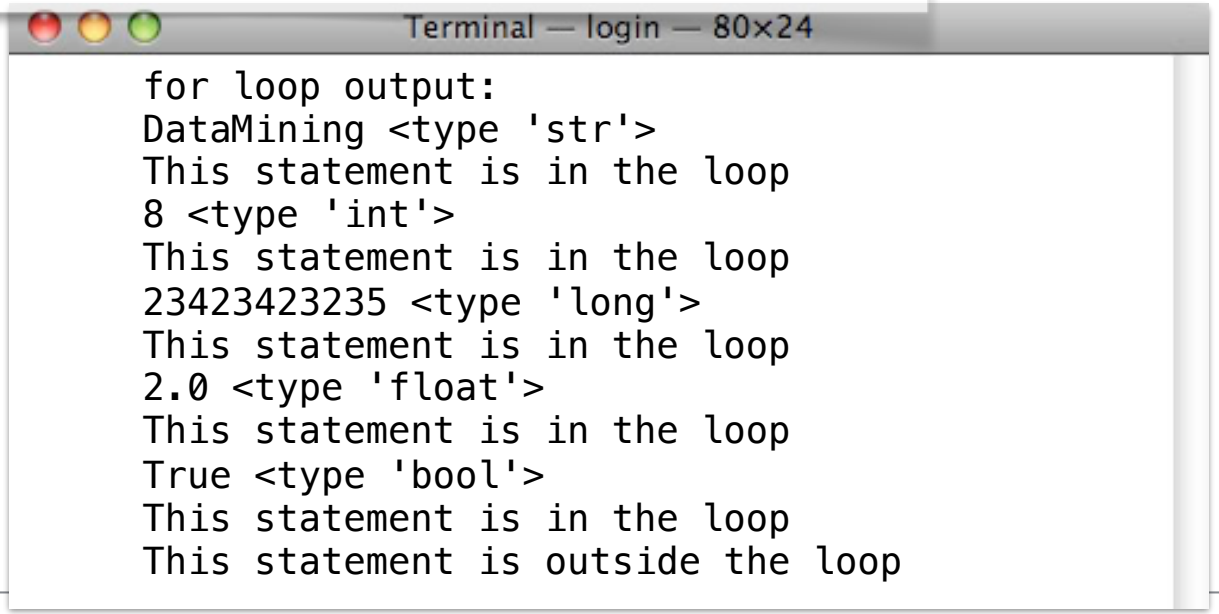
0, 1, 2, 3, 4, 5, 6, 7, 8, 9





python loops

```
#=========================================
# for loop example with list
print "\nfor loop output:"

list_example = [int_val, long_val, float_val, bool_val]
list_example.insert(0, "DataMining")

# notice that the loop ends with a colon and
# is designated by the tab alignment
for val in list_example:
    print str(val) + ' ' + str(type(val))
    print "This statement is in the loop"

print "This statement is outside the loop"
```



python loops with lists

```
# you can also get the index using the enumerate example
for index, val in enumerate(list_example):
    print str(val), '\t is at index \t', index
```

Terminal — login — 80×24

```
DataMining      is at index  0
8               is at index  1
23423423235     is at index  2
2.0            is at index   3
True           is at index   4
```

```
# this is a classic example for zipping, provided by the official python tutorial
# notice the references to Monty Python

# say you have two lists of equal size that you would like to
# loop through without indexing, you can use the "zip" function
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print 'What is your %s?  It is %s.' % (q, a)
```

Terminal — login — 80×24

```
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

building a functional calculator

- using reverse polish notation
- stacks
- user input

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 2 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 2 - II. more python basics

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

pythonic

- 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

```
>>> 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
In the face of ambiguity                                    ss.
There should be one-- ar                                    s way to do it.
Although that way may no                                    ou're Dutch.
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!
```

type this

get this

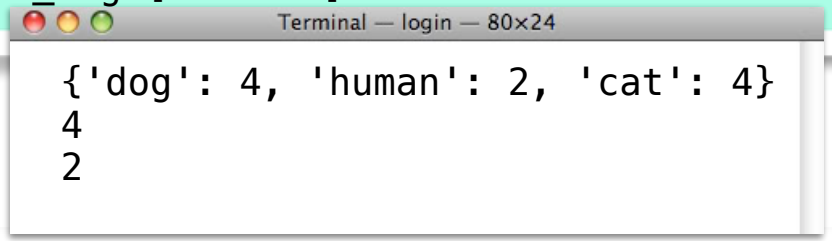
python is quirky
but, don't assume that means
it is not a **serious** tool

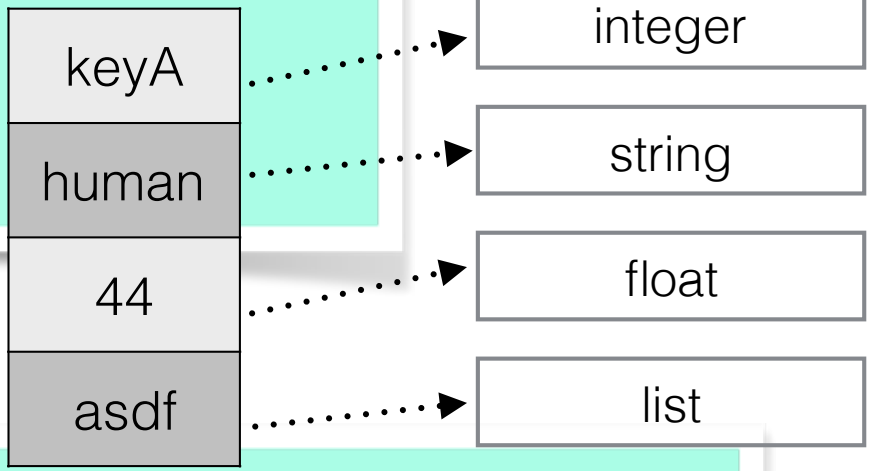
pythonic conventions

- too many to go over
  - check out: http://legacy.python.org/dev/peps/pep-0008/
- name variables with underscore: `this_is_my_variable`
- use indentation to increase clarity
- one space before and after = i.e., `x = 5`
- space mathematics well `hypot2 = x*x + y*y`

more pythonic: dictionaries

```
# Dictionaries map keys to values.
# Here we set up a key as a string and the value as a number
num_legs = { 'dog': 4, 'cat': 4, 'human': 2 }

# You access Subscripts via the "key"
print num_legs
print num_legs['dog']
print num_legs['human']
```






```
# Entries can be added, updated, or deleted.
# Again, these are just containers for any memory type
num_legs['human'] = 'two'
num_legs['bird'] = 'two and some wings'
num_legs[45] = 'a key that is not a string' # notice that key is not a string
# the key just needs to be some immutable memory

del num_legs['cat']
print num_legs
```

```
{'bird': 'two and some wings', 45: 'a key that is not a string', 'dog': 4,
'human': 'two'}
```

more pythonic: sets

```
# Sets, taken from the Python sets tutorial
# https://docs.python.org/2/tutorial/datastructures.html#sets
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
fruit = set(basket) # create a set without duplicates
print fruit
print 'orange' in fruit # fast membership testing
print 'crabgrass' in fruit
```

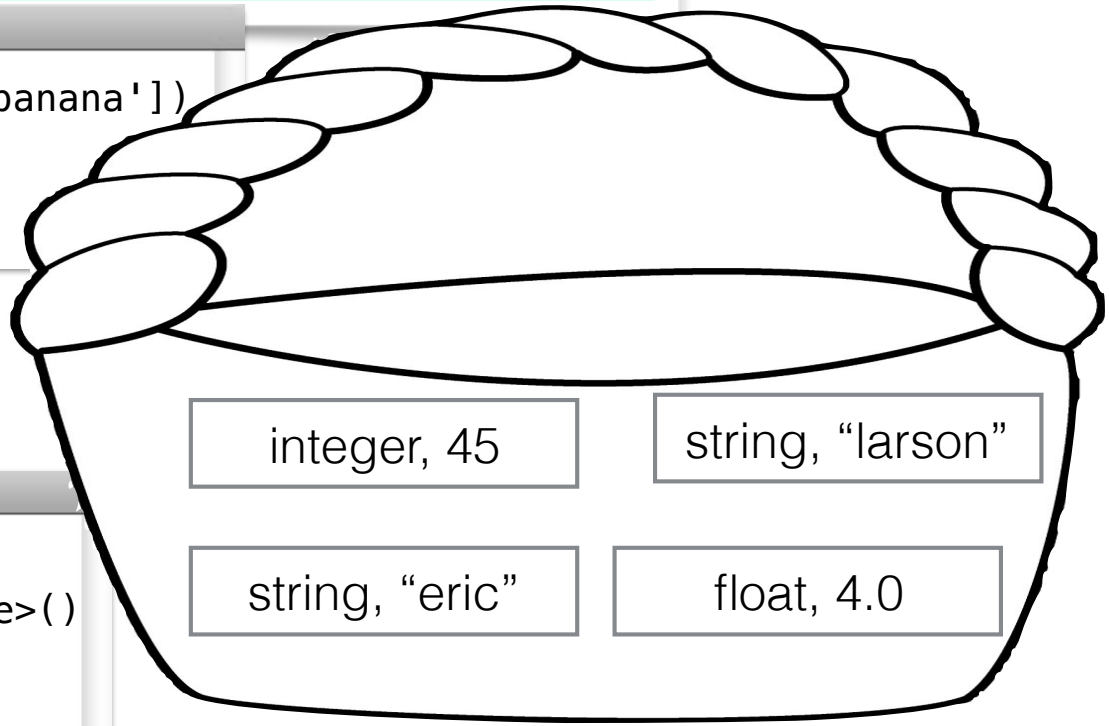
Terminal — login — 80×24

```
set(['orange', 'pear', 'apple', 'banana'])
True
False
```

```
a = set((45,'eric',4.0,[5,6]))
```

Terminal — login — 80×24

```
TypeError                 Traceback
<ipython-input-5-5130bdc6e4fb> in <module>()
----> 1 a = set((45,'eric',4.0,[5,6]))

TypeError: unhashable type: 'list'
```

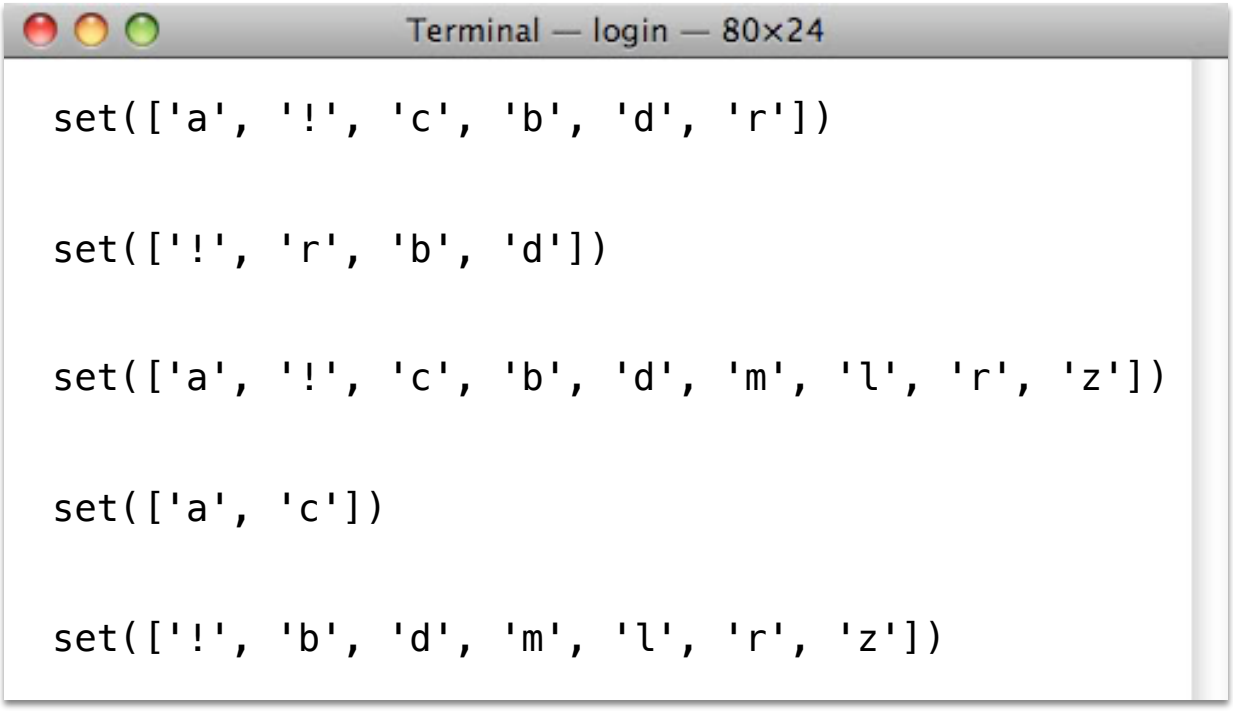


a basket of **hashable** data

more pythonic: sets

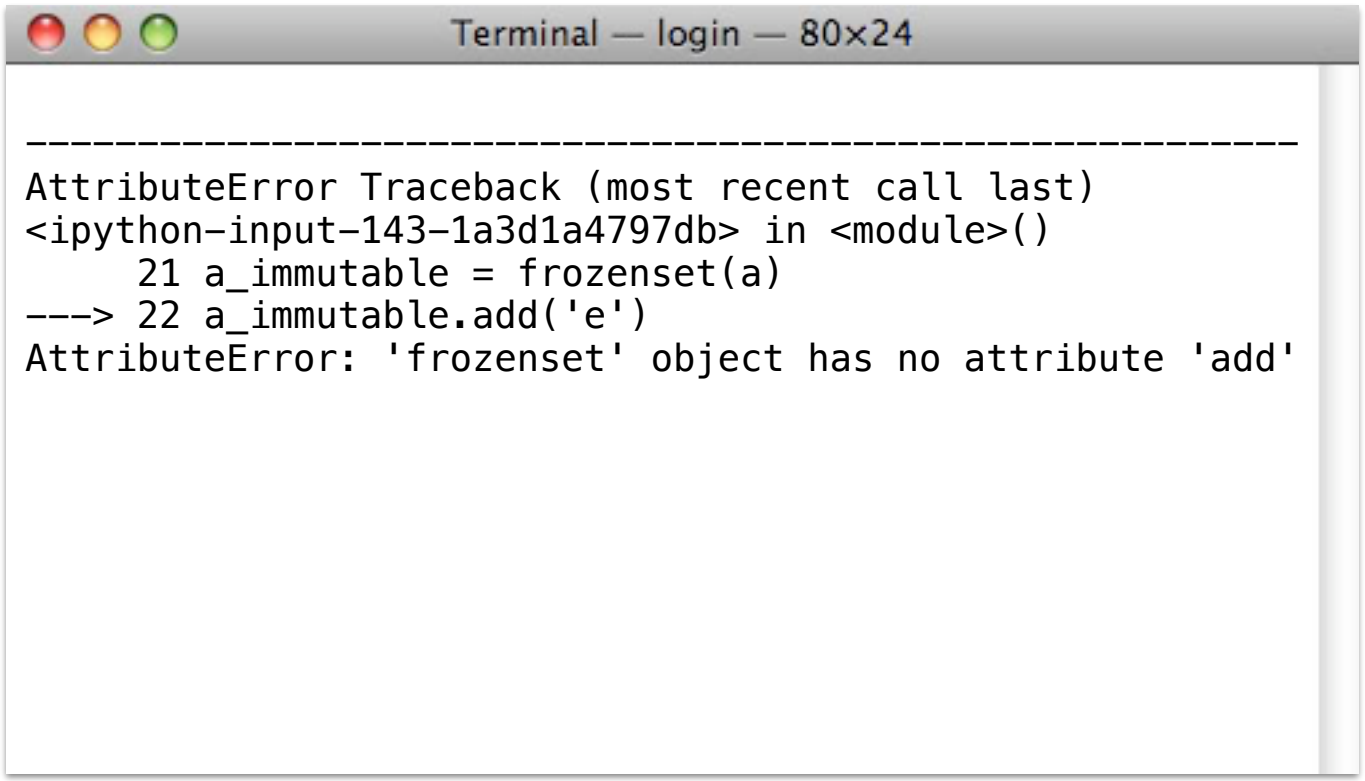
```
# Demonstrate set operations on unique letters from two words
a = set('abracadabra')
b = set('alacazam')
a.add('!') # also add the some punctuation
```

```
# set operations
print a   # unique letters

print a - b # in a but not in b

print a | b # either a or b

print a & b # both a and b

print a ^ b # a or b, not both
```



more pythonic: immutable sets

```
a_immutable = frozenset(a)
a_immutable.add('e')   # the set is immutable, so we cannot add to it, this will
give an error!
```



more pythonic: functions

```
# more functions examples
def show_data(data):
   print data

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

Terminal — login — 80×24

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

```
# you can also define default values for the functions
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')
```

default value

Terminal — login — 80×24

```
[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
```

```
# as well as have multiple return types in the function
def get_square_and_tenth_power(x):
    return x**2,x**10

print get_square_and_tenth_power(2)
```

return a tuple

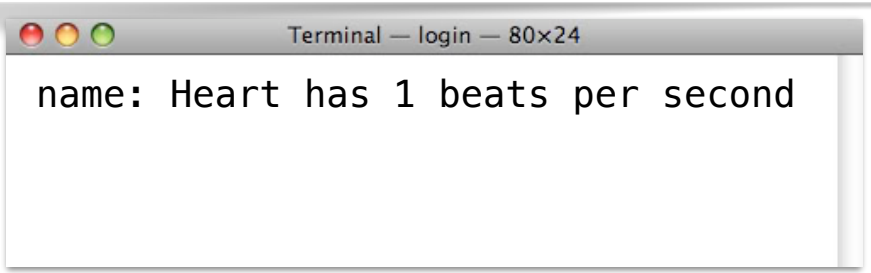
```
(4, 1024)
```

calculator example 3

- making the code more pythonic

classes

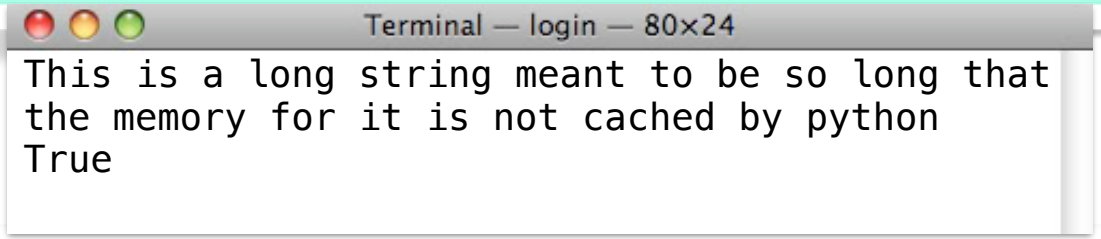
```
# This is a class that inherits from a generic object
class BodyPart(object):
    # this is a class variable, shared across all instances
    def __init__(self,name):
        self.name = name # the name attribute is unique to each instance of the class

# now define a class that sub classes from the defined BodyPart CLass
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


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



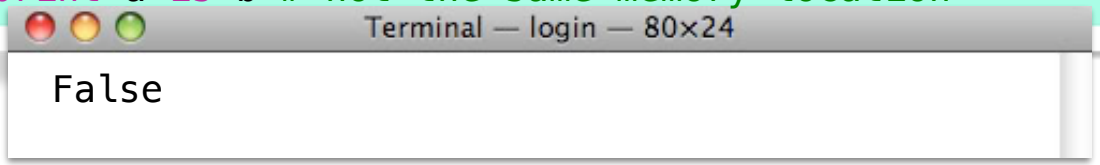
classes

```
# This is a class that inherits from a generic object
class BodyPart(object):
    kind = "This is a long string meant to be so long that the memory for it is not
            cached by python"

    # this is a class variable, shared across all instances
    def __init__(self,name):
        self.name = name # the name attribute is unique to each instance of the class

my_heart = Heart(1,"second")
generic_part = BodyPart("Foot")
print my_heart.kind
print my_heart.kind is generic_part.kind # true, these are the same memory location
```

class shared variable

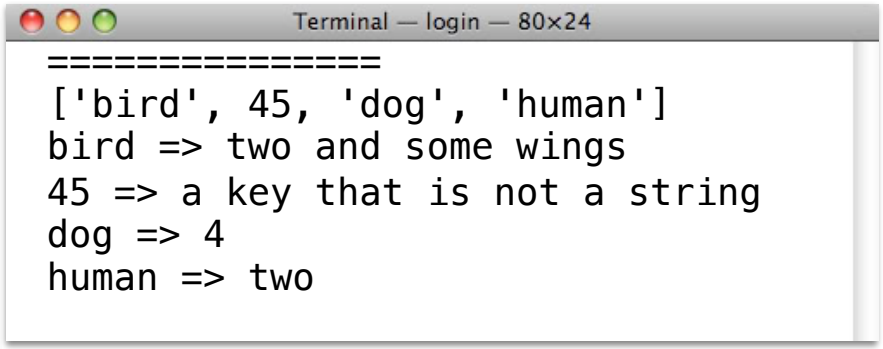


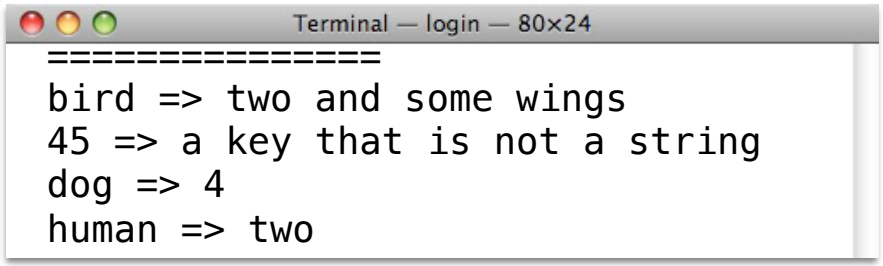

```
# take the following for example, these are not the same object
a = "This is a long string meant to be so long that the memory for it is not cached by
python"
b = "This is a long string meant to be so long that the memory for it is not cached by
python"
print a is b # not the same memory location
```

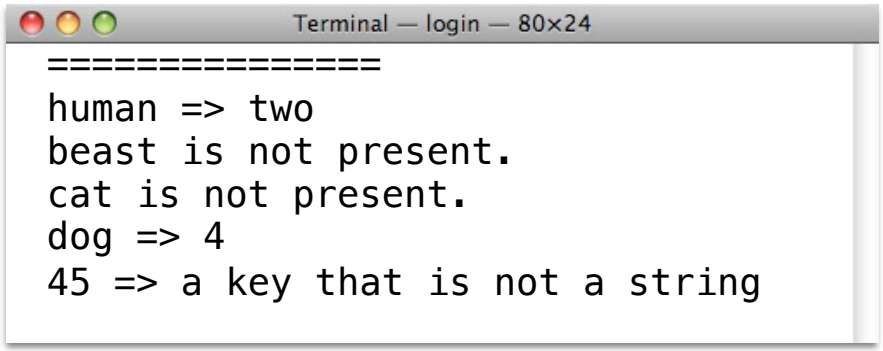


python loops with dictionaries

```
# Looping through dictionaries
print '==============='
# Get all the keys.
print num_legs.keys()
for k in num_legs.keys():
    print k, "=>", num_legs[k]

print '==============='
# you can also use the iter_items function
for k, v in num_legs.items():
    print k, "=>", v

print '==============='
# Test for presence of a key.
for t in [ 'human', 'beast', 'cat', 'dog', 45 ]:
    print t,
    if t in num_legs:
        print '=>', num_legs[t]
    else:
        print 'is not present.'
```







comprehensions

```
# imagine we want to take every element in a range to the fourth power
times_four = [x**4 for x in range(10)]
print times_four
```

Terminal — login — 80×24

```
[0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561]
```

```
# you can also call functions inside a comprehension
questions = ['name', 'quest', 'favorite color']
quest_upper = [x.upper() for x in questions]
print quest_upper
```

Terminal — login — 80×24

```
['NAME', 'QUEST', 'FAVORITE COLOR']
```

```
# you can also do comprehensions with dictionaries
times_four = {x:x**4 for x in range(10)}
# notice curly braces and key placement
print times_four
```

Terminal — login — 80×24

```
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256,
5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

```
# Finally, all of the enumerate, zipping, and slicing we performed also applies to
# list comprehensions
x_array = [10, 20, 30]
y_array = [7, 5, 3]
# this prints the sum of the multiplication of the arrays
print sum(x*y for x,y in zip(x_array, y_array))
```

Terminal — login — 80×

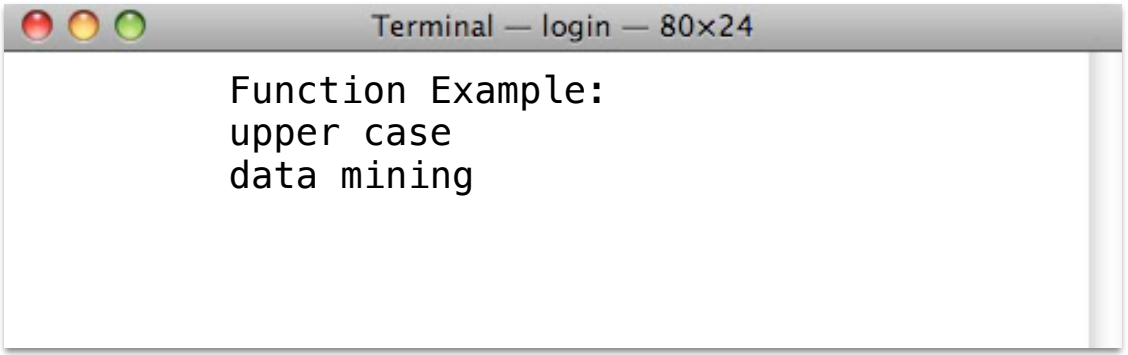
```
260
```

calculator example 4

- making the code more object-oriented
- separating model and view controller

exceptions

```
# create and call a function
# the function can be defined almost anywhere in file, as long as it is defined
before it gets used
def make_strings_lowercase(str_input):
    assert isinstance(str_input, str)  # test the type of input
    return str_input.lower()

# now we are back on the main execution
print make_strings_lowercase("UPPER CASE")
print make_strings_lowercase("Data Mining")
```



exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

```
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```

Terminal — login — 80×24

```
0 2
Division by zero.
```

Terminal — login — 80×24

```
6 5
Some value is out of range: list
index out of range
```

Terminal — login — 80×24

```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

Terminal — login — 80×24

```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

```
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```

Terminal — login — 80×24

```
0 2
Division by zero.
```

Terminal — login — 80×24

```
6 5
Some value is out of range: list
index out of range
```

Terminal — login — 80×24

```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

Terminal — login — 80×24

```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

```
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```

Terminal — login — 80×24

```
0 2
Division by zero.
```

Terminal — login — 80×24

```
6 5
Some value is out of range: list
index out of range
```

Terminal — login — 80×24

```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

Terminal — login — 80×24

```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html

import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```

Terminal — login — 80×24

```
0 2
Division by zero.
```

Terminal — login — 80×24

```
6 5
Some value is out of range: list
index out of range
```

Terminal — login — 80×24

```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

Terminal — login — 80×24

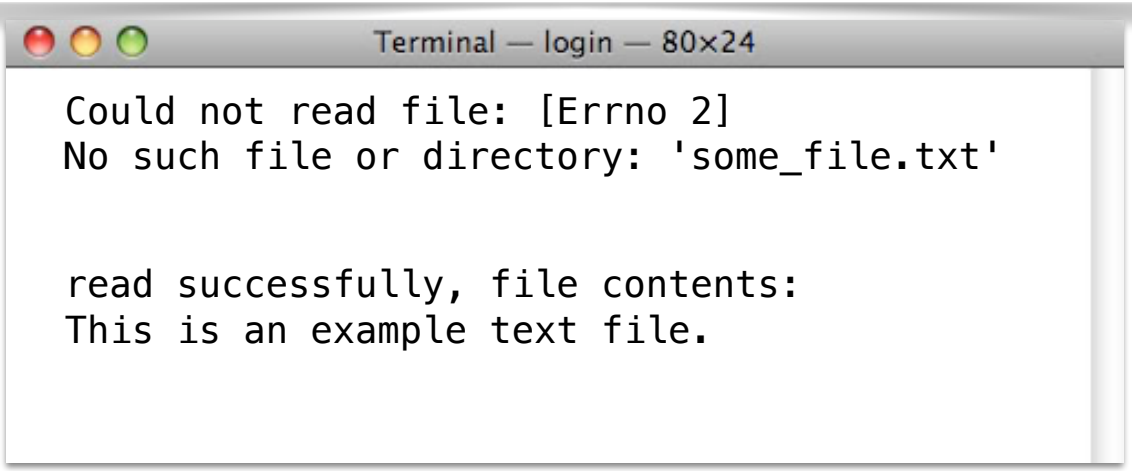
```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

calculator example 5

- add exception handling to class
- adding inheritance for better error handling

opening a file

```
# the regular way of opening a file, lots of error checking
try:
    file = open("some_file.txt")
    data = file.read()
except IOError, detail:
    print "\nCould not read file:", detail
else:
    print "Read successfully, file contents:"
    print data
finally:
    # this always gets called, close the file if it's open
    if not file.closed:
        file.close()
```



we can get rid of this through OOP!

opening a file using “with”

```
with open("some_file.txt") as file:
    data = file.read()
    print "Read successfully, file contents:"
    print data


# is the file closed? Let's check
print file.closed
```

Terminal — login — 80×24

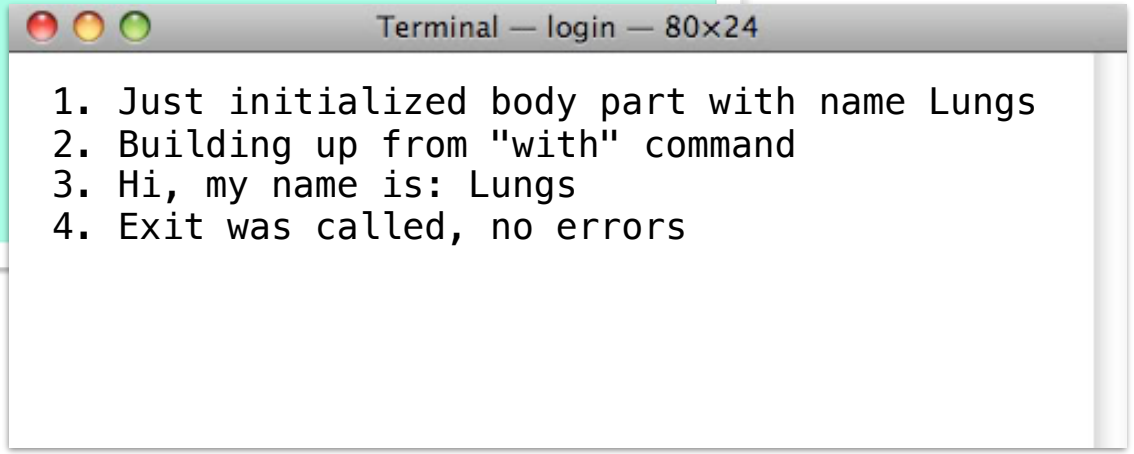
```
Read successfully, file contents:
This is an example text file.
```

Terminal — login — 80×24

```
True
```

writing a class to use “with”

```
class BodyPart(object):
    def __init__(self,name):
        self.name = name
        print '1. Just initialized body part with name', name

    def __enter__(self):
        print '2. Building up from "with" command'
        return self

    def __exit__(self, type, value, traceback):
        if value is not None:
            print '4. An error occurred,', value
        else:
            print '4. Exit was called, no errors'

def print_self(self):
    # 5/0 # uncomment to raise an error
    print '3. Hi, my name is:', self.name


with BodyPart("Lungs") as bp:
    bp.print_self()
```



writing a class to use “with”

```
class BodyPart(object):
    def __init__(self,name):
        self.name = name
        print '1. Just initialized body part with name', name

    def __enter__(self):
        print '2. Building up from "with" command'
        return self

    def __exit__(self, type, value, traceback):
        if value is not None:
            print '4. An error occurred,', value
        else:
            print '4. Exit was called, no errors'

def print_self(self):
    5/0 # uncomment to raise an error
    print '3. Hi, my name is:', self.name


with BodyPart("Lungs") as bp:
    bp.print_self()
```

divide by zero!

```
Terminal — login — 80×24
1. Just initialized body part with name Lungs
2. Building up from "with" command
4. An error occurred, integer division or modulo by zero
---------------------------------------
ZeroDivisionError                Traceback

<ipython-input-23-15b552589a5a> in print_self(self)
---> 19         5/0 # uncomment to raise an error

ZeroDivisionError: integer division or modulo by zero
```

calculator example 6

- adding custom functions from a user file
- possible exercises to extend your python programming knowledge

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 3 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 3 - II. working with the scripting environment

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

python environments

- use the anaconda distribution to create and switch environments
- each environment is like a fresh install
- but retains links to master packages when possible

why install environments?

- don’t want to mess up your system architecture
- you might want to install older versions of packages but don’t want to delete currently installed versions
- you might want different versions of python!

the conda environment

1. name environment
2. tell conda what packages to install
3. switch to environment

   tip: if you don't switch, you are **root**

4. install additional packages as needed

```
conda create --name MyFirstEnv numpy
conda create --name MyPython2Env python=2 numpy scipy

source activate MyPython2Env
conda install jupyter

source deactivate
```

conda packages

- not installed yet on conda for your operating system?
- use the conda cloud to find the right package for the right system

```
eclarson$ anaconda search -t conda rpy2
Packages:
     Name                          |  Version | Package Types   | Platforms
     ------------------------------ |  ------- | --------------- | ---------------
     Richarizardd/rpy2             |    2.5.6 | conda           | win-64
                                   :          : Python interface to the R language (embedded R)
     andywocky/rpy2                |    2.5.6 | conda           | osx-64
                                   :          : Python interface to the R language (embedded R)
     asmeurer/rpy2                 |    2.7.0 | conda           | linux-64, linux-32, osx-64
                                   :          : Python interface to the R language (embedded R)
     bce/rpy2                      |          | conda           | linux-64
                                   :          : Python interface to the R language (embedded R)
     bioconda/rpy2                 |    2.7.8 | conda           | linux-64, osx-64
                                   :          : Python interface to the R language (embedded R)
       r-old/rpy2                  |    2.4.2 | conda           | linux-64
       r/rpy2                      |    2.8.2 | conda           | linux-64, win-32, win-64, linux-3
       ralexx/rpy2                 |    2.7.6 | conda           | osx-64

eclarson$ conda install -c r rpy2
```

jupyter notebooks and conda environments

Scripting for Data Science in Python and R

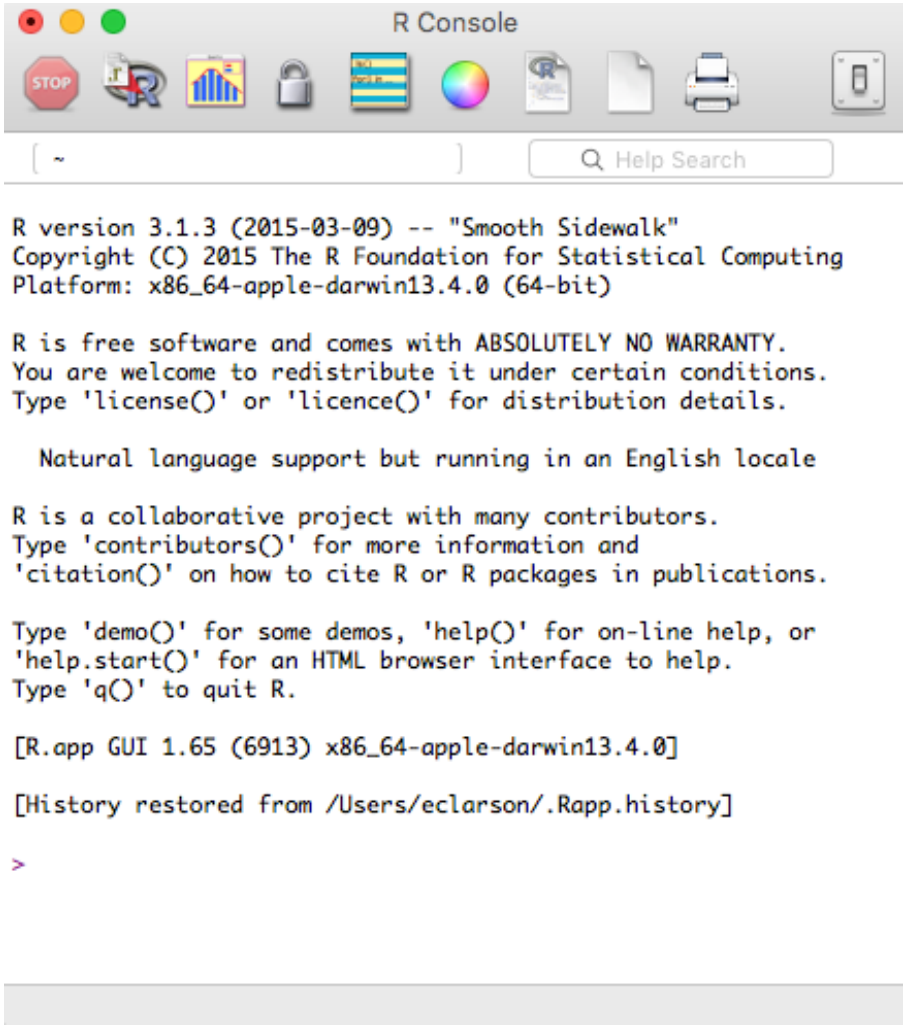
SMU Interdisciplinary Master's Degree in Data Science

Unit 3 - III. basic R syntax

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

installing R

- download and install from
  - https://www.r-project.org
  - use a CRAN mirror
- can use console
  - this is a “Session”
  - session has workspace
  - all memory saved in workspace
    - *a lot like jupyter does*



comments, variables, types in R


```
> x = 10 # this is assignment
> class(x) # what is the type of x?
[1] "numeric"
>
> y = as.integer(10)
> class(y)
[1] "integer"

> z = 10 + 0i
> class(z)
[1] "complex"

> j = sqrt(-1)
Warning message:
In sqrt(-1) : NaNs produced
> j = sqrt(as.complex(-1))
> j
[1] 0+1i
```

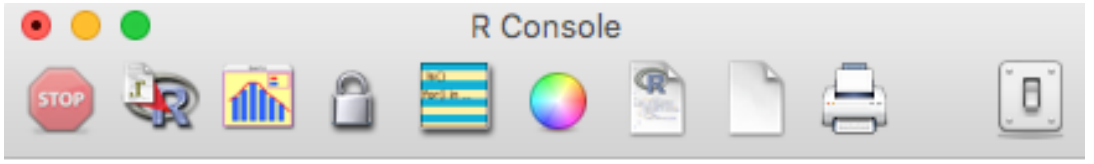
Comment given by #


```
> u=TRUE
> v=FALSE
> class(u)
[1] "logical"
> u&v
[1] FALSE
> u|v
[1] TRUE
> !v
[1] TRUE

> fname="Eric"
> lname="Larson"
> paste(fname,lname)
[1] "Eric Larson"
```

vectors in R

- Denoted by “c”


```
> vec = c(10,5,7)
> class(vec)
[1] "numeric"
> length(vec)
[1] 3

> comb_vec[1]
[1] "a"
> comb_vec[0]
character(0)
```




```
> str_vec = c("a","b","c")
> num_vec = c(10,5,7)
> comb_vec = c(str_vec,num_vec)
> comb_vec
[1] "a"  "b"  "c"  "10" "5"  "7"
> comb_vec[-2]
[1] "a"  "c"  "10" "5"  "7"
> series = c(1.0,5.6,3600)
> names(series) = c("Day","Temp","Sec")
> series
  Day   Temp    Sec
  1.0    5.6 3600.0
```

remove second element

conditionals and loops in R


```
> x = 1:10
> z = c()
> for(xi in 1:10) {
+     if(x[i] < 5) {
+         z = c(z, x[i] - 1)
+     } else {
+         z = c(z, x[i] / 2)
+     }
+ }
> z
 [1] 0.0 1.0 2.0 3.0 2.5 3.0 3.5 4.0 4.5 5.0
```


```
> if(1==0) {
+     print(1)
+ } else {
+     print(2)
+ }
[1] 2
```

highly similar to python syntax

defining functions R


```
> myfct = function(x1, x2=5) {
+      z1 = x1/x1
+      z2 = x2*x2
+      myvec = c(z1, z2)
+      return(myvec)
+ }
> myfct
function(x1, x2=5) {
     z1 = x1/x1
     z2 = x2*x2
     myvec = c(z1, z2)
     return(myvec)
}
> myfct(x1=2, x2=5)
[1]  1 25
> myfct(2, 5)
[1]  1 25
> myfct(x2=5, x1=2)
[1]  1 25
```


```
> myfct2 = function(x1=5, opt_arg) {
+      if(missing(opt_arg)) { #missing?
+           z1 = 1:10
+      } else {
+           z1 = opt_arg
+      }
+      cat("my function returns:", "\n")
+      return(z1/x1)
+ }
```

installing and using libraries in R

calling R from python: Rpy2

using R to extend the calculator

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 4 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 4 - II. vectorized coding in python

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

introduction to numpy

- numpy (numb-pie)
- written in c++ with python wrapper
  - fast but expressive
  - open source, produced by Travis Oliphant
- base for almost all scientific python modules including:
  - scipy, scikit-learn, pandas, scikit-image, opencv, matplotlib, and many more…

Travis Oliphant - CEO

- PhD 2001 from Mayo Clinic in Biomedical Engineering
- MS/BS degrees in Elec. Comp. Engineering
- Creator of **SciPy** (1999-2009)
- Professor at BYU (2001-2007)
- Author of **NumPy** (2005-2012)
- Started **Numba** (2012)
- Founding Chair of **Numfocus** / **PyData**
- Previous PSF Director

a review of linear algebra

some common operations in linear algebra

basic numpy operations

Scripting for Data Science in Python and R

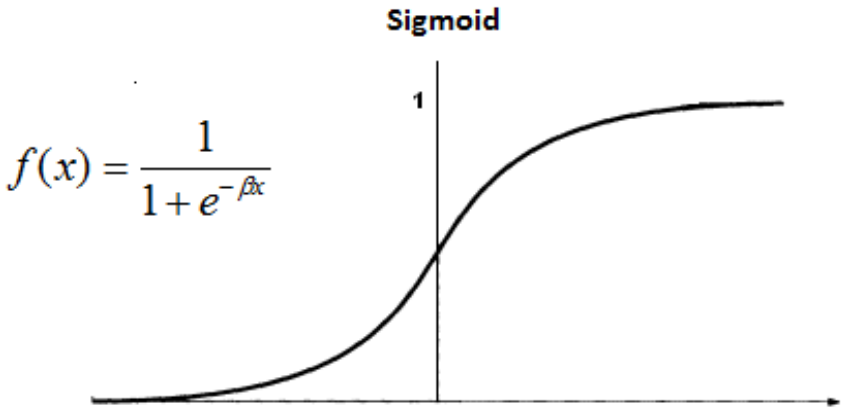
SMU Interdisciplinary Master's Degree in Data Science

Unit 4 - III. numpy for optimization

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

optimization with scipy/numpy

bonus: basic visualization in matplotlib
adding interactive widgets to notebooks



curve fitting a sigmoid

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 5 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 5 - II. putting it all together in python

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

extended example

analyzing the New York Times in **python**

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 6 - I. an introduction to the week

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

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 6 - II. data.frames and S4 classes in R

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

lists and data frames in R

- similar to pandas series and data frame


```
> v = list(bob=c(2, 3, 5),
           john=c("aa", "bb"))
> v
$bob
[1] 2 3 5

$john
[1] "aa" "bb"

> v$bob
[1] 2 3 5

> v$what
NULL
```


```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)
> df
  n  s     b
1 2 aa  TRUE
2 3 bb FALSE
3 5 cc  TRUE

> mtcars[1, 2]
[1] 6
> mtcars["Mazda RX4", "cyl"]
[1] 6
```

S4 classes in R

- S4 classes are the preferred method of class creation in R
- Caveat: object oriented in R is **awkward**
  - **not a strong suit** of the language
  - actually… it was an **afterthought** made to fit current design
  - its incredibly **convoluted**, and you should **not be happy** with the overall implementation
  - as a programmer it makes me want to 😥

S4 classes in R: initialization

- step one: define class name and properties

```
# now let's create an S4 object of the article
Article <- setClass("Article", slots=list(data='list',keys='character'))
```

- step two: create initializer, if necessary

```
#override the default initializer
setMethod(f="initialize",
          signature="Article",
          definition=function(.Object,data){
              .Object@data <- data # keep the data
              .Object@keys <- names(data)
              return(.Object)
          })
```

- class “Article” is now ready to use if you want:

```
a <- Article(some_article)
```

S4 classes in R: class methods

- step three: define class method as a **global function** 😒

```
#create a new function and define it
setGeneric('getParam',
           def=function(object,
                        param='type_of_material'){standardGeneric('getParam')})
```

- step four: define function

```
setMethod(f='getParam',signature='Article',
          definition=function(object,param){
              if(param %in% object@keys){
                  return(object@data[[param]])
              }
              else{
                  return(c(''))
              }
          })
```

- calling class methods, you need to provide the class instance 😒

```
a <- Article(some_article)
getParam(a,'lead_paragraph')
```

S4 classes in R: inheritance

- step five: define define a subclass of Article

```
ArticleWithMedia <- setClass("ArticleWithMedia",
                             slots=list(data='list',keys='character',media='list'),
                             contains="Article")
```

- step six: initializing inherited class

```
setMethod(f="initialize",
          signature="ArticleWithMedia",
          definition=function(.Object,data){
              .Object@media <- vector("list", length(data$multimedia))
              for(i in 1:length(data$multimedia)){
                  .Object@media[i] <- NYMedia(data$multimedia[[i]])
              }

              # now call the inherited initializer for the class
              callNextMethod(.Object,data)
          })
```

- you have **no control** over initialization hierarchy 😒

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 6 - III. putting it all together in R

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

extended example

analyzing the New York Times in **R**