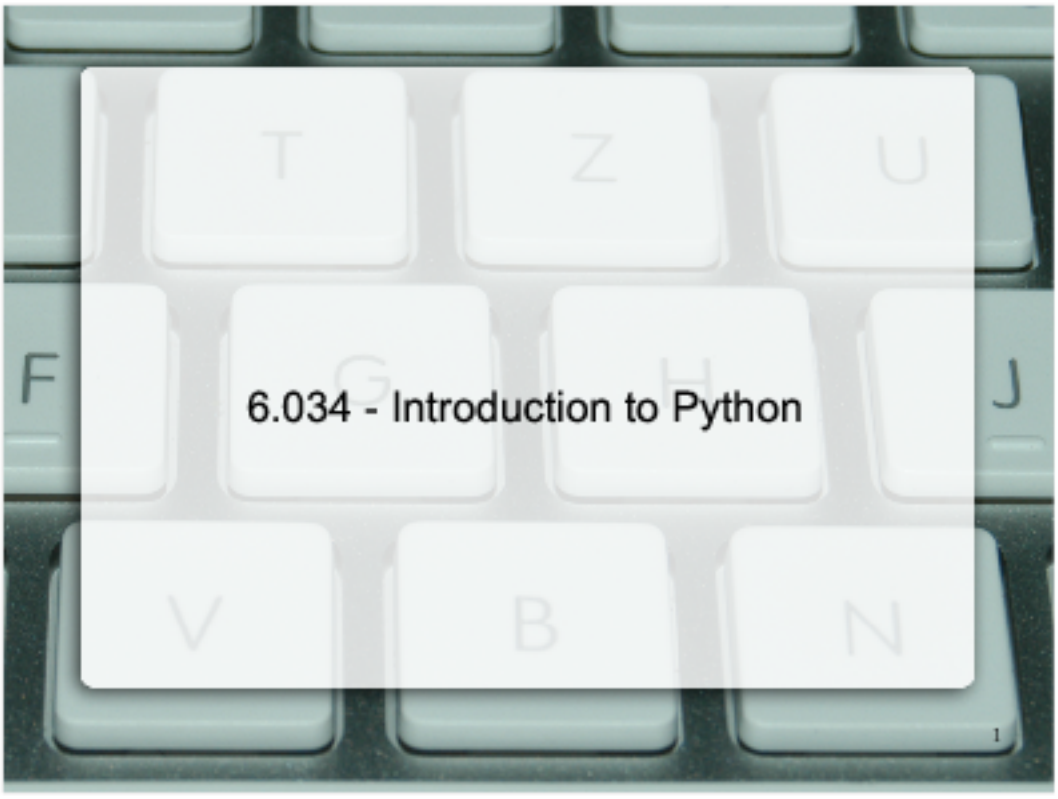
6.034 - Introduction to Python
1


Overview
• Basics
• Syntax Review
• Gotchas
• Scripting
• Objects
• User Interfaces
python
2

Basics

- Python is not a strongly-typed language, so no need to predefine anything

  a=5

  a= “5”

  a=[5,'foo',list(),dict()]

- The above statements are all valid. Python does not care that **a** previously stored an int, nor does it care if all the objects are the same in a list, dict, etc.

Python primitive objects

int	integer	A = 5 A = int(5.3)
float	Floating point number	A = 5.3 A = float(5)/7
char	A single character	A = 'c' A = any_str[i]
str	A string (list of characters)	A= "Hello world"
list	A mutable array object	A= range(0,10,1) A=[5,4,3,2,1] A=list() A.append(4)
dict	Hashmap of key::value	A=dict() A['red']='apple' A['yellow']='banana'
set	Unordered collection of objects with duplicates removed	A=set() for elt in mylist: A.add(elt) print A

Type *help(object)* at the python command prompt for more information about that object

Importing Packages

- Python has many useful packages you may want to import
- You can also import your own modules

```
import sys
import os
from Tkinter import *
import threading
import argparse
import Queue
from myFile import myClass
```

Useful Utility Methods

- sum – returns the sum of a list
- len – returns the length of a list
- max – returns the max element in a list
- min – returns the min element in a list
- abs – absolute value
- eval – evaluates a str input and returns result
- isinstance – method to see if an object is of a certain type
- More here: http://docs.python.org/library/functions.html

Lambdas, Filter, Map, Reduce

- lambda functions – anonymous functions - allows you to create methods on the fly

  my_square = lambda x : x**2

  my_square(8) → 64
- filter – applies a method to each element in a list and returns the list of elements for which the method returned True

  primes = filter(my_is_prime_method,my_list)
- map – applies a method to each element in a list and returns the resulting list

  student_ids = map(lambda student : student.id,my_students)
- Reduce – apply a method of two arguments cumulatively to a list

  my_sum = reduce(lambda x,y : x+y, my_list)

List Sorting

- You can write your own methods to do fast sorts, but there is an in-place sorter that is build in to python lists

  a=[5,4,3,1]

  a.sort()

  a → [1, 3, 4, 5]

  a=[(1,2),(5,4),(0,6)]

  a.sort()

  a → [(0, 6), (1, 2), (5, 4)]

  help(a.sort)

Printing

- There are a few ways to print statements

  print “Print this one thing”

  print “Print this”,"print that"

  print “Print value of an int: "+str(myint)

  print “Print value of an int: %d"%(myint)

  print “Print value of int %d, float %f, string %s"% (myint,myfloat,mystring)

  print “Print int {0} : string {1}”.format(myint,mystring)

If elif else

if *c1* or *c2* or (*c3* and *c4* and not *c5*):

```
    #do something here
    first_condition_met()
```

elif *c5*:

```
    alternative_met()
```

else:

```
    else_condition()
```

For Loops

```
a = list(0,1,2,3,4,5,6,7,8,9)
b = [object1,object2,object3,object4]
myfile = open("myfile",'r')
for elt in a:
    print elt
for elt in b:
    print elt
for i in range(10):
    print i
for line in myfile:
    print line
```


List Comprehensions

- Assume we start with some lists

  A = range(10) #[0,10) counting by 1s

  B = range(0,100,5) #[0,100) counting by 5s
- Now, let's make new arrays based off of these

  squares = [i**2 for i in A]

  some_matrix = [[i*j for i in A] for j in B]

  evens = [i for i in A if i%2==0]

  #noprimes consists of all multiples of 2,3,4,5,6,7,8 from [4,50)

  noprimes = [j for i in range(2,8) for j in range(i*2,50,i)]

  #primes is integers from [2,50) that are not in noprimes

  primes = [x for x in range(2,50) if x not in noprimes]
- Note: Everything done with a list comprehension can be done with a for-loop. Think of this as syntactic sugar

While loops

- Similar to for-loops

```
while a<50 and not_done:
    if some_dict.has_key(a):
        print some_dict[a]
        break
    a+=1
while a<50:
    if(a%2==0):
        continue
    print a
    a+=1
```


Methods

- You need to define methods to break your code into digestible components

```
#Note: The equals signs assign a default value to each argument, so if the #argument is not specified, it still has a value
def average(arg1=2,arg2=5):
    #something happens here
    return (arg1+arg2)/2.0
```

- Note: no return type is specified. No return value is necessary. If there is no return statement, a None object is returned
- Class methods are similar, though the first argument is always self:

```
#note: the *args means it takes any number of arguments
def myObjectsAverage(self,*args):
    objects_to_average = [self.mydict[i] for i in args if self.mydict.has_key(i)]
    if len(objects_to_average)<1:
        return 0.0
    return sum(objects_to_average)/len(objects_to_average)
```

Recursion

```
def sum_up_to(n):
    if n==0:
        return 0
    return n+sum_up_to(n-1)
```

- Base Case(s)
- Recursive call
- Fibonacci, Factorial, etc

Try/Except

- For statements that might throw exceptions, wrap them in a try/except block

```
try:
    this_might_fail()
except ValueError:
    print "This failed because of a value error"
except IOError as (errno,strerror):
    print "I/O error(%d): %s"%(errno,strerror)
except:
    print "failed for some other reason"
    raise
```

- Raise is like a "Throw" statement in Java

for i in range(*lower*,upper,*step*)

- The **range** function is a bit strange
- It returns a list [*lower*,upper), stepping by step, meaning that it includes the first and excludes the last.
- It does not require a *lower* or a *step*. The default lower is 0 and the default step is 1.

Significant Whitespace

- You may have noticed that python does not have a marker to note the end of a section
- The “pass” keyword can be used as a section end marker, but it is usually only used as a placeholder
- Most loops, methods, etc are demarcated by their indentation level, so whitespace matters

Run python code as a script

- Start the file with the line:

  #!/usr/bin/env python

- Include the following if-statement:

  if __name__ == "__main__":

  ```
  #code goes here
  #I personally just do argument parsing here
  #and do the meat of my program in other methods
  print "Hello world"
  ```


Create Classes

- Python can create objects

```
class MyObject(object):
    def __init__(self,arg1,arg2,arg3):
        #This is the constructor of MyObject
        self.vars = [arg1,arg2,arg3]
if __name__== "__main__":
    myobj = MyObject(42, "foo", "bar")
```

- Python can do inheritance

```
class MyCoolerObject(MyObject):
    def __init__(self,arg1,arg2,id):
        MyObject.__init__(self,arg1,arg2,"bar")
        self.id = id
```


Multiple Inheritance

- Python can handle multiple inheritance

```
class MyCoolestObject(MyCoolerObject,dict):
    def __init__(self,arg1,arg2,id):
        MyCoolerObject.__init(self,arg1,arg2,id)
        dict.__init__(self)

    #This overrides dict's getitem method, which is usually
    #invoked via → mydict[i]
    def __getitem__(self,key):
        if not self.has_key(key):
            super(MyCoolestObject,self).__setitem__(key,[])
        return super(MyCoolestObject,self).__getitem__(key)
```

Tkinter

- Tkinter is an easy-to-use ui builder package
- It has Frame, Button, Label, Entry, Scrollbar, Canvas, and much more

```
from Tkinter import *
class MyApp(object):
    def __init__(self):
        self.root = Tk()
        self.label = Label(self.root, text="Hello World")
        self.label.pack()
    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    myapp = MyApp()
    myapp.run()
```