Python

Methodologies for Data Science

Lars Sorensen - 16:137:603
Fall 2015 - biglars@cs.rutgers.edu
Unit Four

Python Methodologies for Data Sciences

Take a breath. You've done a lot in the last seven weeks…

Loading Python on your machine - Figuring out how to run IDLE - Using the Python interactive command window

Creating programs - Print function - Variables - Assignment - Arithmetic

Modulus - Strings - Integers - Floats - Casting - Input function

Booleans - Arithmetic conditionals (>, <, ==, !=, <=, >=) - Boolean operators (And, Or, Not) - if - elif - else

Sequences - strings - slicing - indexing, reverse indexing - len function - ord, chr - string methods

Lists - split method - append method - mutability of lists - mixed data types of lists

Loops - Iteration - for loops - range function - nested loops - while loops - sentinel loops - counters - accumulators

Interactive loops - definite loops - indefinite loops - keyword 'in' as set conditional

Functions - arguments - return types - scope - File I/O - opening, closing, reading from files, writing output to files, pickling

importing libraries - the Python Standard library - writing your own library - from import - third party modules, etc

Now that's a whole lot of stuff….

Python Methodologies for Data Sciences

Welcome, my brave Python soldiers, to Unit 4. Unit 4 is going to be fun, trust me on this, because we are going to be going behind the scenes and learning about some Computer Science (CS). For the most part, I took the beginning of an Intro to CS course and extracted the CS from it to get you the first three units. At the end of the day, I'm trying to make you Data Science programmers, not Computer Scientists, but now that you have the basics of an imperative programming language under your belt (and you do, you're all Python programmers now. Congrats!) I am going to give you a quick survey of some of the CS topics I extracted.

As I am going to explain, when we first learn computer science we get hit with tons about sorting and searching. When you're a 19 year old kid like I was at this stage, you just push through and don't understand why this tedium is necessary. When you become an old seasoned CS dog you realize that all that sorting was the best way to introduce new students to algorithmic efficiency and that different ways of doing things ARE NOT equal.

As a Data Science class we will be interested in searching as well. We're going to find out that the more we know about our data the more efficient we can be as we examine it. If I know my data is sorted I can take a search operation that would take one million steps to complete with sloppy data and complete it in twenty steps. That's right, Twenty.

Python Methodologies for Data Sciences

We are also going to look at one of the most vexing issues a young CS student can run into. Recursion.

It is not as hard as its reputation. It's a simple concept actually, and it's been proven that the most common kind of recursion (the kind called tail recursion) is theoretically indistinguishable from a loop. We are not going to bend your mind too much, but recursion gives us a chance to show you how to take advantage of the ways that computer systems and memory work.

We are going to load a third party module, matlibplot, and create a graph with some data. Some actual Data Science? Alert the media. Yes, we finally have enough chops to start doing some real data science.

Lastly, we are going to clean up some of the orphans as promised. Eric is going to be happy as we will be covering the data structure that is dictionaries and we will be looking at ways to keep our programs from crashing by learning about something called "exception handling."

Sounds like a ton but it's not. Let the fun begin.

Sorting and Searching

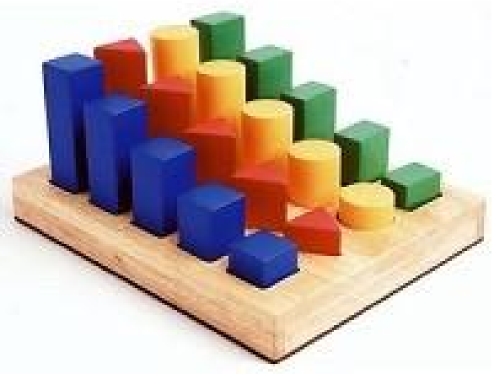
Python Methodologies for Data Sciences

Ever since the sorting hat put me in Slytherin I’ve been trying to prove that I’m actually a nice guy and get myself transferred out.

In order to work towards this goal I have decided that we are just going to talk about sorting and I am not going to force you (as I was forced) to code up seven useless sorts that you will never use or think about again. (That said, feel free to code up the selection and merge sort you see in Zelle.)

We will code up a search, but it’s going to be fun as it’s just a game of high and low...

Python Methodologies for Data Sciences



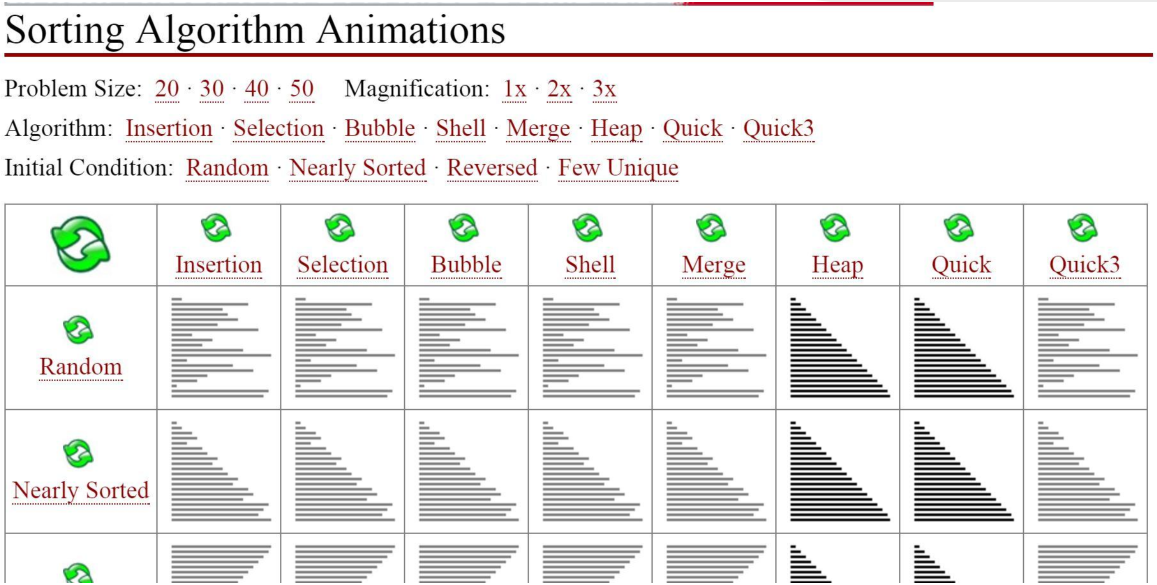
Sorting is merely taking a sequence and putting it in order. It can be numeric order or it can be lexicographic order (alphanumeric), it can be forwards or it can be backwards, but at the end of the day we will have a sequence of objects that can be said to have all the items on one side of it lesser than or equal to it and one side with all of the objects greater than or equal to it.

There are many different ways to sort lists. The first one I was ever shown was called a bubble sort. You went to the bottom of the list and started comparing the bottom to what was above. If the thing above it was greater then you swapped them. Then you went back down to the bottom and started over. You repeated this process until your whole list was in order. A bubble sort (bubble because things ‘bubbled’ to the top of the list) is terribly inefficient. Every time the computer does a comparison or does a swap is a use of resources. This takes time. When we talk about efficient sorting or efficient algorithms we discuss how long they take, how many operations they take and how much memory they use.

Python Methodologies for Data Sciences

Take a few minutes and examine the different sorting algorithms at the sorting algorithms website. Click on an unsorted graphic and watch it get sorted.

Run Bubble and then run the QuickSort to see how differently they work. If you are dead set on coding a sorting algorithm you can find the code (*psuedocode*, that means easily readable words that aren't really a computer language) with the algorithm links on top of the page.



Sorting Algorithm Animations

Problem Size: 20 · 30 · 40 · 50 Magnification: 1x · 2x · 3x

Algorithm: Insertion · Selection · Bubble · Shell · Merge · Heap · Quick · Quick3

Initial Condition: Random · Nearly Sorted · Reversed · Few Unique

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								

http://www.sorting-algorithms.com/

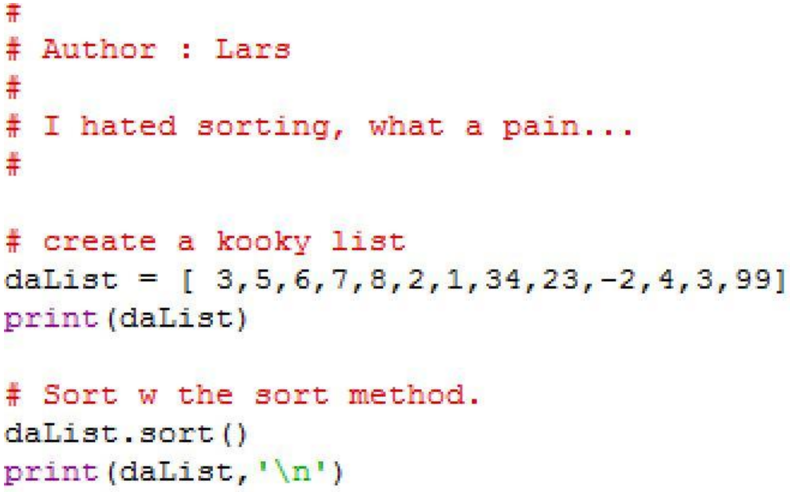
Python Methodologies for Data Sciences

To quickly sort a list in Python we can use the sort() method that is available for lists.

Python uses a sort called the timsort.

https://en.wikipedia.org/wiki/Timsort

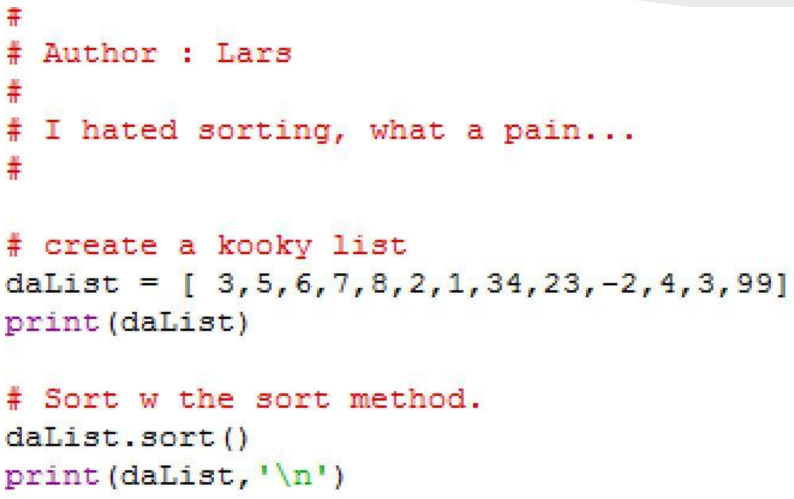
While many sorts have been around for a long time, the timsort has only been around since 2002 and was developed just for Python.



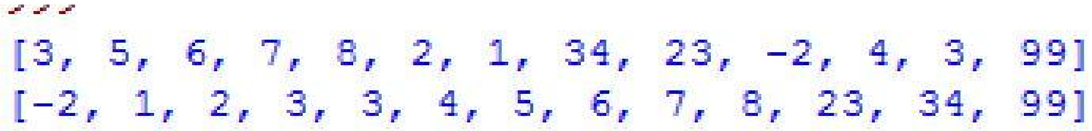
```
#
# Author : Lars
#
# I hated sorting, what a pain...
#

# create a kooky list
daList = [ 3,5,6,7,8,2,1,34,23,-2,4,3,99]
print(daList)

# Sort w the sort method.
daList.sort()
print(daList,'\n')
```

Python Methodologies for Data Sciences

As we can see below the sort() method sorts the items of the list in place. Where daList[2] was 6, it is now equal to 2.



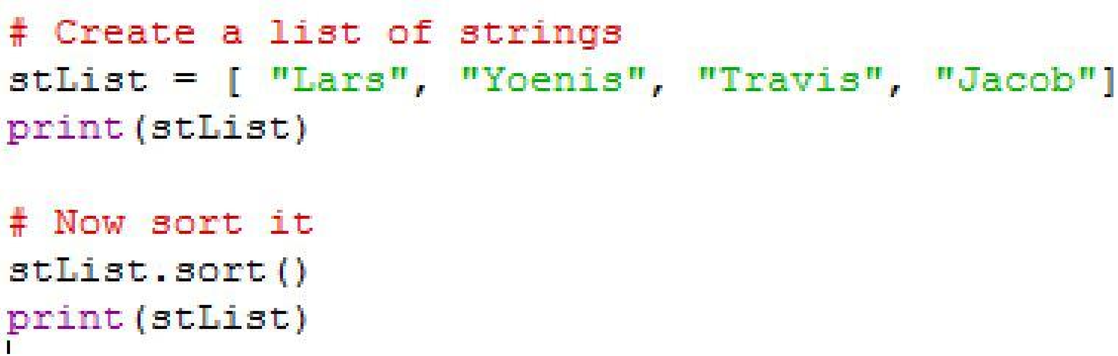
```
#
# Author : Lars
#
# I hated sorting, what a pain...
#

# create a kooky list
daList = [ 3,5,6,7,8,2,1,34,23,-2,4,3,99]
print(daList)

# Sort w the sort method.
daList.sort()
print(daList,'\n')
```



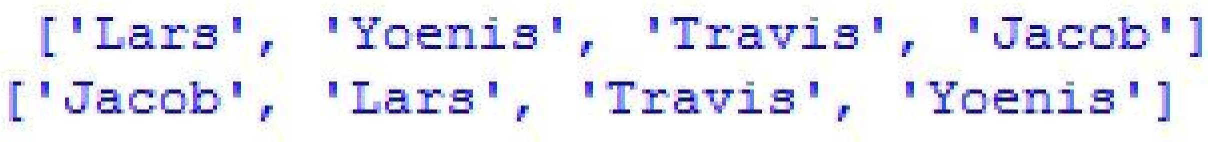
```
[3, 5, 6, 7, 8, 2, 1, 34, 23, -2, 4, 3, 99]
[-2, 1, 2, 3, 3, 4, 5, 6, 7, 8, 23, 34, 99]
```

Python Methodologies for Data Sciences

We can also sort strings with the sort() method.

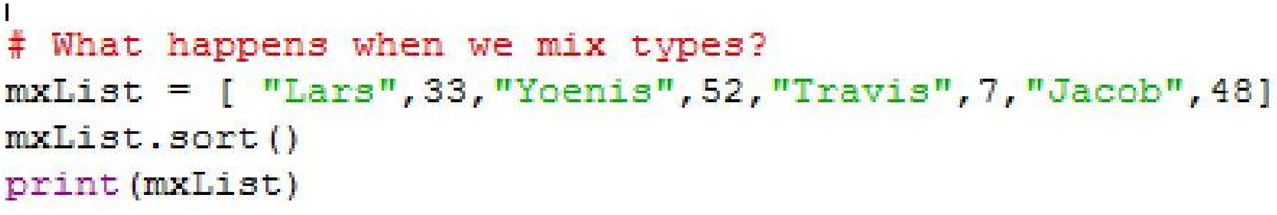


```
# Create a list of strings
stList = [ "Lars", "Yoenis", "Travis", "Jacob"]
print(stList)

# Now sort it
stList.sort()
print(stList)
```



```
 ['Lars', 'Yoenis', 'Travis', 'Jacob']
['Jacob', 'Lars', 'Travis', 'Yoenis']
```

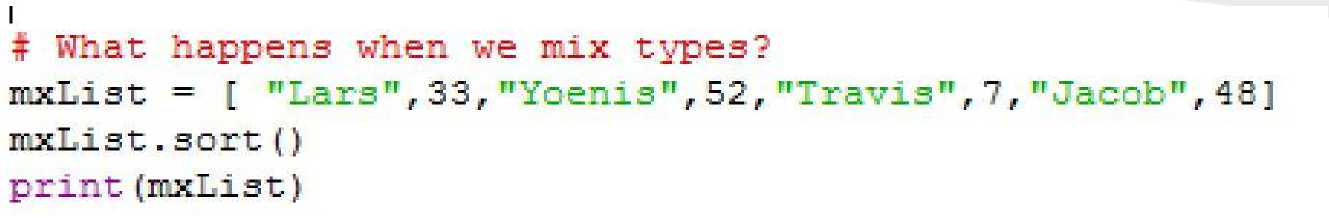
Python Methodologies for Data Sciences



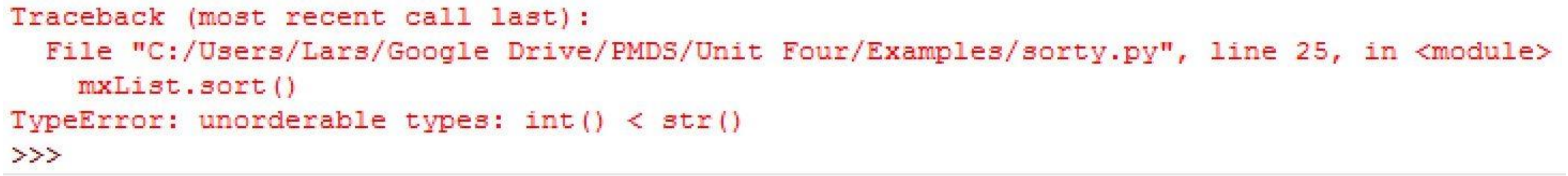
```
# What happens when we mix types?
mxList = [ "Lars",33,"Yoenis",52,"Travis",7,"Jacob",48]
mxList.sort()
print(mxList)
```

Yikes. Here's a list with mixed data types. What happens when we try to sort a list like this?

Python Methodologies for Data Sciences

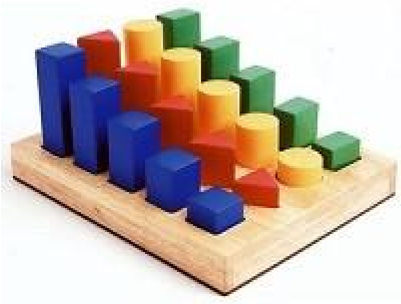


```
# What happens when we mix types?
mxList = [ "Lars",33,"Yoenis",52,"Travis",7,"Jacob",48]
mxList.sort()
print(mxList)
```



```
Traceback (most recent call last):
  File "C:/Users/Lars/Google Drive/PMDS/Unit Four/Examples/sorty.py", line 25, in <module>
    mxList.sort()
TypeError: unorderable types: int() < str()
>>>
```

We get an error. Remember when you sort a list they have to ALL be of the same data type so Python knows they can be put in some kind of order.



Why we care so much about sorting

As I said before, we care about sorting early in computer science education because it is a good way to introduce the topic of algorithm efficiency to students. We will do this too as we are going to talk about Big O in a moment.

But there is another reason we care about sorting. When we have a sorted list we know certain things about our data. This makes searching through it for specific items easier. Let’s explore searching...

Searching

When we have large sets of data sometimes we want to search it to see if certain items are in the list. We can search for names, numbers, anything that resides in a list.

The easiest way to do it is to go into a loop and look for what you are seeking. This is called a linear search. If we have a list of one million items, the item we are seeking might be at the end of the list so, at times, we'll have to do a lot of searching or "look-ups" to see if our item is there.

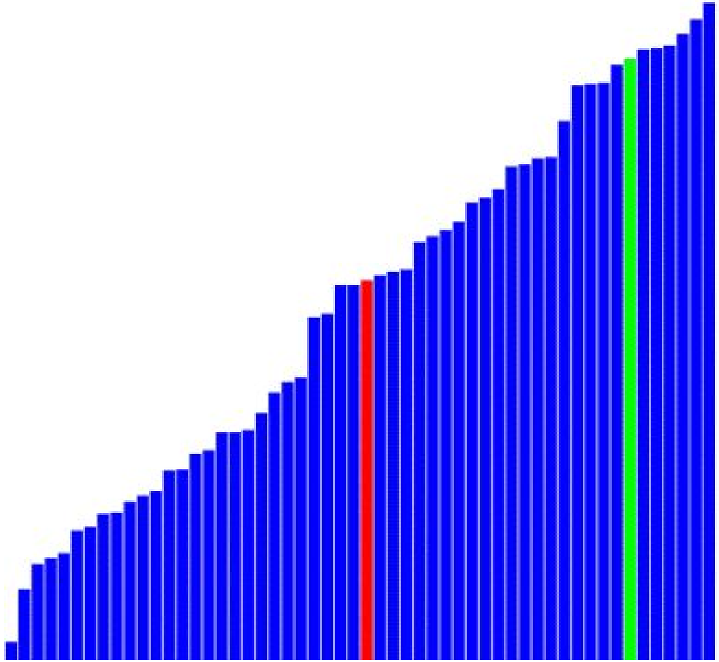
What if there were a way to find items more efficiently? That's where sorting comes into play...

Binary Search

If I know my data is sorted I can play a computer science version of the game High/Low.

If I need to search a list that's one million items long I can look for my item in the middle of the list. As I compare the search item to the middle item in the list I can also check if it's higher or lower than that item. Because I have a sorted list this will instantly let me discount HALF of my list! With one lookup I have cut my search area down by 500,000 numbers! That's powerful.

That's the idea behind a search method called Binary Search.

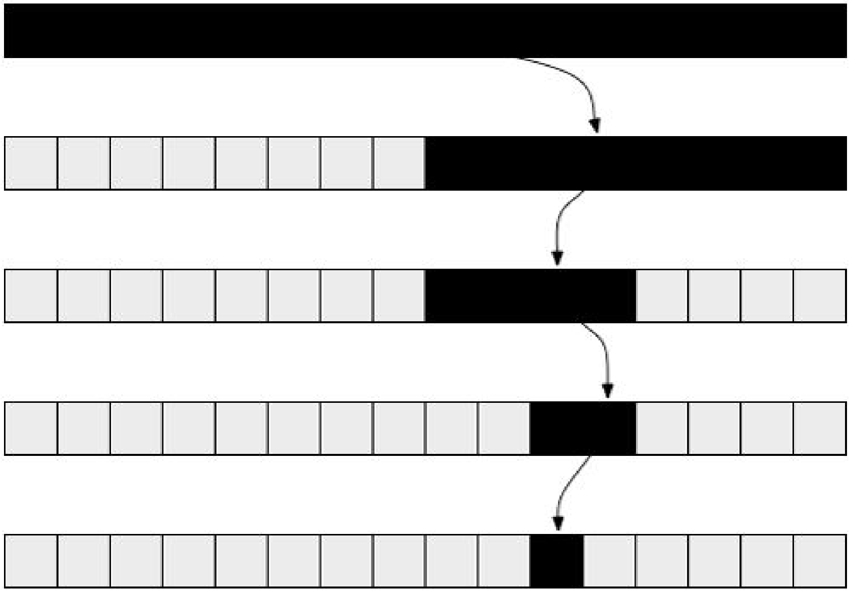


Python Methodologies for Data Sciences

Here's a list where we search for eleven in a list sixteen long. We go to the middle. Higher or lower? It's higher (11>8), so we discount the bottom of the list and only search 9 thru 16.

We then look at thirteen. Now I'm lower. We remove 13 through 16 and only have 9 through 12 to search now.

This process continues, cutting the existing list in half each time we go through the loop. In only 5 lookups we have successfully searched a list 16 items long. Had we used linear search our worse case scenario would have been 16 lookups. That's a big savings in efficiency.



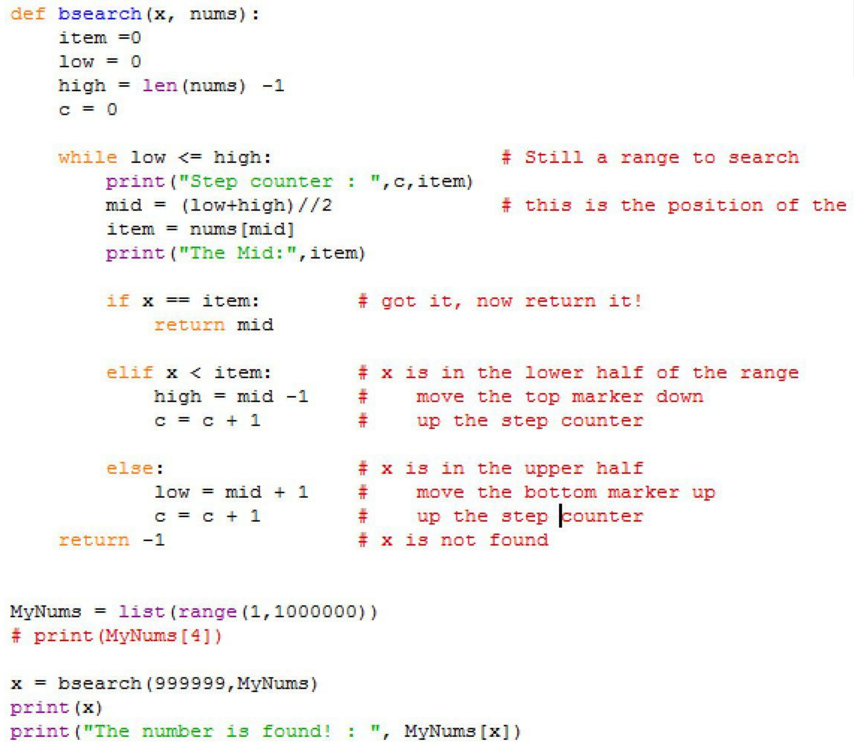
Python Methodologies for Data Sciences

Do not start typing, the code to the right is in your Unit 4 resource folder.

This is a binary search executed in Python. Read it and follow what it's doing. Trace through it and understand it.

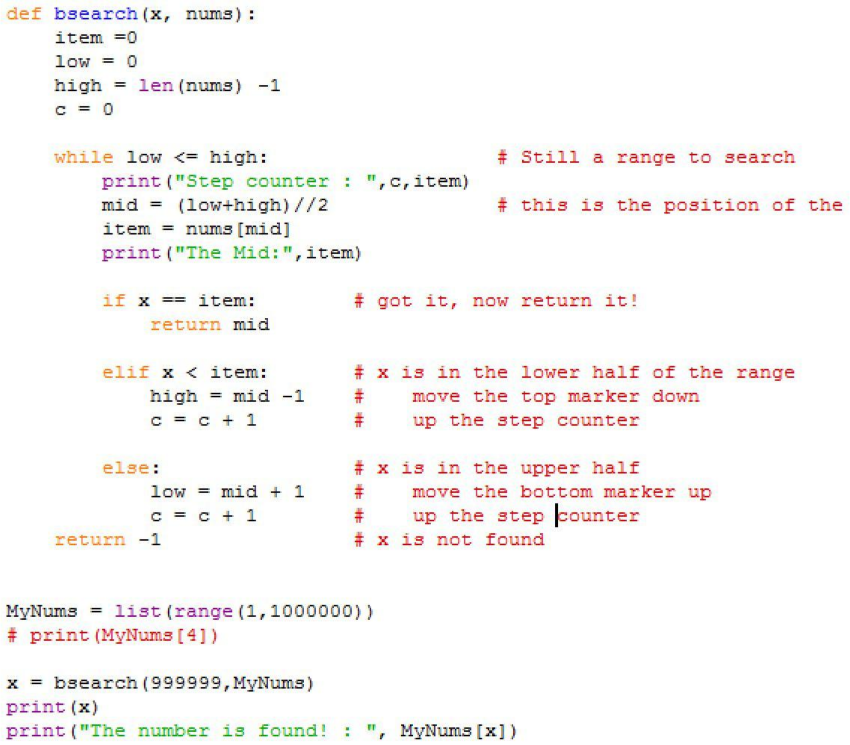
I create a list of numbers from 1 to 999999 and search for the last number. In a linear search that would be a million lookups.

With binary search, by cutting the list in half every time I search, we can cut that down to just 20 lookups. That's the power of sorted data. Look at the next slide...



```
def bsearch(x, nums):
    item =0
    low = 0
    high = len(nums) -1
    c = 0

    while low <= high:                        # Still a range to search
        print("Step counter : ",c,item)
        mid = (low+high)//2                   # this is the position of the
        item = nums[mid]
        print("The Mid:",item)

        if x == item:            # got it, now return it!
            return mid

        elif x < item:           # x is in the lower half of the range
            high = mid -1        #    move the top marker down
            c = c + 1            #    up the step counter

        else:                    # x is in the upper half
            low = mid + 1        #    move the bottom marker up
            c = c + 1            #    up the step counter
    return -1                    # x is not found


MyNums = list(range(1,1000000))
# print(MyNums[4])

x = bsearch(999999,MyNums)
print(x)
print("The number is found! : ", MyNums[x])
```

Python Methodologies for Data Sciences

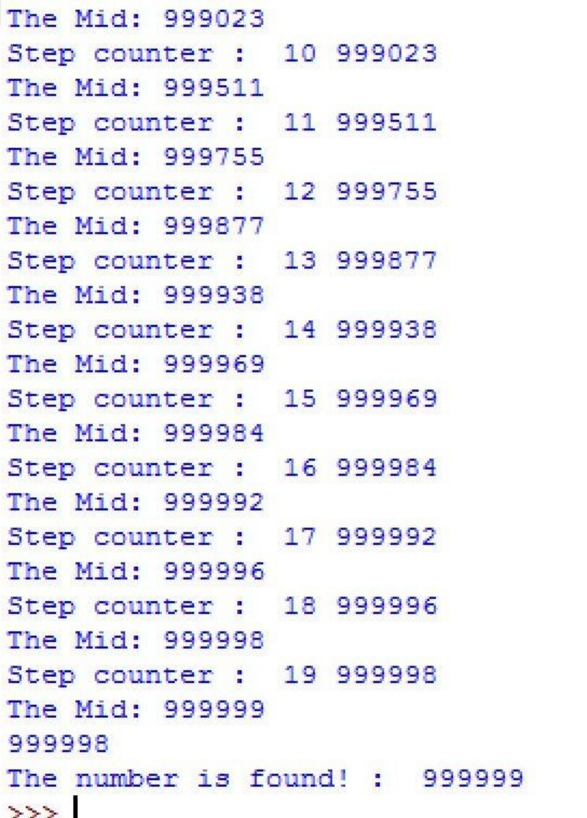


```
def bsearch(x, nums):
    item =0
    low = 0
    high = len(nums) -1
    c = 0

    while low <= high:                  # Still a range to search
        print("Step counter : ",c,item)
        mid = (low+high)//2             # this is the position of the
        item = nums[mid]
        print("The Mid:",item)

        if x == item:          # got it, now return it!
            return mid

        elif x < item:         # x is in the lower half of the range
            high = mid -1      #    move the top marker down
            c = c + 1          #    up the step counter

        else:                  # x is in the upper half
            low = mid + 1      #    move the bottom marker up
            c = c + 1          #    up the step counter
    return -1                  # x is not found


MyNums = list(range(1,1000000))
# print(MyNums[4])

x = bsearch(999999,MyNums)
print(x)
print("The number is found! : ", MyNums[x])
```

I show the end of my output to the right here.

In twenty lookups I have found my number.

This is very efficient, especially when compared with a one million lookup linear search. We will look at this efficiency when we discuss Big O in a moment.



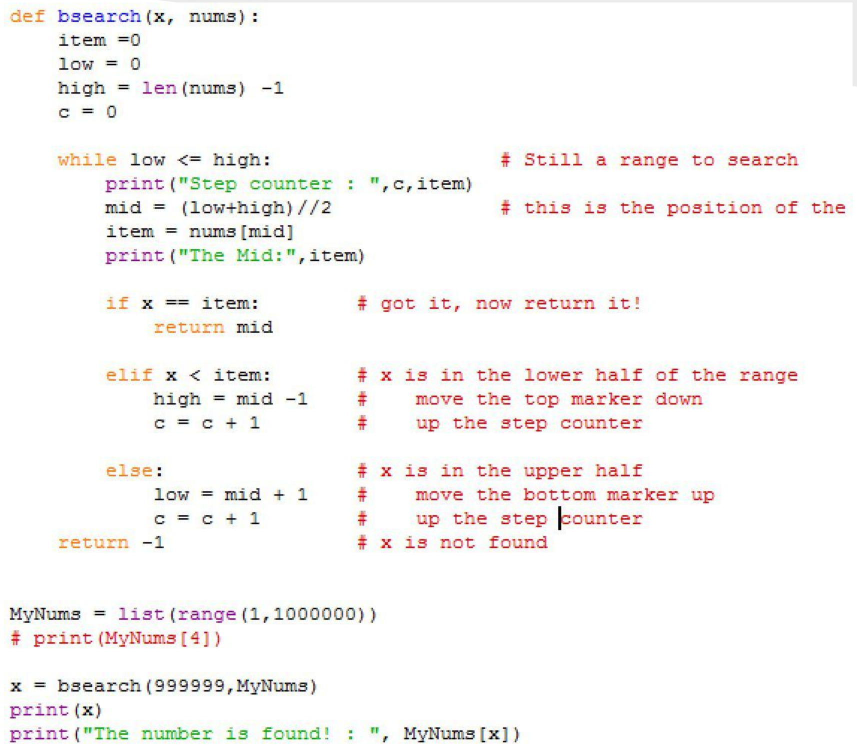
```
The Mid: 999023
Step counter :  10 999023
The Mid: 999511
Step counter :  11 999511
The Mid: 999755
Step counter :  12 999755
The Mid: 999877
Step counter :  13 999877
The Mid: 999938
Step counter :  14 999938
The Mid: 999969
Step counter :  15 999969
The Mid: 999984
Step counter :  16 999984
The Mid: 999992
Step counter :  17 999992
The Mid: 999996
Step counter :  18 999996
The Mid: 999998
Step counter :  19 999998
The Mid: 999999
999998
The number is found! :   999999
>>>
```

Python Methodologies for Data Sciences

Grab the code from the resource folder and play around with a few different searches. Make the range bigger and see how many lookups it takes to find a number.

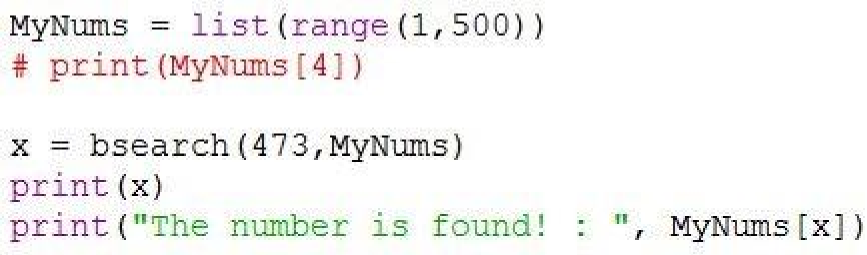
If you want, code up a linear search for the same problem and compare the two.

(Guessing this will happen in a YouTube video methinks...)

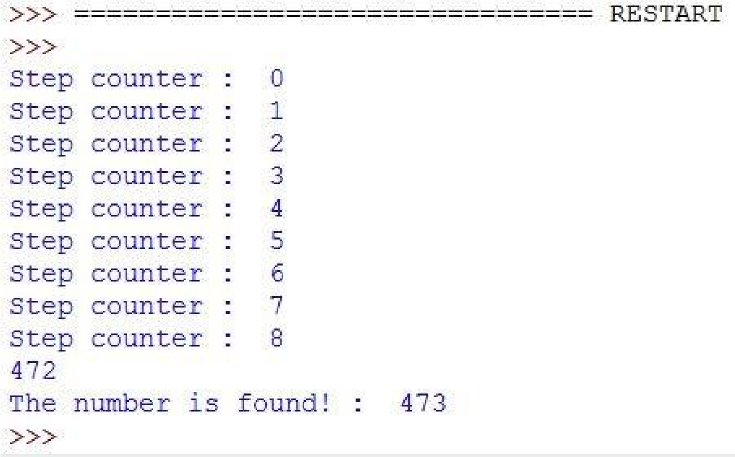


```python
def bsearch(x, nums):
    item =0
    low = 0
    high = len(nums) -1
    c = 0

    while low <= high:                   # Still a range to search
        print("Step counter : ",c,item)
        mid = (low+high)//2              # this is the position of the
        item = nums[mid]
        print("The Mid:",item)

        if x == item:        # got it, now return it!
            return mid

        elif x < item:       # x is in the lower half of the range
            high = mid -1    #    move the top marker down
            c = c + 1        #    up the step counter

        else:                # x is in the upper half
            low = mid + 1    #    move the bottom marker up
            c = c + 1        #    up the step counter
    return -1                # x is not found


MyNums = list(range(1,1000000))
# print(MyNums[4])

x = bsearch(999999,MyNums)
print(x)
print("The number is found! : ", MyNums[x])
```

Python Methodologies for Data Sciences



```
MyNums = list(range(1,500))
# print(MyNums[4])

x = bsearch(473,MyNums)
print(x)
print("The number is found! : ", MyNums[x])
```



```
>>> ================================ RESTART
>>>
Step counter :  0
Step counter :  1
Step counter :  2
Step counter :  3
Step counter :  4
Step counter :  5
Step counter :  6
Step counter :  7
Step counter :  8
472
The number is found! :  473
>>>
```

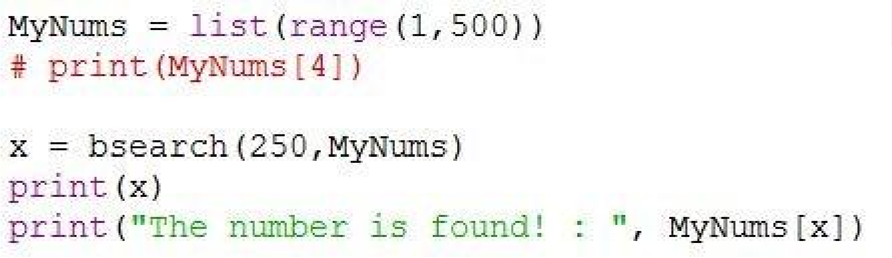
When we look at efficiency we often discuss the worst and best case scenario for the workings of algorithms. With a list of 500 elements, an 8 step binary search is nearing a worst case scenario. Still, this is incredibly efficient when compared to linear search.

Python Methodologies for Data Sciences

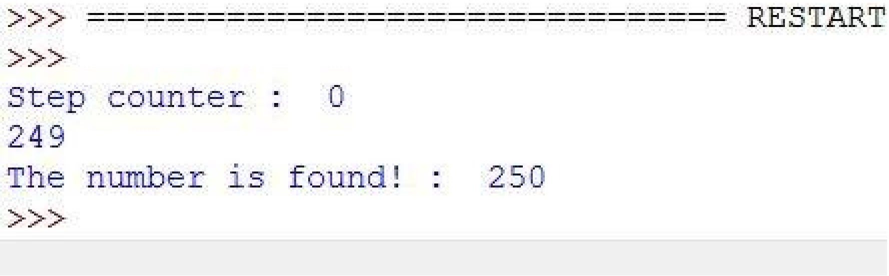
Here we see a best case scenario for a binary search. With a linear search we'd need 250 lookups to find 250. here we get it on the first lookup.

When our data is sorted it tells us something important about the format of our data that allows us to use algorithms that save incredible amounts of time and work.

It's easy for me to say it, now let's be more rigorous about these facts.



```
MyNums = list(range(1,500))
# print(MyNums[4])

x = bsearch(250,MyNums)
print(x)
print("The number is found! : ", MyNums[x])
```

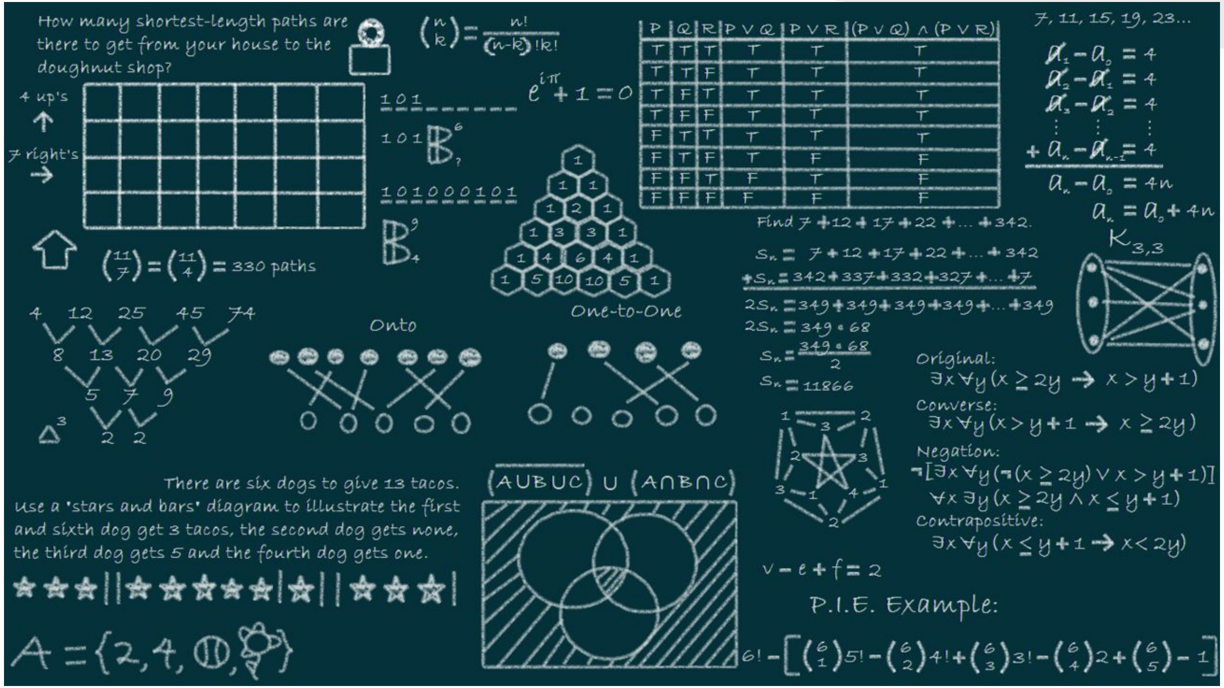


```
>>> ================================ RESTART
>>>
Step counter :  0
249
The number is found! :  250
>>>
```

Python Methodologies for Data Sciences

Remember, we cared about sorting not just for searching but because it’s a good way to show beginning programmers that different algorithms have different efficiencies, especially when the input becomes bigger.

This leads us to the analysis of algorithms and Big O.


How many shortest-length paths are there to get from your house to the doughnut shop?
4 up's
7 right's
(11 7) = (11 4) = 330 paths
(n k) = n!/(n-k)!k!
e^iπ + 1 = 0
101
101000101
P Q R P∨Q P∨R (P∨Q)∧(P∨R)
7, 11, 15, 19, 23...
a_n = a_0 + 4n
Find 7+12+17+22+...+342.
S_n = 11866
K3,3
One-to-One
Onto
Original: ∃x∀y(x ≥ 2y → x > y+1)
Converse: ∃x∀y(x > y+1 → x ≥ 2y)
Negation: ¬[∃x∀y(¬(x ≥ 2y) ∨ x > y+1)]
∀x∃y(x ≥ 2y ∧ x ≤ y+1)
Contrapositive: ∃x∀y(x ≤ y+1 → x < 2y)
There are six dogs to give 13 tacos. Use a 'stars and bars' diagram to illustrate the first and sixth dog get 3 tacos, the second dog gets none, the third dog gets 5 and the fourth dog gets one.
(A∪B∪C) ∪ (A∩B∩C)
v − e + f = 2
P.I.E. Example:
A = {2, 4, ...}

Big O

Python Methodologies for Data Sciences

Do you know your NBA history? Me too. We’re not talking about this Big O though...

Do you watch anime? Me too. But we’re not talking about this Big O either....

Python Methodologies for Data Sciences

Big O is just the notation used by programmers when they examine how efficient their algorithms are as their inputs get bigger. That's it. End of story. It's nothing confusing or crazy. People ascribe all kinds of crazy things to Big O, but it's rather simple.

The O come the word Order in "Order of Magnitude." That's because when we look at algorithms we don't sweat the small stuff and usually just look at the largest term (if we have a polynomial) so when we look at O(n) or O(1) we're dealing with different levels, not things that are close to each other. Big O is just a function that maps the number of items in a list versus the number of operations needed to search or examine the list in a "worst case" scenario.

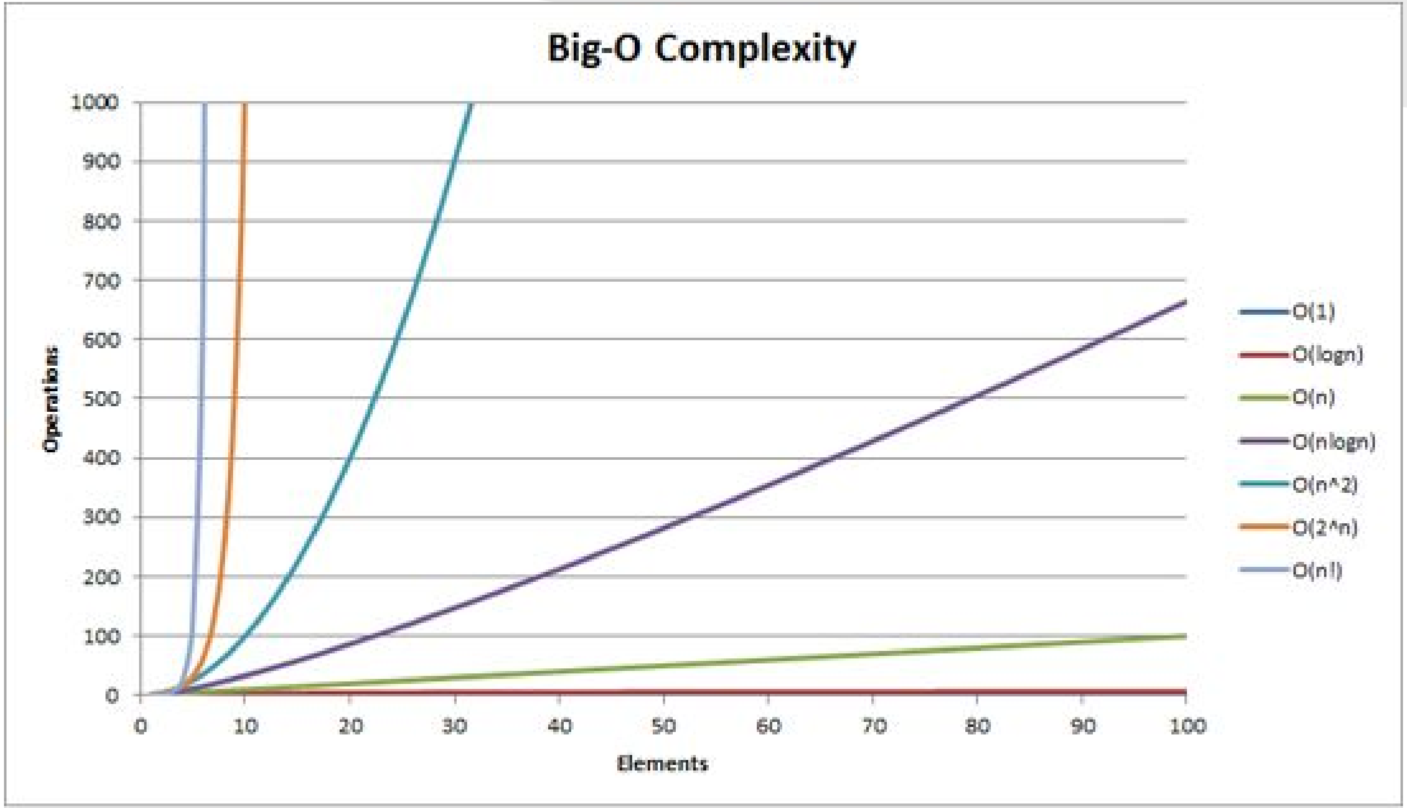
If I have a task that takes the same amount of time no matter what my input is that's O(1) or constant. When I have a task that maps one to one with the size of my data, that's O(n). This is the Big O for a Linear search. If we have 1,000 items then we may have 1,000 operations in a worst case.

Python Methodologies for Data Sciences

Here are some of the Big O complexities graphed out for us.

The algorithms we've been dealing with are fairly efficient when you look here. Our Binary Search has Big O -> O (log n); it's better than linear as we have shown.

A Computer Scientist's nightmare is O(n!). That's what's called combinatorial explosion, and most of these problems are unsolved because they are what's called "intractable." That's geek for "too big to handle."

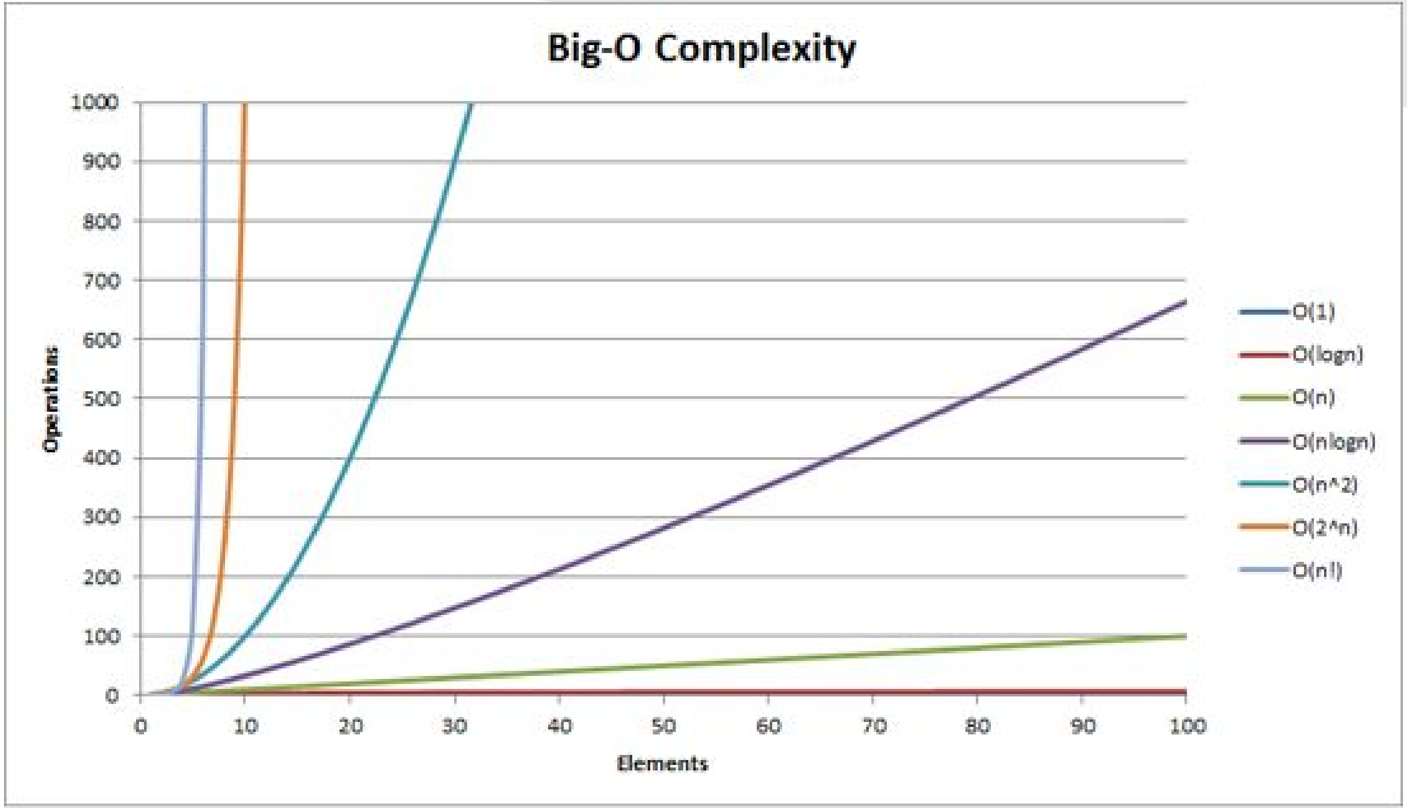


Python Methodologies for Data Sciences

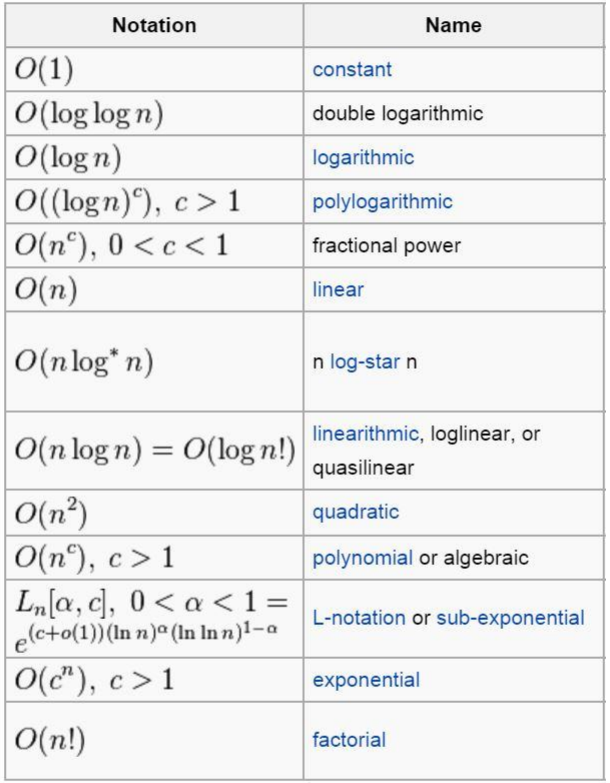
Even with small inputs an algorithm with Big O of O(n!) gets out of control.

There is a famous computer science problem called the Travelling Salesman Problem or TSP. TSP has Big O of O (n!).

These problems are not unsolvable because theory tells us they cannot be solved (there are things that computers can't solve...), this is unsolvable because there is not enough memory or time in the universe to do so. That's a big, scary problem, yeah?



Python Methodologies for Data Sciences



Notation	Name
$O(1)$	constant
$O(\log \log n)$	double logarithmic
$O(\log n)$	logarithmic
$O((\log n)^c),\ c > 1$	polylogarithmic
$O(n^c),\ 0 < c < 1$	fractional power
$O(n)$	linear
$O(n \log^* n)$	n log-star n
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or quasilinear
$O(n^2)$	quadratic
$O(n^c),\ c > 1$	polynomial or algebraic
$L_n[\alpha, c],\ 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation or sub-exponential
$O(c^n),\ c > 1$	exponential
$O(n!)$	factorial

Suffice it to say that Big O is important in that we want to make sure that when we use computers and algorithms to solve problems that they are efficient. This is the way we analyze algorithms in Computer Science.

There are other ways to examine problems and algorithms but they deal with a field called complexity. In complexity studies we look at the constraints of memory (space) and how many operations we need (time) and examine things that way.

Want a million dollars? Solve the P vs NP problem at the Clay Mathematics Institute, it’s the biggest unsolved problem in complexity and in all of computer science.

https://en.wikipedia.org/wiki/Big_O_notation

Python Methodologies for Data Sciences

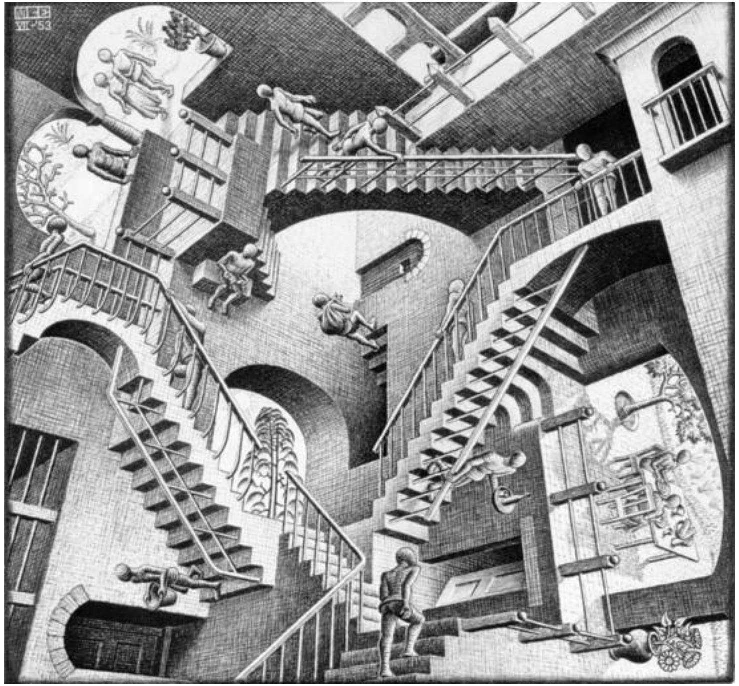
Okay: Sorting, searching, Big O; you have a bunch to soak in.

Take a break for a while and then come back for the mind bending fun that is...

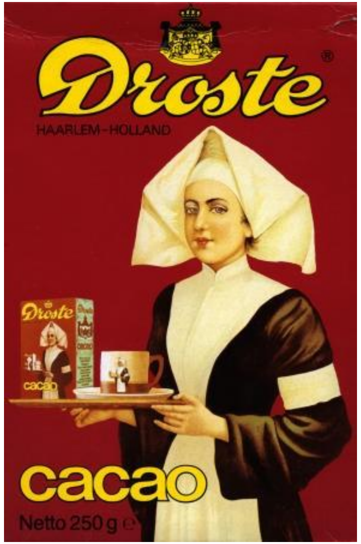
Recursion!

Recursion

Python Methodologies for Data Sciences



Python Methodologies for Data Sciences



Again, I know I sound like a broken record, but people make a bigger deal out of recursion than they have to.

Recursion, in computer science, refers to a function that, in the body of said function, refers to and calls itself. It's *self-referential*. If you study math or know who Bertrand Russell is you know how much trouble that can get you into, but we will always give our functions a way to get out.

The two main things we have to do when we learn about recursion are:

Understand how functions work (we're taking advantage of this)

Learn to spot problems that can be broken down into the same operation applied over a list in a "self-similiar" way.

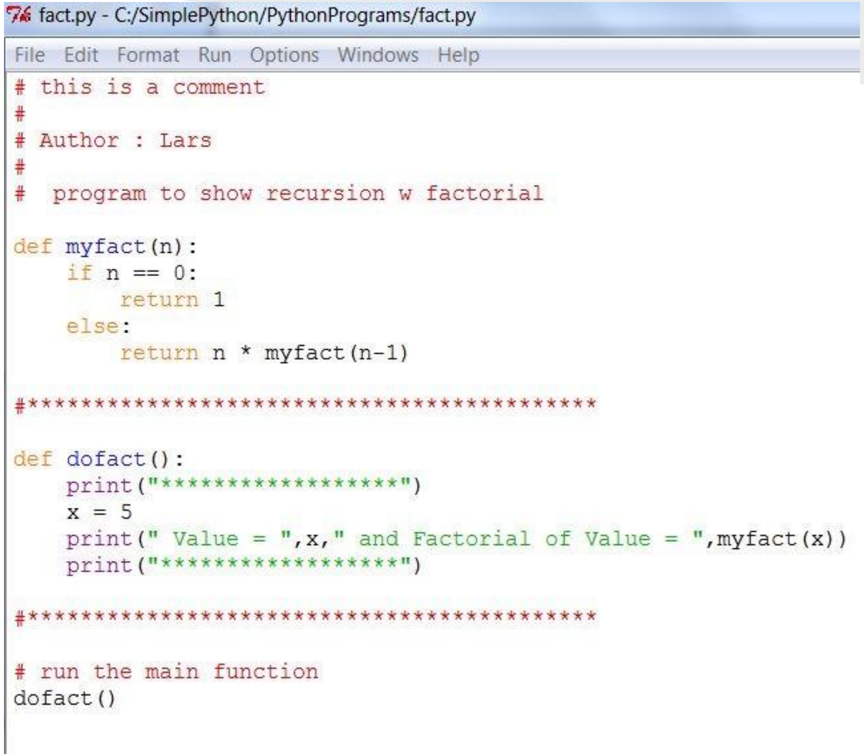
This sounds nuts, huh? Let's look at an example.

Python Methodologies for Data Sciences

To the right I have a program that figures out factorial values (4! = 4 * 3 * 2 * 1). Look in the function myfact(). In the last line, the function refers to itself and calls itself with the original parameter minus 1.

At first this seems weird, but it's not. The function that's called is just a fresh copy of the same function you are in. That's all. You create a new function run in memory that returns to the previous function when it's done. That's it.

Eventually you will get to the point where n == 0 and does not self refer. Then, all the values will cascade down as they return their values.


fact.py - C:/SimplePython/PythonPrograms/fact.py
File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars
#
#  program to show recursion w factorial

def myfact(n):
    if n == 0:
        return 1
    else:
        return n * myfact(n-1)

#*******************************************

def dofact():
    print("******************")
    x = 5
    print(" Value = ",x," and Factorial of Value = ",myfact(x))
    print("******************")

#*******************************************

# run the main function
dofact()
```

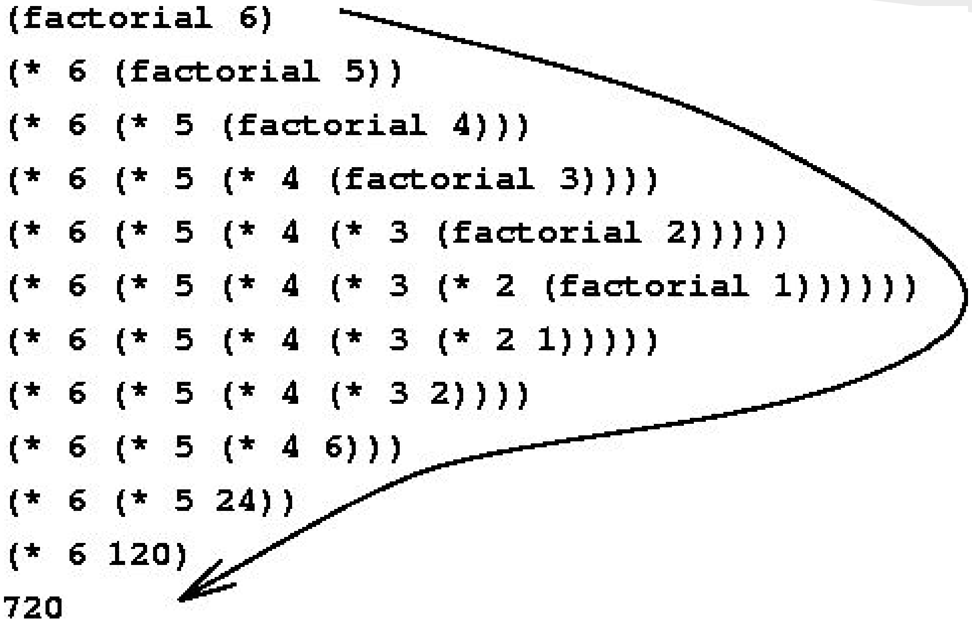
Python Methodologies for Data Sciences

Look at this diagram of factorial for 6. This is a good way to think about a recursive operation.

All we are doing is extracting the top number and then running the procedure on the smaller list of numbers left behind.

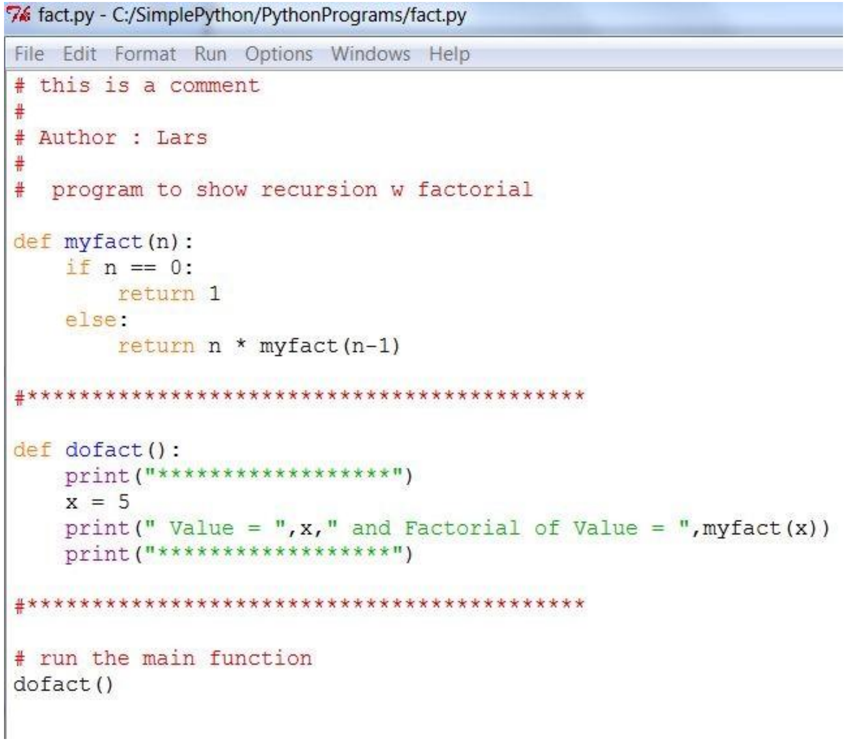
Once we get to the 1 we begin to multiply as we go back to the original function call.

This all makes sense, as factorial can be defined as “n multiplied by the factorial of n - 1.”



```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

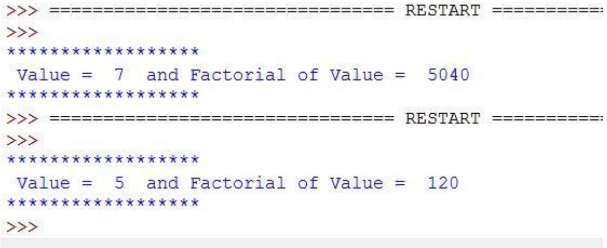
Python Methodologies for Data Sciences


fact.py - C:/SimplePython/PythonPrograms/fact.py
File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars
#
#  program to show recursion w factorial

def myfact(n):
    if n == 0:
        return 1
    else:
        return n * myfact(n-1)

#*******************************************

def dofact():
    print("******************")
    x = 5
    print(" Value = ",x," and Factorial of Value = ",myfact(x))
    print("******************")

#*******************************************

# run the main function
dofact()
```

Here’s the results. Recursion is something that takes time to get and seeing the answers doesn’t always help. Recursion works because of the behind the scenes. It works because it essentially does the following:

1. Make the list smaller
2. Run the operation again (the function call)
3. Repeat until list is done (tail condition)
4. As the functions all return your answer is crafted.



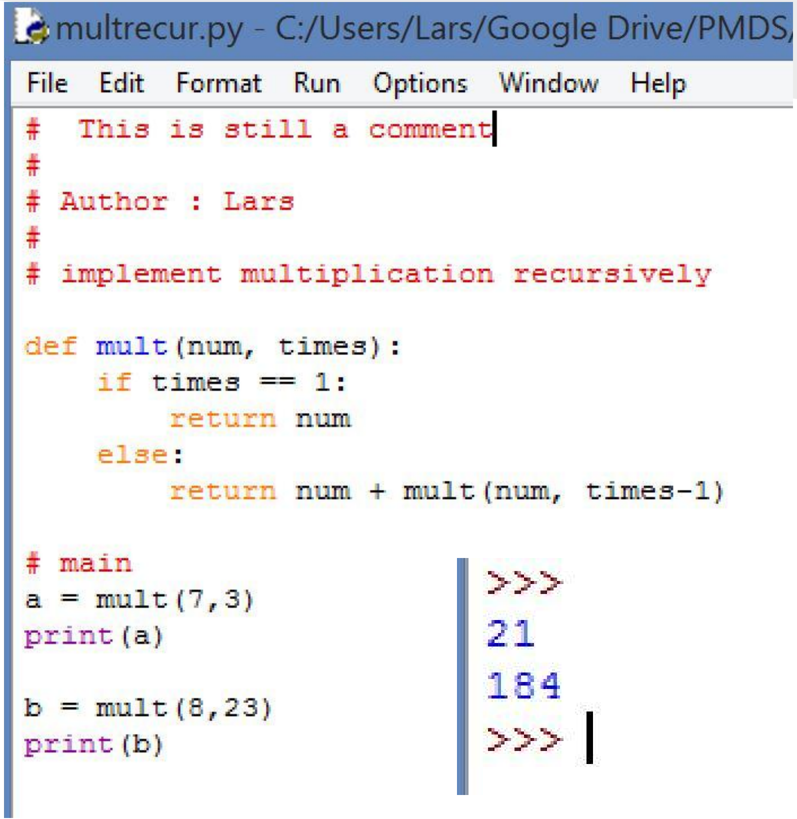
```
>>> ================================ RESTART ==========
>>>
******************
 Value =  7  and Factorial of Value =  5040
******************
>>> ================================ RESTART ==========
>>>
******************
 Value =  5  and Factorial of Value =  120
******************
>>>
```

Python Methodologies for Data Sciences

The Secret Sauce

In CS we almost always teach recursion with Factorial as an example (some use Fibonacci) and I do understand why (I mean, I just did it), but then most people stop there. CS teachers do not realize that most people are not familiar with factorials and the whole idea of teaching recursion should be showing someone a NEW way of doing something they are already used to doing. Enter multiplication.

I use multiplication because you have been intimate with this operation since you were in third grade. That, and I think you need to see more than one example. Look to the right and see how I implement multiplication with recursion.



```
#  This is still a comment
#
# Author : Lars
#
# implement multiplication recursively

def mult(num, times):
    if times == 1:
        return num
    else:
        return num + mult(num, times-1)

# main
a = mult(7,3)
print(a)

b = mult(8,23)
print(b)
```

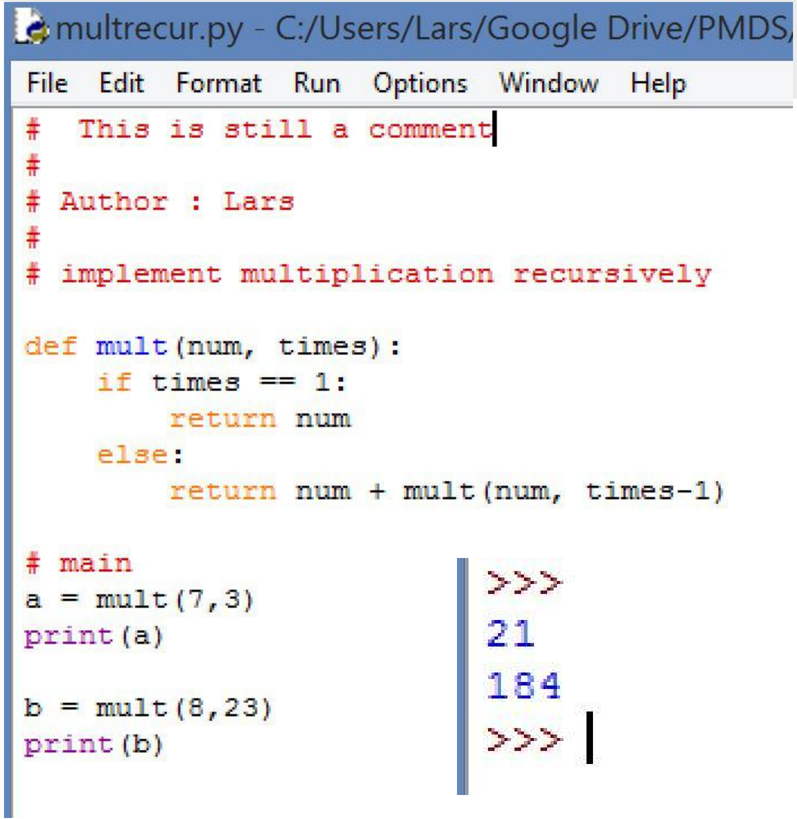
```
>>>
21
184
>>>
```

The secret sauce cont.

Multiplication is just a series of additions, remember? When I call 7,3 I think of 7+7+7. That's perfect for recursion. Save the one seven and send the smaller list (7+7 in this instance) to the same function (with a mult(7,2)).

When you call mult(7,1) then times will be 1. This is your tail condition. You'll return 7 to a function that adds 7 to it and then that function returns 14 to a function that adds 7 to it and returns 21 to the original call.

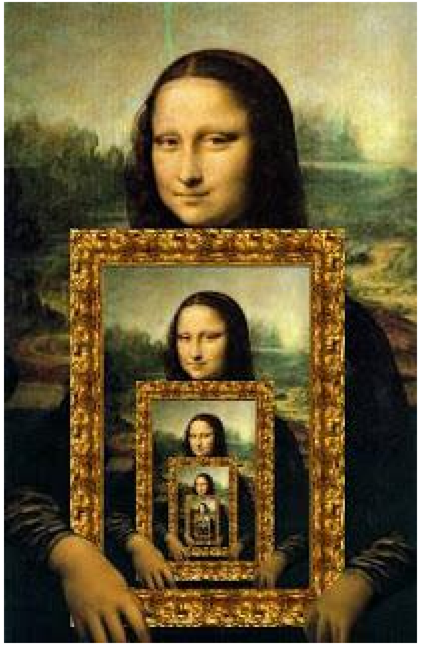
Look at this carefully and understand how it works before you move on. That sounds like some silly teacher thing to say, I know. I want you to do it though because when you get it, when you have that "eureka" recursion moment and understand what it's about, it's awesome and I want you to get there.



```
#  This is still a comment
#
# Author : Lars
#
# implement multiplication recursively

def mult(num, times):
    if times == 1:
        return num
    else:
        return num + mult(num, times-1)

# main
a = mult(7,3)
print(a)

b = mult(8,23)
print(b)
```

```
>>>
21
184
>>>
```

Python Methodologies for Data Sciences



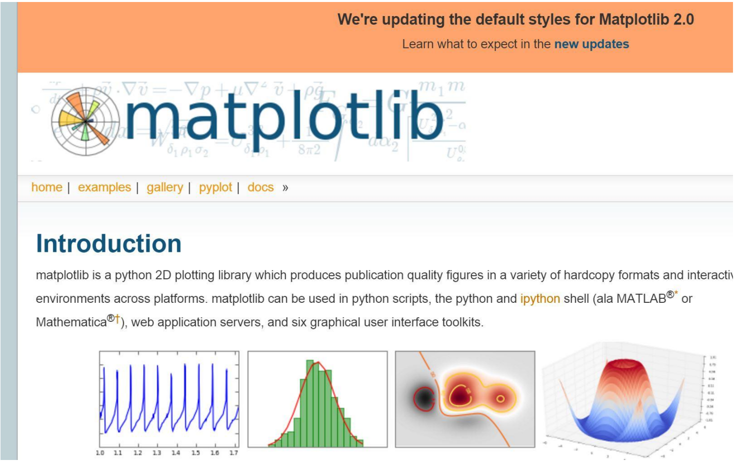
You will not be left hanging. Recursion is something that is best learned by seeing it done. I will be doing an entire video dedicated to just this and we will be live coding recursive functions during our October 28th in-class meeting.

Make sure you are there!

Python Methodologies for Data Sciences

Matplotlib and Simple Graphs

Python Methodologies for Data Sciences



We're updating the default styles for Matplotlib 2.0

Learn what to expect in the new updates

home | examples | gallery | pyplot | docs »

Introduction

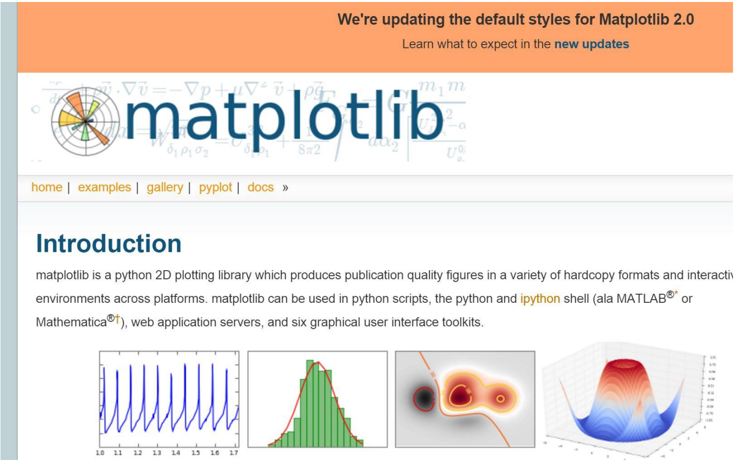
matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactiv environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®†), web application servers, and six graphical user interface toolkits.

http://matplotlib.org/

Now we're getting somewhere. We are going to venture into the world of third party libraries this week. We are going to load and use a package called "Matplotlib." It's a series of functions, classes and methods that allow us to craft visualizations with our data. It mimics much of the functionality of MATLAB.

You will find that programmers are a cheap lot (me included, FREEEEE). Why pay a lot of money for software when we're able to create it for ourselves? Because of this it's often a good idea to see if there is a freeware solution to the things you'd like to use. It also allows for customization (you do know Python now) that is not available when you use commercial proprietary software.

Python Methodologies for Data Sciences



We're updating the default styles for Matplotlib 2.0

Learn what to expect in the **new updates**

home | examples | gallery | pyplot | docs »

Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactiv environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®†), web application servers, and six graphical user interface toolkits.

You can try to load matplotlib on your system before we meet, but be wary of 32 vs 64 bit systems and make sure you download a copy that works with your version of Python!

Now, we have one issue. Sometimes loading these third party libraries can be a pain for beginning programmers.

Packages often rely on you having loaded other packages beforehand. These requirements are called "dependencies" and you may as well get used to them now.

Matplotlib has a few, not the least of which is NUMPY, one of the most popular Python libraries for math and science programming (so it's good you have it.) so we have to load this library too.

I will discuss the process of loading matplotlib when we meet on October 28th so make sure you're there.

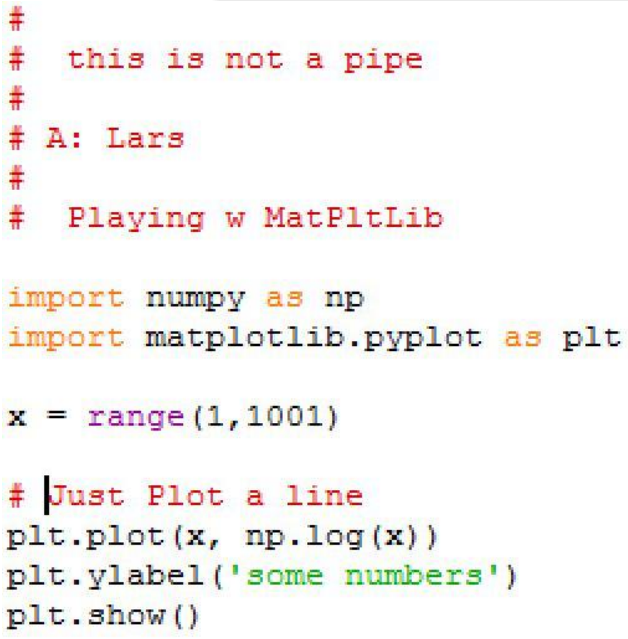
Python Methodologies for Data Sciences

As you may have discovered, often the hardest part about using third party libraries is getting them installed and working properly. Once that's done they are often easy to use though.

Look to the right and you can see a simple program to plot the log of n as n goes from 1 to 1000...

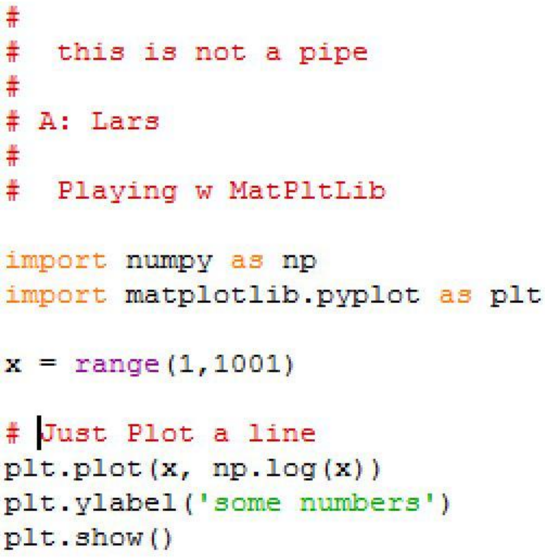
Notice that we import differently here. It's not because of matplotlib or NUMPY, it's to show you some new ways to use import.

The first import is just importing NUMPY but telling Python we want to refer to it as "np". That's all. The second import is just a way to import a sub library. A sub library is a module that resides inside a directory, in this case the top level directory is called matplotlib.



```
#
#  this is not a pipe
#
# A: Lars
#
#  Playing w MatPltLib

import numpy as np
import matplotlib.pyplot as plt

x = range(1,1001)

# Just Plot a line
plt.plot(x, np.log(x))
plt.ylabel('some numbers')
plt.show()
```

Python Methodologies for Data Sciences

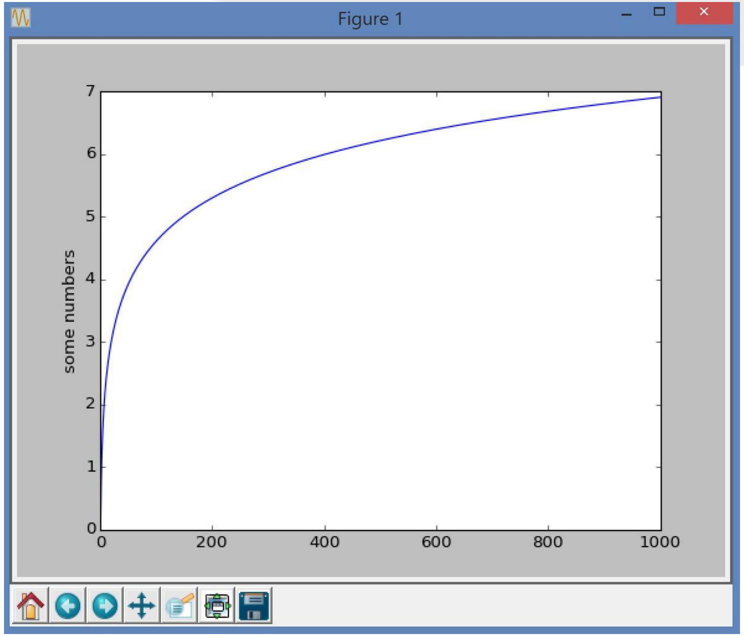


```
#
#  this is not a pipe
#
# A: Lars
#
#  Playing w MatPltLib

import numpy as np
import matplotlib.pyplot as plt

x = range(1,1001)

# Just Plot a line
plt.plot(x, np.log(x))
plt.ylabel('some numbers')
plt.show()
```

When we run the code we get a plot of our data!

Check out the URL under the graph for examples of all the graphs you can make with matplotlib.

After that take some time with the pyplot tutorial. Now that you understand Python you can avail yourself of these kinds of resources!



http://matplotlib.org/gallery.html

http://matplotlib.org/users/pyplot_tutorial.html

The Orphans

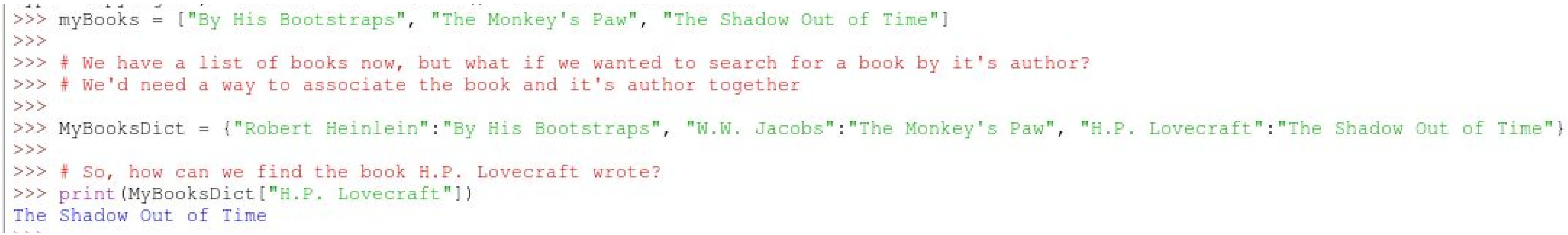
Dictionaries

In addition to the previously seen Lists, Python has another container built into it: the Dictionary.

With Lists, the only ways to access an item were to iterate across the entire list, or remember *exactly* at what spot the item was at. You were only able to *index* your list with integer numbers.

But with Dictionaries, you can index with ***anything***. A dict is what we call an “associative list” or a “key-value store.” It works just list a list, but associates one value with another, rather than a location with its value.

Try creating some code similar to the code below. What happens if we try to access the author by book? Or the pair by position? (Hint: Bad things)

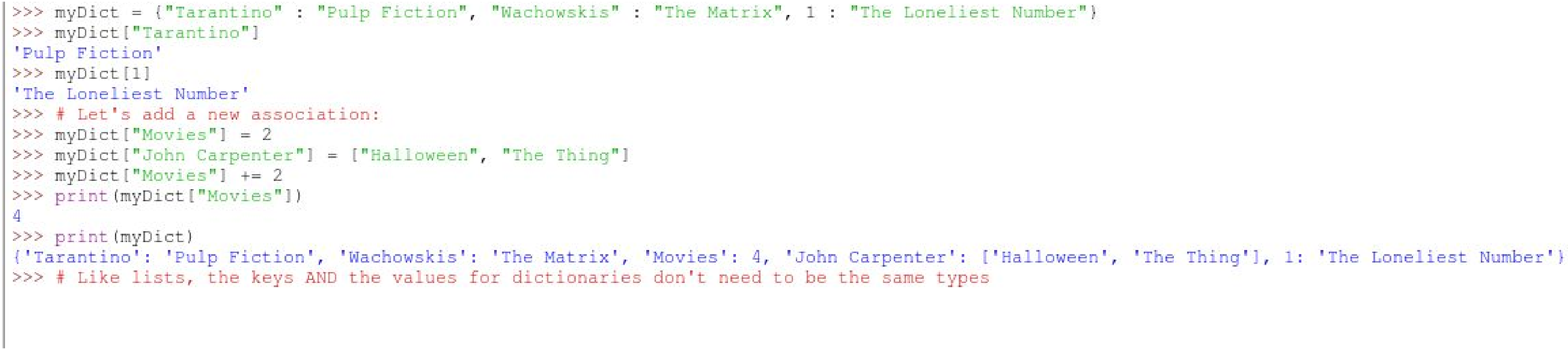


```
>>> myBooks = ["By His Bootstraps", "The Monkey's Paw", "The Shadow Out of Time"]
>>>
>>> # We have a list of books now, but what if we wanted to search for a book by it's author?
>>> # We'd need a way to associate the book and it's author together
>>>
>>> MyBooksDict = {"Robert Heinlein":"By His Bootstraps", "W.W. Jacobs":"The Monkey's Paw", "H.P. Lovecraft":"The Shadow Out of Time"}
>>>
>>> # So, how can we find the book H.P. Lovecraft wrote?
>>> print(MyBooksDict["H.P. Lovecraft"])
The Shadow Out of Time
```

Python Methodologies for Data Sciences

Dictionaries operate as a *key-value store*, where the *key* is what you can use to access the *value*.

The other great thing about dicts is that we can create associations on the fly:

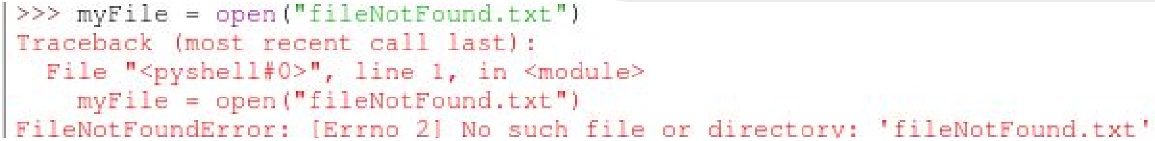


```
>>> myDict = {"Tarantino" : "Pulp Fiction", "Wachowskis" : "The Matrix", 1 : "The Loneliest Number"}
>>> myDict["Tarantino"]
'Pulp Fiction'
>>> myDict[1]
'The Loneliest Number'
>>> # Let's add a new association:
>>> myDict["Movies"] = 2
>>> myDict["John Carpenter"] = ["Halloween", "The Thing"]
>>> myDict["Movies"] += 2
>>> print(myDict["Movies"])
4
>>> print(myDict)
{'Tarantino': 'Pulp Fiction', 'Wachowskis': 'The Matrix', 'Movies': 4, 'John Carpenter': ['Halloween', 'The Thing'], 1: 'The Loneliest Number'}
>>> # Like lists, the keys AND the values for dictionaries don't need to be the same types
```

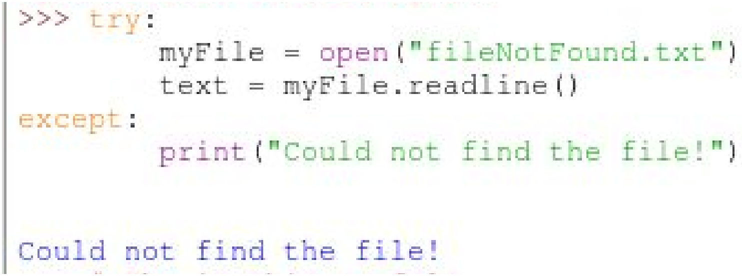
Exception Handling

Sometimes, errors are bound to happen, and we can’t predict everything. When something goes wrong in our program, we usually don’t want the whole thing to crash violently; we should at least let the user (or the programmer) know what went wrong and exit gracefully!

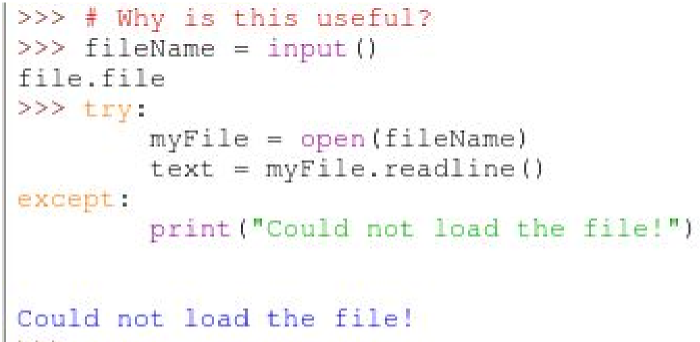
To solve these problems, we have exception handling and try-except blocks.



```
>>> myFile = open("fileNotFound.txt")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    myFile = open("fileNotFound.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'fileNotFound.txt'
```



```
>>> try:
	myFile = open("fileNotFound.txt")
	text = myFile.readline()
except:
	print("Could not find the file!")


Could not find the file!
```



```
>>> # Why is this useful?
>>> fileName = input()
file.file
>>> try:
	myFile = open(fileName)
	text = myFile.readline()
except:
	print("Could not load the file!")


Could not load the file!
```

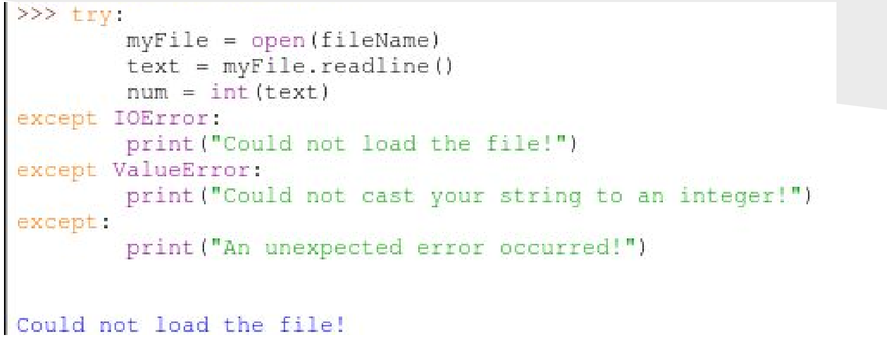
Python Methodologies for Data Sciences

In except blocks, we can catch specific errors to deal with each one separately. We can give our errors names like variables using the *as* keyword, so we can print out our error messages.

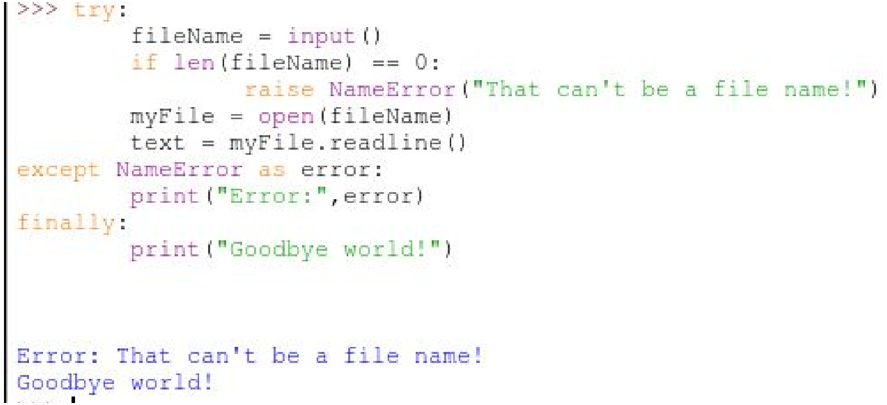
An except block that does not specify an error will catch *any* error (so this except block should always go last, as Python checks errors in the order you specify). This is our *default* case.

For a list of possible errors to catch, see:
https://docs.python.org/2/library/exceptions.html

And note that you can raise exceptions yourself too!



```
>>> try:
        myFile = open(fileName)
        text = myFile.readline()
        num = int(text)
except IOError:
        print("Could not load the file!")
except ValueError:
        print("Could not cast your string to an integer!")
except:
        print("An unexpected error occurred!")


Could not load the file!
```



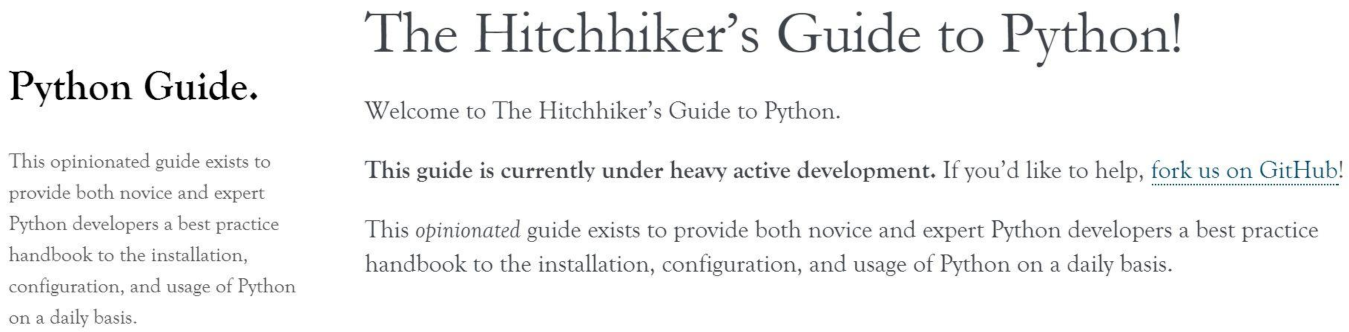
```
>>> try:
        fileName = input()
        if len(fileName) == 0:
                raise NameError("That can't be a file name!")
        myFile = open(fileName)
        text = myFile.readline()
except NameError as error:
        print("Error:",error)
finally:
        print("Goodbye world!")



Error: That can't be a file name!
Goodbye world!
```

To sum things up

This Unit was a tour of some Computer Science concepts that I wanted you to be aware of. They will serve you well as continue your journey down the road to becoming a productive programmer. I do not expect you to become experts in any of these topics. I want you to be exposed to them, play around with them and to continue to explore the ones that you find interesting or that you think you can use in your programs (Dictionaries are used quite often in Python).

Sorting and searching, Big O, recursion, using third party modules and being aware of dictionaries and exception handling are all powerful tools for you to have in your Python toolbox. As you can see we are slowly but surely leaving the world of learning the syntax of Python and getting more into the realm of learning HOW to use the Python we've learned. This is higher level stuff. It's learning new discrete problem solving techniques and then merely using Python to implement them. It's a big time step.

Other cool resources



Python Guide.

This opinionated guide exists to provide both novice and expert Python developers a best practice handbook to the installation, configuration, and usage of Python on a daily basis.

The Hitchhiker's Guide to Python!

Welcome to The Hitchhiker's Guide to Python.

This guide is currently under heavy active development. If you'd like to help, fork us on GitHub!

This *opinionated* guide exists to provide both novice and expert Python developers a best practice handbook to the installation, configuration, and usage of Python on a daily basis.

This is not a tutorial, you're done with that. this is about writing good code, structure, scraping the web, some next level stuff. Check it out.

http://docs.python-guide.org/en/latest/

I know, I know...

This is the section where I apologize for not getting to something that was promised in the Syllabus. This week we finally got to Dictionaries and Exception handling, but at the expense of Object graphics and Event Driven programming.

I will do some graphics in Unit 5 (it makes sense, you'll see) and we'll look at Event Driven programming in Unit 6 when we look at some Data Structures, Design patterns and higher level stuff.

Python Methodologies for Data Sciences

A few simple exercises

Zelle -

Page 463 - Do the Palindrome problem, # 3

Write the Anagram program from page 437

Code up the Selection Sort from page 444