INTRODUCTION TO PYTHON

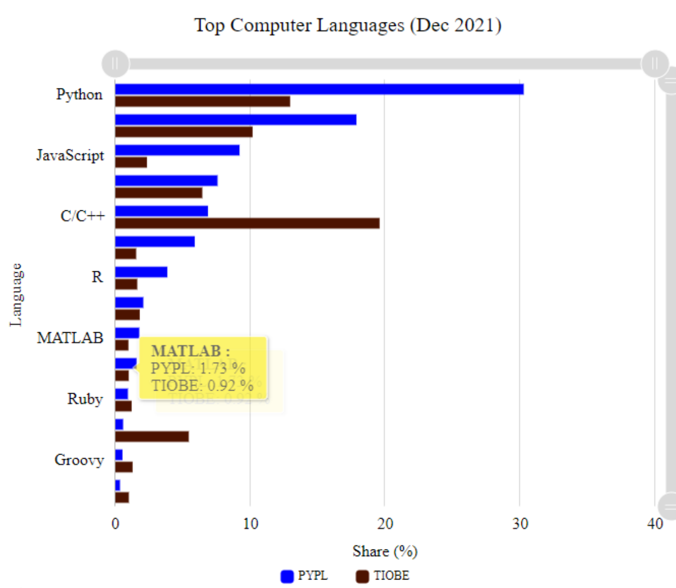
Dr. Yaroslav Rosokha
Associate Professor of Economics
E-mail: yrosokha@purdue.edu

Research
Experimental Economics, Computational Economics
Behavioral Economics & Operations Management

S2 **PYTHON**

- Why python?
  - **easy** to learn
  - open source (**free**!)
  - reasonably **fast**
  - one of the two most **popular** for data science (the other one: **R**)
- "Glue" language
  - general-purpose programming
  - analytical and quantitative computing
  - web applications
  - many packages
  - popular across many domains

https://statisticstimes.com/tech/top-computer-languages.php



S3 PYTHON OVERVIEW

- Interpreted
  - Processed at runtime by interpreter
  - Do not need to compile your program

- Object-oriented
  - Data and function are “bundled together” into “objects”
  - Objects have attributes (data) and methods (functions)
  - **Everything in Python is an object**, and almost everything has attributes and methods.

- “Beginner’s Language”
  - Simple structure and clearly defined syntax
  - Easy to read: uses keywords (in English) instead of punctuation
  - Interactive (in particular IPython/Jupyter implementation)

S4 PYTHON SETUP

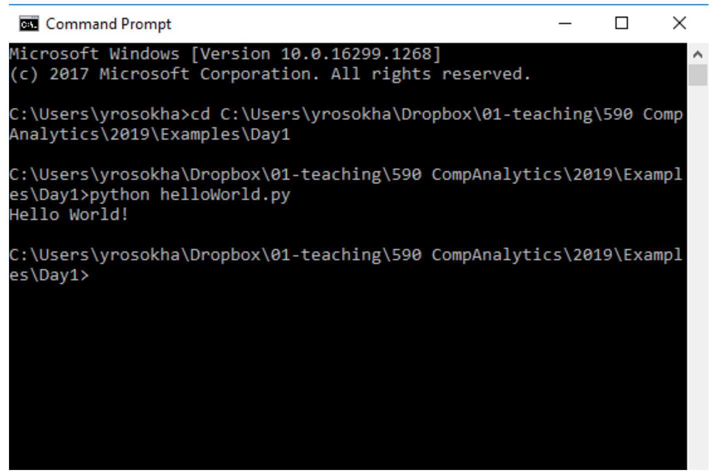
- **Anaconda Distribution**
  - Download: https://www.anaconda.com/products/individual
  - Install Python 3.9 64-bit version
    - Includes over 100 of most popular packages for data science

- **Jupyter Notebooks**
  - Included in the Anaconda distribution
  - We will use Jupyter Notebooks for many in-class examples
  - Mix of text, code, and code output
    - Different from running Python using command line
    - Structured, self-explanatory worksheets
    - Collection of Jupyter Notebooks (here)

S5 RUNNING PYTHON DIRECTLY (WITHOUT JUPYTER)

- Create **helloWorld.py** file in your working directory

helloWorld.py
```
print("Hello World!")

```



- To run this file
    - Open Command Prompt
    - Navigate to your working directory
        - cd <Directory Name>
        - dir – list files in the current directory
    - Type ‘python HelloWorld.py’

S6 RUNNING PYTHON WITH JUPYTER (LAB OR NOTEBOOK)

- Browser-based interactive development environment
    - Code
    - Equations
    - Visualization
    - Text
    - Output

- **Ex:**
    - **hello_world.ipynb**
    - **getting_started.ipynb**

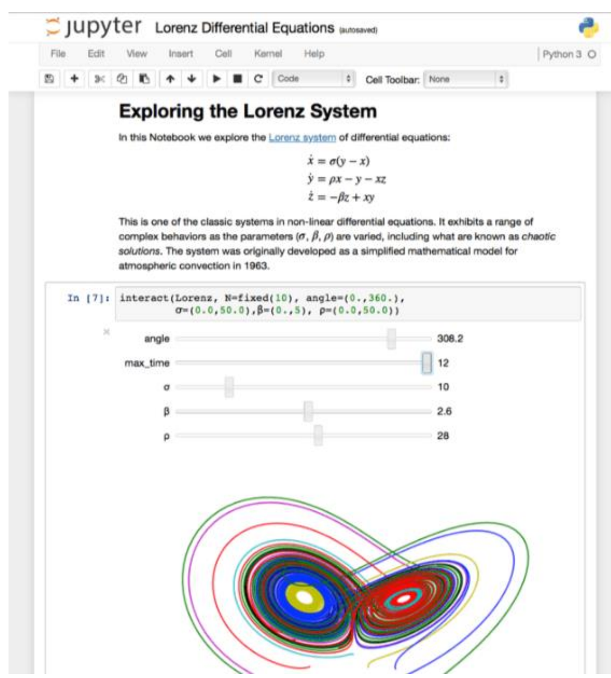


Image source: jupyter.org

S7 PYTHON SYNTAX

- Identifiers
  - Names of variables, functions, classes
  - Unlimited in length
  - Case sensitive

- Assignments
  - Used to bind names to values and to modify attributes or items of mutable objects
  - Example:

```
myVar1 = 'Hello World!'
myVar2 = 42
```

S8 PYTHON SYNTAX

- Keywords
  - Reserved words
  - Cannot be used as ordinary identifiers
  - Must be spelled exactly

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

S9 PYTHON SYNTAX: OPERATIONS

- x + y   sum of x and y
- x - y   difference of x and y
- x * y   product of x and y
- x / y   quotient of x and y
- x // y   (floored) quotient of x and y
- x % y   remainder of x / y
- -x   x negated
- +x   x unchanged
- abs(x)   absolute value of x
- int(x)   x converted to integer
- long(x)   x converted to long integer
- float(x)   x converted to floating point
- complex(re,im) a complex number with real part re, imaginary part im. im defaults to zero.
- c.conjugate() conjugate of the complex number c. (Identity on real numbers
- divmod(x, y) the pair (x // y, x % y)
- pow(x, y)   x to the power y
- x ** y   x to the power y

BUILT-IN DATA STRUCTURES

S11 DEFINITION

- **Data structure** refers to the format in which data is organized, managed, and stored
  - While there are few common data structures, it is also important to know how to design own data structure for a specific purpose

- What are data structures used for?
  - Input
  - Processing
  - Storing
  - Retrieving

- **Abstract Data Types** (Abstract Data Structures) refers to the logical form of the data structure. In other words, data structure is the implementation of the abstract type

S12 BUILT-IN DATA STRUCTURES IN PYTHON

- Primitive Data Structures
  - plain integers (usually 32 bits of precision; sys.maxint)
  - long integers (unlimited precision)
  - floating point numbers (precision depends on the machine; sys.float_info)
  - complex numbers
  - booleans (are a subtype of plain integers)
    - 0, '', [], (), {}, and None are false; everything else is true
  - strings (see below)
- Python Built-in Data Structures (Objects)
  - tuples

```
myTuple1 = (1 , 2, 3)
myTuple2 = (4, "Some Text")
```

  - strings

```
myString1 = "value 0"
myString2 = "key2"
```

  - lists

```
myList1 = [1 , 2, 3]
myList2 = [4, myVar1, myList1, "Some Text"]
```

  - dictionaries

```
myDictionary1 = {0: "value 0", "a": "letter a"}
myDictionary2 = {4: myVar1, myString2: myList1}
```

S13 PYTHON DATA STRUCTURES: LISTS

- Collection of objects
  - A comma-delimited sequence within square brackets
  - You can extract values by an index
    - The first element starts at index 0

```
myList1 = [1 , 2, 3]
myList2 = [4, myList1, “Text1”, “Text2”]
```

- Accessing Values
  - Use the square brackets
  - Can slice multiple entries

```
myList2[1]
myList2[1:3]
```

Ex: lists.ipynb

- Basic Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 3	['Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: …	…	Iteration

- Methods
  - .append(obj) .extend(obj)
  - .index(obj) .count(obj)
  - .insert(index, obj) .pop(index)
  - .reverse() .sort()
  - For more details go to: https://docs.python.org/3.6/tutorial/datastructures.html

S14 PYTHON DATA STRUCTURES: STRINGS

- Collection of characters
  - Enclose characters in quotes: ‘ ’ or “ ”
  - You can extract string values by an index
    - The first element starts at index 0

```
myString1 = “Hello World!”
myString2 = “Programming in Python!”
```

- Accessing Values
  - Use the square brackets
  - Can slice multiple entries

```
print(myString1[2])
print(myString2[4:12])
```

- **Ex: strings.ipynb, string_formatting.ipynb**

- Basic Operations

Operator	Description
+	Concatenation
*	Repetition
[]	Accessing
[:]	Slicing
in	Membership
not in	Membership

- Methods
  - .capitalize(), .center(width, fillchar), .split(str), .count(str, beg,end), .lower(), .upper()
  - For more go to: https://docs.python.org/3.6/tutorial/introduction.html#strings

S15 PYTHON DATA STRUCTURES: DICTIONARIES

- Collection of objects indexed by keys (unordered set of key : value pairs)
  - Each key is separated by object/value by :

```
myDictionary1 = {0: "value 0", "a": "letter a"}
myDictionary2 = {4: myVar1, "b": myDictionary1}
```

- Accessing Values
  - Use the square brackets along with the key

```
print(myDictionary1[0])
print(myDictionary2["b"])
```

- Updating Values
  - You can add a new entry

```
myDictionary1['xyz'] = "Hello Dictionary!"
print(myDictionary1)
```

- Basic Functions and Methods
  - cmp(dict1,dict2), len(dict)
  - .clear(), .keys(), .values(), .items()
  - For more details go to: https://docs.python.org/3.6/tutorial/datastructures.html#dictionaries

- Remember
  - You can not have duplicate keys in a dictionary.
  - Assigning a value to an existing key will wipe out the old value

- **Ex: dictionaries.ipynb**

S16 PYTHON DATA STRUCTURES: TUPLES

- Collection of immutable objects
  - A comma-delimited sequence within parentheses
  - You can extract values by an index
  - The first element starts at index 0

```
myTuple1 = (1 , 2, 3)
myTuple2 = (4, myTuple1, "Text1", "Text2")
```

- Accessing Value
  - Use the square brackets
  - Can slice multiple entries

```
print(myTuple1[1])
print(myTuple2[1:3])
```

- Basic Operations
  - len(tuple)
  - max (tuple) min(tuple)
  - For more details go to: http://www.tutorialspoint.com/python/python_tuples.htm

- Remember
  - Tuples are immutable which means you cannot update or change the values of tuple elements

- **Ex: tuples_and_mutability.ipynb**

S17 COMPARING IN OPERATOR FOR LISTS, DICTIONARIES

- Write a program to compare the speed of in operator for lists, dictionaries
- **Ex: comparing_in_operator.ipynb**

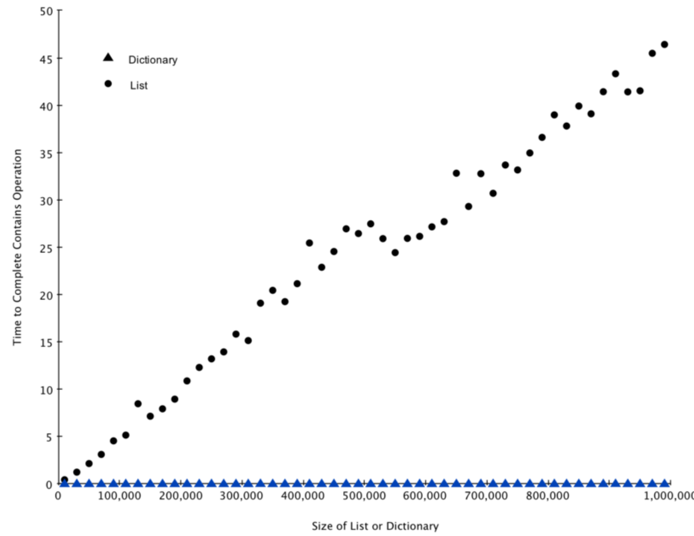


Image source: Source: Problem Solving with Algorithms and Data Structures by Brad Miller and David Ranum

CONTROL FLOW

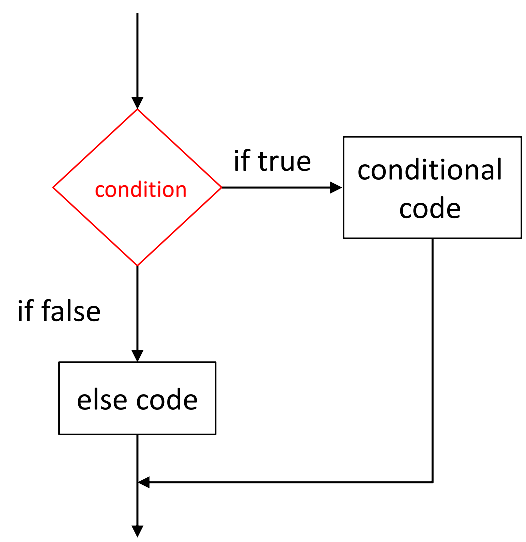
S19 CONTROL FLOW

- What if you wanted control the flow of the program?
  - **Repeat** a part of the program
  - Make a decision whether to **execute** or **skip** part of program if some **condition** is met
- There are three main control flow statements in Python
  - **for**
  - **while**
  - **if**
- Code blocks
  - Defined by indentation
  - Indentation starts a block
  - Indentation doesn't need to be specific number of spaces but needs to be consistent
  - The first line that is not indented is outside the block
  - There are no explicit braces, brackets, or keywords
  - Benefit: readability

S20 PYTHON SYNTAX: DECISION MAKING WITH IF/ELSE

- if statements are used to make decisions within the program
  - if/else
  - if/elif/else
  - Example:

```
n=5
if n<20:
        n=n+3
        print(n)
else:
        print(n)
```



- **Ex: if_else.ipynb**

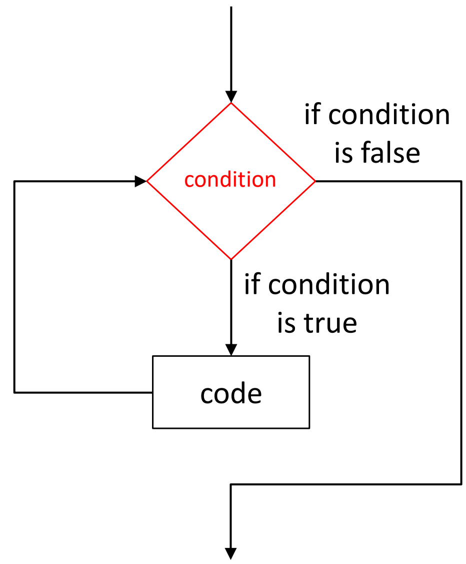
http://interactivepython.org/runestone/static/StudentCSP/CSPIntroDecisions/if.html

S21 Python Syntax: while loops

- Loop is used to repeatedly execute the code block that follows
  - Example:

```
n=5
while n<20:
        n=n+3
        print(n)
```

- **Ex: loops.ipynb**



http://interactivepython.org/runestone/static/StudentCSP/CSPWhileAndForLoops/forAndWhile.html

S22 Python Syntax: for loops

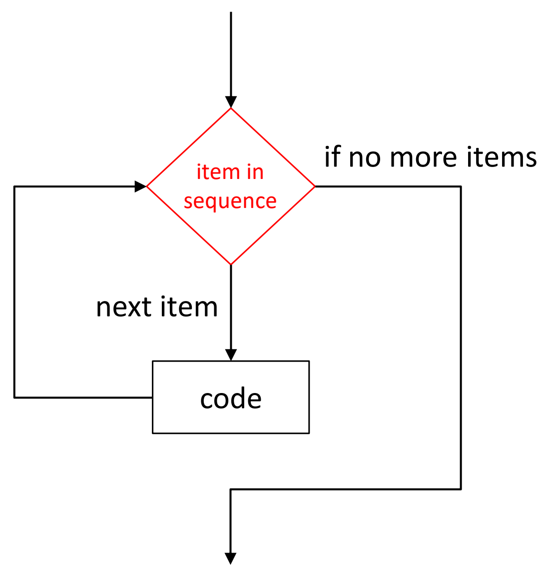
- Loop is used to repeatedly execute the code block that follows
  - Examples:

```
for n in [0,1,2]:
    print(n)
```

```
for x in myList2:
    print(x*2)
```

```
for x in myDictionary2:
    print(x)
```

- **Ex: loops.ipynb**



http://interactivepython.org/runestone/static/StudentCSP/CSPWhileAndForLoops/forAndWhile.html

Functions

S24 Python Syntax: Functions

- Functions
  - a block of code that is used to perform an action
  - provide modularity and code (re-)usability
  - the code within the function is indented

```
def fib(n): #Fibonacci sequence up to (and including) integer n
    a = 0 #starts with 0 and 1
    b = 1
    while a <= n:
        print(a) #print number to screen
        tempVar=a
        a=b
        b = tempVar+a
```

- **Ex: functions.ipynb**

S25 PYTHON SYNTAX: IMPORT

- You will want to (re-)use functions that you (and others) have written in other programs without copying each time
- Python enables this through import statement
- For example, I saved myTax() and kyTax() into a file called taxes.py
  - You can import this file (module) by calling import statement

```
import taxes #imports taxes as an object with all functions as attributes
taxes.myTax(10)
```

```
import taxes as tax #imports taxes as an object named tax with all functions as attributes
tax.myTax(10)
```

```
from taxes import myTax #imports a particular function from taxes
myTax(10)
```

```
from taxes import * #imports all functions from taxes
myTax(10)
```

- **Ex: import.ipynb**

S26 PUTTING IT ALL TOGETHER

- Suppose that we have a dictionary that stores customer purchase price information

```
paidInfo = {“Alice”:10.5, “Bob”:7.8, “Chris”:15.0}
```

- Suppose that we want to find how much tax each of them paid (at 7% tax rate). We can construct a function

```
def myTax(x):
    return  0.07*x
```

- **Ex: combined_example.ipynb**

- We can construct loops to access each individual object in the list/dictionary

```
for p in paidInfo.values():
    print(“price:“,p,”tax:”,myTax(p))
```

```
for k in paidInfo:
    print(“key=“,k)
    print(“tax:”,myTax(paidInfo[k]))
```

- **Question: What if we want to store that information in a separate list?**

Mutability

S28 **Mutability**

- List and dictionaries are **mutable** data structures
  - their contents can be altered (mutated) in memory after initialization
- Here's an example:

```
a = [42, 44]
b = a
b[0]=0.0
```

- What is a?
- **Ex: mutability_revisit.ipynb**

- The name b is bound to a and becomes just another **reference** to the list
  - Hence it has equal rights to make changes to that list
  - This is in fact the most sensible default behavior
  - It means that we pass only references to data, rather than making copies
- Why do we care?
  - Making copies is expensive in terms of both speed and memory

Data Input\Output and File Operations

S30 Python Data Input\Output: read(), readline(), and write()

- Reading and writing to files:
    - open(*filename*,*mode*) returns a file object
        - the first argument is a string containing file name
        - the second argument is a string describing the way in which the file will be used: 'r' – read only; 'w' – write only; 'a' – append; 'r+' – reading and writing

```
f = open("testFile1.txt", "w")
for x in range(10):
    f.write( str(x) + "\n")
f.close()
```

- **Ex: file_input_output.ipynb**

- You can read lines from a file as follows:

```
f = open("testFile1.txt", "r")
for l in f:
    print(l)
f.close()
```

- More information can be found here:
    - https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

- Note: this approach is the most general but requires more work when you are dealing with numbers, lists, dictionaries

S31 Working with Files and Folders: os and shutil

- **os** library has a number of functions to work with directories and files:
  - os.getcwd() is the method to get the current working directory
  - os.listdir() is the method to use to get a directory listing *[similar functionality is with os.scandir()]*
  - os.stat() provides information such as file size and the time of last modification
  - os.remove() – removes a file
  - os.rmdir() – removes an empty directory
- **Ex: os_shutil.ipynb**
- **shutil** library has a number of high-level operations for files and collections of files
  - shutil.copyfile(*src*, *dst*) -- copy the contents (no metadata) of the file named *src* to a file named *dst*
  - shutil.make_archive() -- create archived files
  - shutil.unpack_archive() – unpack archived file
  - shutil.rmtree() – delete a directory an all its contents

Scientific Computing in Python
NumPy, SciPy, Pandas

S33 SCIENTIFIC COMPUTING IN PYTHON: NUMPY

- NumPy
  - Python library for scientific computing
  - Used in
    - Academia
    - Finance
    - Industry
  - Pros
    - Fast
    - Stable
    - Good Documentation
- **Ex: numpy.ipynb**
- Fast array processing
  - Loops in Python carry significant overhead
    - C and Fortran is much faster because they carry data type information that can be used for optimization
    - Optimization can be carried out during compilation
  - NumPy is fast because it sends operations *in batches* to optimized C and Fortran code
- Very fast with *vectorized operations*
  - linear algebra routines
  - generating a vector of random numbers
  - applying a fixed function to an entire array

S34 NUMPY ARRAYS

- NumPy defines an array data type called a numpy.ndarray
  - NumPy arrays power a large proportion of the scientific Python ecosystem
- Examples of creating NumPy arrays

```
a = np.zeros(3)
z = np.empty((3,3))
z = np.linspace(2, 4, 5)
z = np.array([10, 20])
```

- NumPy arrays are different form Python lists in that data must be homogeneous
  - all elements of the same type
  - These types must be one of the data types (dtypes) provided by NumPy
    - http://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html

S35 Numpy Arrays: Accessing Data

- Suppose we need to work with a 2D array

```
z = np.array([[1, 2], [3, 4]])
```

- For 2D arrays the index syntax is as follows:
  - Indices are zero-based, to maintain compatibility with Python sequences
  - You can extract columns and rows as follows
    - z[*row number*,:]
    - z[:,*column number*]

- You can use NumPy arrays of integers to extract elements of other arrays

```
z = np.linspace(2, 4, 5)
indices = np.array((0, 2, 3))
```

  - What is z[indices]?

- You can use Numpy array of dtype bool to extract elements of other arrays

```
d = np.array([0, 1, 1, 0, 0], dtype=bool)
```

  - What is z[d]?

S36 Numpy Arrays: Algebraic Operations

- The algebraic operators +, -, *, / and ** all act elementwise on arrays

- Suppose

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
```

- What is a + b?
- What is a * b?
- What is a + 10?
- What is a * 10?

- The two dimensional arrays follow the same general rules
- Supose

```
A = np.ones((2, 2))
B = np.ones((2, 2))
```

- What is A + B?
- What is A + 10?
- What is A * B?

- **Remember:**
  - A * B is not the matrix product, it is an elementwise product.

S37 NUMPY ARRAYS: METHODS AND ATTRIBUTES

- Arrays have a variety of methods
  - .sort() # Sorts in place
  - .sum() # Sum
  - .mean() # Mean
  - .max() # Max
  - .argmax() # Returns the index of the maximal element
  - .cumsum() # Cumulative sum of the elements of A
  - .cumprod() # Cumulative product of the elements of A
  - .var() # Variance
  - .std() # Standard deviation
  - .transpose() # Transpose
- Attributes
  - .shape # Shape of the array

- NumPy arrays are **mutable** data types
  - their contents can be altered (mutated) in memory after initialization
- Here's an example:

```
a = np.array([42, 44])
b = a
b[0]=0.0
```

- What is a?

S38 SCIPY

- SciPy builds on top of NumPy to provide tools for scientific computing
  - Statistical testing
  - numerical integration
  - optimization
  - distributions and random number generation
  - signal processing
  - etc.
- SciPy is stable, mature and widely used
  - Many SciPy routines are thin wrappers around industry-standard Fortran libraries such as LAPACK, BLAS, etc.
  - It's not really necessary to "learn" SciPy as a whole
  - A more common approach is to get some idea of what's in the library and then look up documentation as required
- **Ex: numpyScipy.ipynb**

S39 EXAMPLES: SIMPLE STATS AND T-TEST

- Suppose we have sales totals for 30 sales reps over 12-month horizon
  - Find the average and standard deviation for each Rep
  - Find the average and standard deviation for each Month
  - Test whether sales in the second six-months are higher than in the first
    - Hint: use .mean(axis=…) and .std(axis=…) to find the average and standard deviation along a particular axis; use ttest_rel(… , …) to run a matched-pairs t-test

- First we will begin by importing the two libraries
  - Recall different ways of importing

```
import numpy as np
from scipy import stats
```

  - Why do we care which way to import?

S40 EXAMPLES: CORRELATION AND SIMPLE LINEAR REGRESSION

```
import numpy as np
from scipy import stats
```

- Correlation: np.corrcoef()
- Covariance: np.cov()
- Simple Linear Regression: stats.linregress()

S41 DATA FRAMES: PANDAS LIBRARY

- Pandas is a library in Python that handles Series and Data Frames data structures

```
import pandas as pd
```

- You can think of a Series as a "column" of data in excel
  - i.e., a collection of observations on a single variable
- You can think of a Data Frame as a "table" of data in excel
  - i.e., labeled data with multiple attributes

- More on Pandas can be found here
  - http://pandas.pydata.org/
  - 10 Minutes to Pandas: http://pandas.pydata.org/pandas-docs/stable/10min.html

→ Fast and Efficient Exploratory Analysis
  - → Easy to import / export data
  - → Easy to clean / filter data
  - → Easy to analyze / model
  - → Easy to organize / present the results

- **Ex: pandasDataFrames.ipynb**

S42 PANDAS: DATA FRAMES

- Data structure: **Data Frame**
  - Labeled Data
  - Multiple Attributes/Features
- Example: Cars Dataset

mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name	maker
18	8	307	130	3504	12	70	America	chevrolet chevelle malibu	chevrolet
15	8	350	165	3693	11.5	70	America	buick skylark 320	buick
18	8	318	150	3436	11	70	America	plymouth satellite	plymouth
16	8	304	150	3433	12	70	America	amc rebel sst	amc
…	…	…	…	…	…	…	…	…	…

S43 PANDAS: DATA FRAMES

- Why Data Frames?
  - Label-based slicing, indexing, subsetting
  - Intuitive merging and joining data sets
  - Easy to create or delete columns
  - Used as the fundamental data structure by most modeling software

S44 PANDAS: DATA INPUT / OUTPUT

- Read
  - CSV: pd.read_csv("cars_data.csv")
  - Excel: pd.read_excel("someFile.xlsx", "Sheet1")
- Example:

```
df = pd.read_csv("cars_data.csv")
```

- The cars dataset is made up of 392 samples of cars
  - Each sample contains multiple features
    - mpg
    - cylinders
    - horsepower
    - Etc.

- Write (data frame labeled df)
  - CSV: df.to_csv("someFileName.csv")
  - Excel: df.to_excel("someFileName.csv", sheetName="Sheet1"

S45 PANDAS: COMMON ATTRIBUTES AND METHODS

- Attributes
  - T – transpose
  - dtypes – data types
  - size – number of elements
  - shape – tuple representing the dimensionality
  - ix – label-location based indexer
- Methods
  - .head(n) – return first n rows
  - .tail(n) – return last n rows
  - .rename() – rename columns
  - .dropna() – drop missing values
  - .fillna() – fill missing values
- Complete List:
  - http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
- Hint: Use df.<TAB>

S46 PANDAS: DATA SUMMARY

- Label Based Subsetting – create a subset of the data

```
dfOneCountry=df[df['origin']=="America"]
dfOneCountry['horsepower'].describe()
```

- Group The Data

```
dfByCountry=df.groupby('origin')
dfByCountry['mpg'].describe()
```

- Pandas includes all common functions: mean(), max(), median(), etc.

```
dfByCountry['mpg'].mean()
```

- You can also apply your own functions using .aggregate() and .agg()

```
dfByCountry['weight'].aggregate(myMeanFun1)
dfByCountry['weight'].aggregate([myMeanFun1, myMedianFun, myStdDev])
```

RECURSION

S48 RECURSION

- **Recursion** is a method of solving problems that involves breaking a problem down into smaller and smaller sub-problems
  - Usually **involves calling a function itself**
  - May provide easier solutions to otherwise difficult programs

- **Drawback:** Every time a function calls itself it uses some memory
  - By default python stops the function calls after a depth of 1000 calls
  - You can modify the number of recursive calls using **sys** library:
    - sys.setrecursionlimit()

- Example of two programs:

```
def sumOfN(someList):
  theSum = 0
  for i in someList:
     theSum = theSum + i
  return theSum
```

```
def sumOfNRecursive (someList):
   if len(someList) == 1:
      return someList[0]
   else:
      return someList[0]+sumOfNRecursive(someList[1:])
```

- **Ex: recursion.ipynb**

S49 RECURSION

- The Three Laws of Recursion
  1. A recursive algorithm must have a base case
  2. A recursive algorithm must change its state and move toward the base case
  3. A recursive algorithm must call itself, recursively
- Base case is typically a problem that can be solved directly
  - Allows the algorithm to stop recursing
- A change of state means that some data that the algorithm is using is modified
  - Usually data gets smaller in some way

```
def sumOfNRecursive (someList):
  if len(someList) == 1:
    return someList[0]
  else:
    return someList[0]+sumOfNRecursive(someList[1:])
```

- A function fun1 is **direct recursive** if it calls itself
- A function fun1 is **indirect recursive** if it calls another function and that function calls fun1 directly or indirectly

- Examples
  1. Covering an Integer to Binary
  2. Calculating factorial

S50 MEMOIZATION

- Consider the following recursive implementation:

```
def fibRecursive(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fibRecursive(n-1) + fibRecursive(n-2)
```

- What is the issue?
- Example: fibRecursive(6)

- **Ex: memoization.ipynb**

- **Memoization** – record the value of the call and then look it up rather than compute each time

```
def fibFast(n, values=None):

  if values==None:
    values={}

  if n==0:
    return 0
  if n==1:
    return 1

  if n in values:
    return values[n]
  else:
    result=fibFast(n-1,values)+fibFast(n-2,values)
    values[n]=result
    return result
```

Testing and Debugging

S52 **Programs Don't Always Function Properly**

- **Testing** is the process of running a program to confirm whether it works as intended
- **Debugging** is the process of trying to fix a program that you already know doesn't work

Split the program into separate components that can be implemented, tested, and debugged independently of other components

- functions
- cells in notebook

S53 TESTING

- Black-box testing – tests without looking at the code
  - Testers and implementers are drawn from separate populations (e.g., development vs quality assurance divisions)
  - During testing consider typical cases and boundary conditions
  - If multiple inputs are mutable, think about passing the same input for two arguments
- Glass-box testing – tests with looking at the code
  - Identify special cases based on the code
  - Ideally would test all possible paths through the program

TEST EARLY AND OFTEN!

S54 DEBUGGING

- "Bug" refers to an error in the program
  - **Overt bug** has obvious manifestation (e.g., the program crashes)
  - **Covert bug** has no obvious manifestation (e.g., the program may run to conclusion, but provide an incorrect answer)
  - **Persistent bugs** occur every time the program is run with the same input
  - **Intermittent bugs** occur only some of the time even for the same inputs
- Question: which bugs are best?
- **Debugging** is the process of searching for an explanation of the error
  - starts when testing has demonstrated that the program behaves erroneously
- The key to debugging is being systematic
  - Start by studying the data (both on tests that revealed the problem and those that did not)
  - Form a hypothesis that you believe to be consistent with all the data
  - Design and run a repeatable experiment with the potential to refute the hypothesis
  - Keep records

S55 DEBUGGING

- Look for common mistakes
  - Misspelled names
  - Failed to reinitialized a variable
  - Testing floating point values equality with '=='
  - Mutability

- Helpful approaches
  - Try to explain why the program is doing what it is doing
  - Try to explain your code to somebody else
  - Try to write documentation (e.g., detailed comments) for your code
  - Take a break and revisit later

- Brute force debugging:
  - **print** statements

- Debuggers
  - Setting a break point
    - Can investigate the value of variables at particular points in the program
    - Step through the program
    - To find out what you can do inside ipdb (or pdb) you can use help 'h'

- **Ex: debugging.ipynb**

EXCEPTIONS AND ASSERTIONS

S57 EXCEPTIONS AND ASSERTIONS

- Exception are anomalous or exceptional conditions
- You have probably have encountered **unhandled exception** which cause program to terminate
- An exception does not need to lead to program termination, instead exceptions can be handled by the program
  - An exception is something that the programmer should anticipate (e.g., trying to open a file that does not exist)
  - **try** / **except** syntax
  - Can be used as a control flow mechanism
- **assert** statement provides programmers with a simple way to confirm that the state of the program is as expected
  - **assert** *Boolean expression*
  - **assert** *Boolean expression, argument*
- Assertions are useful defensive programming tool
  - Confirm that argument of appropriate types
  - Confirm that intermediate values are as expected
- **Ex: assertions.ipynb**

S58 MANY EXCEPTIONS ARE BUILT-IN

- **IndexError** – when a program tries to access an element that is outside the bounds of an indexable type
- **TypeError** – when an operation or function is applied to an object of inappropriate type
- **NameError** – when a program tries to use a variable or function name that has not been defined
- **MemoryError** – when a program runs out of RAM
- **ValueError** – when a built-in operation or function receives an argument that has the right type but an inappropriate value (e.g., int("21") vs int("hello"))
- **SyntaxError** – these are usually typing mistakes or something is wrong with the structure of your program
- **ZeroDivisionError** – when trying to divide by zero (also applies to the modulus operator)
- **Ex: exceptions.ipynb**

Object Oriented Programming

S60 Object Oriented Programming

- What is a class?
  - A user-defined **template for creating an object**
  - Logical grouping of **data** (*attributes*) and **functions** (*methods*)
- Keyword **class** is used to define a class
- Few things to keep in mind as you are starting to learn classes
  - __init__() is the method to initialize the object
  - self is the parameter that refers to the object itself
  - you can define operations for the classes you construct
- Some Terminology
  - **Instance**
    - An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
  - **Operator overloading**
    - The assignment of more than one function to a particular operator.
  - **Inheritance**
    - The transfer of the characteristics of a class to other classes that are derived from it.
- **Ex: classes.ipynb**

S61 **SAVING OBJECTS**

- **pickle** is a module that provides Python with ability to save objects
  - Writes the object as one long stream of bytes
  - pickling is also known as serialization, marshaling, or flattening

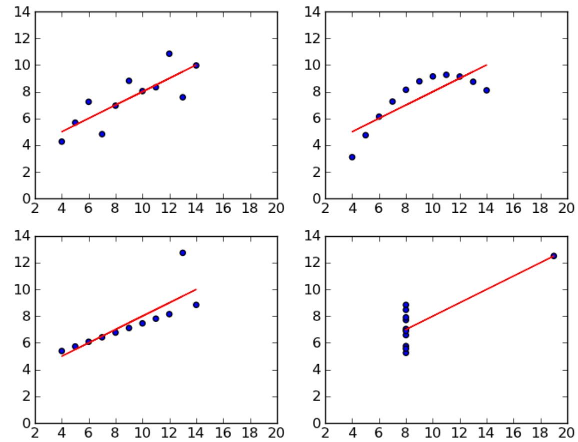
```
import pickle

myDictionary1 = {0: “value 0”, “a”: “letter a”}

f = open(“test.p”, “w”)

pickle.dump(myDictionary1,f)

f.close()
```

- **Ex: pickle.ipynb**

BASIC DATA VISUALIZATION
MATPLOTLIB, SEABORN

S63 WHY DATA VISUALIZATION?

- Consider four datasets A, B, C, D
  - Each dataset has 11 observations (x,y)
- Questions
  - How similar are these data sets?
  - Is the relationship between x,y the same among the four datasets?
- We know
  - A.x.mean() = B.x.mean() = C.x.mean() = D.x.mean() = 9
  - A.x.var() = B.x.var() = C.x.var() = D.x.var() = 11
  - A.y.mean() = B.y.mean() = C.y.mean() = D.y.mean() = 7.50*
  - A.y.var() = B.y.var() = C.y.var() = D.y.var() = 4.12*
  - Correlation between x and y in each case is 0.816
  - Linear regression line in each case is y = 3 + .5x
- **Ex: anscombe.ipynb**



S64 DATA VISUALIZATION

- Ten Simple Rules for Better Figures
  - Rougier, Droettboom, Bourne (2014)
  - [Article link]

1. Know your audience
2. Identify your message
3. Adapt the Figure to the Medium
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid “Chartjunk”
9. Message Trumps Beauty
10. Get the right tool

- Chart Message
  - Distribution
  - Relationship
  - Composition
  - Comparison
- Chart Types
  - Bar Charts, Histograms, Box Plots, Violin Plots
  - Scatter Plots, Bubble Plots, Line Plots, Heat Maps
  - Pie Chart, Stacked Bar Charts

S65 DATA VISUALIZATION IN PYTHON: MATPLOTLIB AND SEABORN

- matplotlib is a graphics library designed for scientific computing

```
import matplotlib.pylab as plt
```

```
# Magic function to make matplotlib inline
%matplotlib inline
```

- **Ex: matplotlib.ipynb**
  - Scatter Plot
  - Pie Chart
  - Histogram

- More info on matplotlib here: http://matplotlib.org/users/beginner.html

- seaborn is a Python visualization library based on matplotlib
  - Aim to make plots that are "production ready"

```
import seaborn as sns
```

- **Ex: seaborn.ipynb**

- More info on seaborn here: https://stanford.edu/~mwaskom/software/seaborn/introduction.html#introduction

RANDOMIZATION AND MONTE CARLO SIMULATIONS

S67 MONTE CARLO SIMULATIONS

- The **Monte Carlo** method uses repeated random sampling to generate simulated data to explore the behavior of a complex system or a process.

- The MC simulations can be applied to a wide range of problems
  - Physics
  - Economics and Finance
  - Engineering
  - Biology
  - Politics, Sports (e.g., FiveThirtyEight.com)

- Approach
  1. Determine the model
  2. Repeatedly sample from the random components of the model to obtain many realizations
  3. Evaluate the measures of interest

- The Birthday Problem
  - How many people would you need in a group in order for there to be a 50-50 chance that at least two people will share a birthday?

- **Ex: monte_carlo_birthday.ipynb**

S68 RANDOM SAMPLING: NUMPY.RANDOM, SCIPY.STATS, RANDOM

- random, numpy.random and scipy.stats provide functions for generating random variables

Ex: random_sampling.ipynb

```
import random
print(random.choice([1,2,3,4]))          #random sample with replacement (each is equally likely)

import numpy as np
np.random.beta(5, 5, size=3)

from scipy.stats import beta
q = beta(5, 5)                           # Beta(a, b), with a = b = 5
obs = q.rvs(2000)                        # 2000 observations
print(q.cdf(0.4))                        # Cumulative distribution function
print(q.pdf(0.4))                        # Density function
print(q.ppf(0.8))                        # Quantile (inverse cdf) function
print(q.mean())                          # Average
```

S69 RESAMPLING

- If you know what distribution data come from or can justify an assumption about the distribution, then you have a variety of tools to assist you with hypotheses testing
- **However**, most of the time
  - cannot make a reasonable assumption on the distribution
  - using small sample size
  - testing a parameter near the boundary
- Then there could be a **problem…**
- In this case you can use resampling techniques
  - **Bootstrap** to obtain standard errors
  - **Permutation Tests** (exact tests) or randomization tests to get a null distribution

S70 BOOTSTRAP

- Method of estimating the sampling distribution of a statistic
- Suppose you have a sample $\boldsymbol{S} = \{x_1, x_2, \dots x_{10}\}$ and you are interested in a certain statistic $\boldsymbol{M}$ (e.g., mean, median, standard deviation, etc.)
  - Repeat many times (e.g., $N = 10{,}000$)
    - Sample from $\boldsymbol{S}$ (with replacement) to create a new sample S_n
    - Calculate the statistic of interest M_n for S_n
  - → Thus, you have a distribution of 10,000 statistics
  - → Standard deviation of this distribution is the standard error of the statistic of interest

- **Ex: bootstrap.ipynb**

- **Exercise**: write a program to calculate bootstrap confidence interval of $\boldsymbol{M}$
- **Question:** when would we want to calculate bootstrap confidence intervals?

S71 PERMUTATION TEST

- Permutation test (also known as exact test) is a non-parametric approach to test a hypothesis by establishing the distribution of the test statistic under the null and comparing the original test statistic to that distribution.
  - The **permutation distribution** is obtained by calculating all possible values of the test statistic
  - In practice, we use an approximation to a true permutation test: instead of checking all possible permutation, we randomly sample using a Monte-Carlo method.

- **Ex: permutation_test.ipynb**

- Example: samples [8, 11,13,15] and [14, 20, 20, 23]
  - What is the chance of difference in means of $\frac{8+11+13+15}{4} - \frac{14+20+20+23}{4} = -7.5$?
  - Under the null hypothesis we can combine the two samples
    - [8, 11, 13, 15, 14, 20, 20, 23]
  - Permute and split into two samples of the original size
    - Example permutation 1: [8, 11, 13, 14] and [15, 20, 20, 23]
    - Example permutation 2: [23, 11, 13, 14] and [15, 20, 20, 8]
  - Calculate the statistic of interest
    - Example permutation 1: -8
    - Example permutation 2: -.5

DYNAMIC DATA VISUALIZATION

S73 Dynamic Data Visualization in Python: bokeh

- bokeh is an interactive visualization library for web browsers
  - Can transform visualization written in other libraries (matplotlib, seaborn, ggplot)
- If it is not already installed, you need to go to the command prompt and either use conda or pip to install bokeh:

```
> conda install bokeh
```

or

```
> pip install bokeh
```

- More info on bokeh here: http://bokeh.pydata.org/en/latest/
- Bokeh Tutorial
- http://nbviewer.jupyter.org/github/bokeh/bokeh-notebooks/tree/master/tutorial/

S74 bokeh: Getting Started

- There several modules within bokeh library that we will be working with
- bokeh.io – input output module
  - output_notebook() – output to Jupyter notebook
  - output_file() – output to file (typically .html file)
  - show() – display plot in new window in addition to saving to notebook or file
  - save() – save plot
- bokeh.plotting – module that simplifies plot creation with default axes, grids, tools, etc.
- Example: let us plot a simple scatter plot with five points
  - X = [1,2,3,4,5]
  - Y = [6,7,2,4,5]
- **Ex: bokeh.ipynb**

S75 HOLOVIEWS

- HoloViews is an open-source Python library for data analysis and visualization
  - instead of building a plot using direct calls to a plotting library, you first organize your data into appropriate format and provide semantic information required to make it visualizable
  - then you specify additional metadata as needed to determine more detailed aspects of your visualization
  - Works with Bokeh and Matplotlib

Intro

- http://holoviews.org/getting_started/Introduction.html

- Ex: **holoviews.ipynb**

S76 FOLIUM

- Folium is a Python library that uses leaflet.js JavaScript library for interactive maps
  - Makes it relatively easy to visualize data on a leaflet map
  - Can bind data to a map for choropleth visualization as well as passing markers on the map

- Folium is not as well developed or supported as Holoviews and Bokeh, so it is not included with Anaconda by default
- to install folium:

```
> pip install folium
```

or

```
> conda install folium -c conda-forge
```

- **Ex: folium.ipynb**

INTRODUCTION TO PARALLEL COMPUTING

S78 DESIGNING PARALLEL PROGRAMS

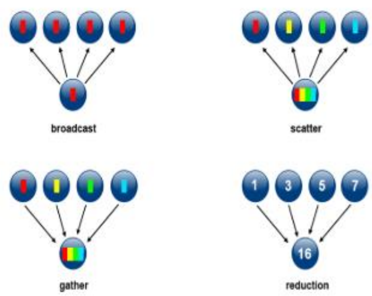
- First step is to determine whether the problem is **parallelizable**
- Some examples of **parallelizable** programs:
  - Monte-Carlo Simulations
  - Search
- Some examples of non-parallelizable (a.k.a. **serial**) programs
  - Newton's method (and most optimization algorithms)
  - Dynamic Programming
- Need to Consider
  - Communication overhead
  - Task synchronization
  - Data Dependence
- **Question**: which of these loops has data dependence?

```
t = np.ones(10)
for i in range(1,10):
    t[i] = t[i-1]*2
```

```
t = np.ones(10)
for i in range(1,10):
    t[i] = i**2
```

S79 MESSAGE PASSING INTERFACE

- Message Passing Interface (MPI)
  - standardized message-passing specification
  - runs on virtually any hardware platform
    - Distributed Memory
    - Shared Memory
    - Hybrid



- [Many libraries](#) available in Python to work with parallel processes. E.g.,
  - mpi4py
  - pyMPI
  - IPython parallel
  - multiprocessing

- **multiprocessing** module has a lot of tools dealing with parallel programming
  - Symmetric Multiprocessing
  - [Documentation](#)

S80 ASIDE: MAP

- map() is built-in a function which takes two arguments: *func*, *seq*
  - *func* is the name of a function
  - *seq* is an iterable (i.e., list, tuple, etc.)

- map() applies the same function *func* to all the elements of the sequence *seq*
  - map() returns an **iterator**
  - An **iterator** in Python is an object which will return data, one element at a time

```
items = [1, 2, 3, 4, 5]
def sqr(x): return x ** 2
map(sqr, items)
```

- Notice that the exact same tasks is done, and the sequence does not matter.

- **multiprocessing** module has a function **pool.map()** which does the same thing as map() but on multiple processors

S81 EXAMPLES

- Example 1: **HelloParallel.py**

```
from multiprocessing import Pool
def f(x): return "Hello "+str(x)
if __name__ == '__main__':
    p = Pool(4)
    print(p.map(f, [1, 2, 3]))
```

- The pool class represents a pool of worker processes
  - Each worker is assigned one of the input items in the list until all items in the list have been processed
  - Each worker carries out the function on the iterable
- In **ParallelExample2.py** we will asses the gain in execution time when we scale out from 1 to 2 to 3 … to 8 processes.
  - What do you think will happen?
  - What do you think will happen if we try to assign this to 10 cores?
- In **MultiArgWrapper.py** we will see an example of how to pass multiple arguments to the function
  - Note that pool.map() takes only two arguments
  - Note on variable length arguments.
  - Basic idea – write a 'wrapper' function that will take advantage of variable length arguments and only require 1 parameter

REGRESSION

S83 LINEAR REGRESSION

- What are the main assumptions of the linear regression?
  - Variables are normally distributed
  - Linear relationship between the independent and dependent variables
  - Variables are measured reliably
  - Homoskedasticity
    - Variance of errors is the same
- We will work with Boston Housing data set from sklearn.

```
from sklearn.datasets import load_boston
boston = load_boston()
```

- Let us run the linear regression with the Boston housing data
  - By brute force
  - Using built-in functions
- **Ex: regression.ipynb**

S84 LOGISTIC REGRESSION

- In the OLS framework, the dependent variable is a linear function of the independent variables
  - $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon$
- What if the dependent variable is a categorical variable (e.g., yes/no)?
- Logistic regression
  - Predicts the **probability** of an outcome rather than the outcome itself
  - $y = \begin{cases} 1 & \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon \quad > 0 \\ 0 & \qquad otherwise \end{cases}$
  - ϵ distributed according to logistic distribution
- Another way of writing it is
  - $\log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$
- Solved by **maximizing** the "likelihood function"
  - i.e., finding the set of parameters $\beta_0, \beta_1, \dots, \beta_n$ for which the probability of the observed data is the greatest
  - Procedure known as **maximum likelihood estimation** (beyond the scope of this class)
- **Ex: logistic_regression.ipynb**

UNSUPERVISED LEARNING

S86 **K-MEANS**

- The idea is to find K groups of observations (**clusters**), denoted by C_k, which are similar to one another.
- The mathematical objective is to partition observations into K sets so as to **minimize the within-cluster sum of squares**:

$$Min \sum_{k=1}^{K} \sum_{\{x_n \in C_k\}} ||x_n - \mu_k||^2 \text{ with respect to } C_k, \mu_k$$

- where $\boldsymbol{\mu_k}$ is the mean point of C_k, and is referred to as ***centroid***.
- The problem is NP hard (similar to Knapsack problem)
  - At least as difficult as non-deterministic polynomial-time. Solution may or may not be verified in polynomial time
- Good heuristic algorithms. Example: Iterative Refinement (Lloyd's Algorithm)
  - **Step 0:** Start with an initial guess of a set of centroids μ_k.
  - **Step 1:** Create clusters containing points closest in distance to each centroid
  - **Step 2:** Update the centroids as the means of all points in each cluster.
  - **Step 3:** Repeat 1 and 2 until the assignments of clusters and centroids does not change (or max number of steps reached)
- **Ex: k_means.ipynb**

S87 DETERMINING NUMBER OF CLUSTERS

- Need some way to determine number of clusters
- One method to validate the number of clusters is the **elbow method**
  - Run k-means clustering on the dataset for a range of values of *k* (say, *k* from 1 to 10),
  - For each value of *k* calculate the sum of squared errors (SSE)
  - Plot a line chart of the SSE for each value of *k*. If the line chart looks like an arm, then the "elbow" on the arm is the value of *k* that is the best.
- Our goal is to choose a small value of *k* that still has a low SSE, and the elbow usually represents where we start to have diminishing returns by increasing *k*.

- **Ex: k_means.ipynb**

ADDITIONAL TOPICS

S89 PYTHON DATA STRUCTURES: SETS

- Sets are unordered lists with no duplicate entries
  - A comma-delimited sequence within parentheses
  - You can extract values by an index
  - The first element starts at index 0

```
mySet1 = set([1, 2, 3])
mySet2 = set([2, “Text1”, “Text2”])
```

- Basic Operations
  - difference
  - intersections
  - union

```
print mySet1 - mySet2
print mySet1 & mySet2
print mySet1 | mySet2
```

- Sets can be a powerful tool
  - For more go to http://www.learnpython.org/en/Sets
- **Ex: sets.ipynb**

S90 ADDITIONAL CONTROL FLOW STATEMENTS

- break
  - Break out of the smallest inclosing loop

```
n=10
while n<20:
        n=n+3
        if n==16:
            break
        print(n)
```

- continue
  - Continue with the next iteration of the loop
- pass
  - Does nothing
  - Often used as a place-holder when you are working on a new code
- **Ex: loops.ipynb**

S91 USEFUL FUNCTIONS

- range() [in Python 3 it is not a function but rather an object]
  - Commonly used with for loops

```
range(5) # [0, 1, 2, 3, 4]
```

```
range(2, 5) # [2, 3, 4]
```

```
range(2, 5, 2) # [2, 4]
```

- enumerate()
  - Yields both the index and the item itself

```
for index, drink in enumerate([‘Water’, ‘Gatorade’, ‘Coke’]):
        print(“Drink #”, index, “ is ”, drink)
```

- len()
  - Get length of a list
- sum()
  - Get sum of a list

S92 USEFUL FUNCTIONS

- zip()
  - Returns an iterator of tuples, where the *i*-th tuple contains the *i*-th element from each of the argument sequences or iterables.
- filter()
  - Construct an iterator from those elements of *iterable* for which *function* returns true.
- For a complete list of functions and explanations go to
  - https://docs.python.org/3.6/library/functions.html
- **Ex: useful_functions.ipynb**

- Another I\O Approach: CSV library
  - File reading and writing module
  - Comma* Separated Values

```
import csv
```

- Methods
  - .reader()
  - .writer()

```
f = open(“testFile1.txt”)
reader = csv.reader(f)
for row in reader:
    print(row)
f.close()
```

S93 USEFUL FUNCTIONS

- map() is built-in a function which takes two arguments: map(*func*, *seq*)
  - *func* is the name of a function
  - *seq* is a list or a tuple
  - map() applies the function *func* to all the elements of the sequence *seq*
  - map() returns an **iterator**
- DataFrame.apply(*func*, axis=0, …)
  - Applies a function func along an axis of the DataFrame
- **Ex: map_apply.ipynb**

- An iterator in Python is an object which will return data, one element at a time
- An object is called iterable if we can get an iterator from it. Most of built-in containers in Python like: list, tuple, string etc. are iterables.

S94 FUNCTIONS WITH DEFAULT ARGUMENTS

- Python functions may have default arguments
  - if the function is called without the argument, the argument gets its default value.
  - arguments can be specified in any order by using named arguments.
  - Example function with two optional arguments:

```
def fibArb(n, arg1=0, arg2=1): #Fibonacci sequence starting with arg1 and arg2
    a, b = arg1, arg2 #multiple assignments can be done in one line
    while a < n and arg1<arg2:
        print(a,)
        a, b = b, a+b #note multiple assignments
```

  - Notes:
    - In Java, C++ you must specify the datatype of the function return value and each function argument (statically-typed)
    - In Python, you never explicitly specify the datatype. Based on what value you assign, Python keeps track of the datatype internally.

S95 TIMING YOUR CODE WITH TIME LIBRARY

- time library

```
import  time
t1=time.time()
for n in range(10000000):
        myTax(n)
t2=time.time()
print(“Took “,str(t2-t1),”Seconds”)
```

- **Ex: timing_code.ipynb**

S96 TIMING YOUR CODE WITH IPYTHON

- IPython treats any line whose first character is ‘%’ as a special call to a ‘magic’ function
  - magic allows you to control behavior of IPython itself
  - ‘%’ is a line magic
  - ‘%%’ is a cell magic
- timeit magic

```
from taxes import *
%timeit myTax(10000000)
```

```
%%timeit
from taxes import *
for n in range(10000000):
        myTax(n)
```

- **Ex: timing_code.ipynb**

S97 List Comprehension

- A compact way of constructing lists
  - similar to the way sets/vectors are defined in mathematics
- Suppose we have the following price list

```
myPriceList = [10.5, 7.8, 15.0]
```

- How would we specify a tax list in math notation?
  - myTaxList = { x*0.0625 : x in myPriceList}
- List comprehension in Python

```
myTaxList = [x*0.0625 for x in myPriceList]
```

- **Ex: list_comprehension.ipynb**

- Look at it from right to left
  - Python loops through myPriceList one element at a time, temporarily assigning the value of each element to the variable x.
  - Python then applies the function x*0.0625 and appends that result to the returned list.
- Notes
  - list comprehensions do not change the original list.
  - Python constructs the new list in memory, and then assigns the result to the variable.
  - It is safe to assign the result of a list comprehension to the variable that you're changing.

S98 IPython magic functions

- IPython treats any line whose first character is '%' as a special call to a 'magic' function that allow you to control behavior of IPython itself
  - '%' is a line magic
  - '%%' is a cell magic
- Some functions:
  - magic – print information about the magic function system
  - cd – change the current working directory
  - run – convenient way to run python files within your IPython session
  - load – convenient way to load python files within your IPython sessions
  - file – convenient way to create python files within your IPython session
  - timeit – time execution of a Python statement or expression
  - debug – activate the interactive debugger

- **Ex: magic_functions.ipynb**

S99 SYS LIBRARY

- **sys** module provides access to variables maintained by the interpreter
  - sys.path -- a list of directories that Python searches every time you import a module
    - sys.path.append('<*somedirectory*>') – if you want to load a file that is in folder other than your working folder, you may want to add that folder the path
  - sys.platform – platform identifier that can be used to append platform-specific components
  - sys.version – version number of the Python interpreter plus additional information on the build number and compiler used
  - sys.float_info – information about the float type
    - sys.float_info.max – maximum representable finite float
    - sys.float_info.epsilon – smallest representable float that is greater than 0

- **Ex: sys_module.ipynb**

- More details: https://docs.python.org/3/library/sys.html#module-sys

S100 PYTHON 2: REDUCE(), PYTHON 3: FUNCTOOLS.REDUCE()

- reduce() is a function that takes two arguments: reduce(*func*, *seq*)
  - *func* is the name of a function
  - *seq* is an iterable (i.e., list, tuple, etc.)
  - Apply *function* of two arguments **cumulatively** to the items of *iterable*, from left to right, so as to **reduce** the iterable to a single value.

- Suppose we have a function sum2():

```
def sum2(a,b):
        return a+b
```

- And we have a list x=[1,2,3,4,5,6], then what is the output of:

```
functools.reduce(sum2,x)
```