Python Notes

* Variable Names
  + Conventions
    - snake\_case - separate “words” w/underscore
    - Start w/letter or “\_”
      * Private vars: vars beginning w/”\_”:
    - Letters, #s, or \_
    - Case sensitive
    - Constants: user upper (e.g. PI)
    - Dunder: \_\_<var>\_\_
  + Keywords: [*https://www.w3schools.com/python/python\_ref\_keywords.asp*](https://www.w3schools.com/python/python_ref_keywords.asp)
* Strings
  + Escape Sequence
    - Includes: \n, \t,
    - Ex: ‘It\’s sunny’
  + Formatted Strings
    - Python 3
      * print(f'Hi {name}. You are{age} years old.')
    - Python 2
      * print(‘Hi {}. You are {} years old.’.format(name, age))
  + String Indexing
    - <string>[start:stop:stepover]
    - <string>[::-1] - reverse a string
* Lists
  + Are ordered
  + new\_cart\_ptr = amazon\_cart - new cart pts to mem amazon\_cart pts to
  + new\_cart = amazon\_cart[:] - new copy is made of amazon\_cart
  + Matrix
    - Multi-dimensional Array *(array of arrays)*
  + <list>.reverse() == <list>[::-1]
  + <list>.copy() == <lis>[::]
  + Combine <list> into a string
    - new\_str = ‘ ‘.join([...iterable items…]) *(comma-separates items)*
  + List Unpacking
* Dictionaries
  + Are unordered
  + Keys
    - Must be immutable - are hashed
    - Must be unique w/in a single dict.
      * Assigning multi-values to same key will work, but just overwrites the previous value
        + Ex:  
          My\_dict = {  
           ‘A’: ‘mystr’,  
           ‘A’: ‘yourstr  
          }
* Tuples
  + Immutable
    - Can’t sort, reverse, etc.
  + Can access values via indexes []
  + Can use tuples as dict keys *(can be hashed - immutable)*
    - User = {  
       ‘(1,2): [1,2,3],  
       ‘greet’: ‘hello’  
      }  
      print(user[(1,2)] ⇒ [1,2,3]
  + Can index a tuple
  + Can slice [... : ... : …] a tuple
  + Can append to tuple
    - <new\_tuple> = <tuple\_var> + (<new\_value>,)  
      *(****Note****: the appended value* ***must*** *contain the trailing comma)*
* Sets
  + Unordered collection of **unique** objects
    - No duplicate members
  + Not support indexing
  + <set1> | <set2> -- union of sets
  + <set1> & <set2> -- intersection of sets
* Operator
  + Syntax: [on\_true] if [expression] else [on\_false]
  + condition\_if\_true if condition else condition if false
  + Ex:  
    can\_message = “message\_allowed” if is\_friend else “not allowed to message”
* Logical Operators
  + and, or, not
* Equality (==) vs. “is”
  + “is” - checks if same memory location
* Iterables
  + list, dict, tuple, set, string
  + Iterating over a dict:
    - for item in <dict>.items(): to produce tuples of (key:value) pairs
      * If want to split keys:values into distinct vars:
        + for key, value in <dict>.items():
    - for item in <dict>.keys(): to produce keys
    - for item in <dict>.values(): to produce values
  + Range function
    - Iterate over sequence of ints
    - for i in range(100):
    - for i in range(5,25):
    - Descending Count *(reverse iteration)*
      * for i in range(35,5,-5):
    - Convention: if !need “i” in loop body, use “ \_ “
      * for \_ in range(100):
    - Create lists of ints
      * for \_ in range(100):  
         print(list(range(10)))
* Enumerate Function
  + Iterates iterable
  + Returns index & value
  + for i, char in enumerate(‘myString’):
* While Loop
  + Can add an “else:” statement to while loop (a la “if” statement)
  + “break” breaks past the else statement
* Functions
  + “*First-class citizens*”
    - Passed as vars, arguments w/in other funcs
  + Can nest function definitions
  + Docstrings
    - help(<func\_name>) - displays func’s docstring  
      OR
    - *<func\_name>*.\_\_doc\_\_ - must print() this to view it w/in editor
  + Arguments
    - Positional - listed in the correct order for parms in func
    - Keyword - listed by name (e.g. func(name = ‘*<name>*’, age = ‘*<age>*’)
  + Parameters
    - “arguments” w/in func definition
      * Default parms: assign default values in func definition
        + Values used if func call !include args
  + \*args & \*\*kwargs parameters
    - Order of func parms:
      * parms, \*args, default parms, \*\*kwargs
      * def my\_function(<std parms>, \*args, \*\*kwargs)  
         <code>
    - \*args
      * Accepts positional args
      * “\*” - unpacking operator for positional args
        + Can use in func definition &/or func call

def my\_sum(\*args):

print(my\_sum(\*list1, \*list2, \*list3))

Unpacks before sending to func

* + - * Converts inbound args to an immutable tuple
      * for x in args:  
         <code>
    - \*\*kwargs
      * Accepts keyword args (*“named” arg*s) - !positional args
      * “\*\*” - unpacking operator for keyword args
      * Converts inbound args to a dict
      * Iterate over keys
        + for arg in kwargs:  
           <code>
      * Iterate over values
        + for arg in kwargs.values():  
           <code>
* Scope Rules
  + 1) Local scope
  + 2) Parent local scope
  + 3) Global scope
  + 4) Builtin Python funcs
  + global keyword
    - Enables using global var w/in func
    - Best practice - don’t use global w/in func - pass in necessary global value to func
  + “nonlocal” keyword
    - Refers to var w/in the parent func
* OOP (Classes)
  + Naming Convention
    - Camel-case - begins w/uppercase
    - Typically singular name - !plural as it’s a “blueprint"
  + 4 Pillars
    - Encapsulation: *binding of data (attributes) & funcs (methods) into a “black box”*
    - Abstraction: *permit access to only what’s req’d & hide implementation details*
      * Public & Private
        + No private category of vars & funcs
        + Use “\_” by convention

\_<var\_name> (e.g. \_name)

\_<func\_name> (e.g. \_speak())

* + - Inheritance: *derived classes (aka sub-classes)*
      * “object” is parent of all classes
        + object’s Dunder methods are avail to all its derived classes
      * isinstance() -- used to determine class hierarchy
      * super() - access parent class’ methods/attributes
    - Polymorphism: *reuse diff forms of same method*
  + Class Object Attribute
    - Common “global” attribute avail to all instances
  + Class Methods
    - [*Instance vs. Static vs. Class Methods in Python: The Important Differences*](https://www.makeuseof.com/tag/python-instance-static-class-methods/)
    - Instance Method:
      * *access data/properties unique to each instance*
      * Relies on **self** - !need to pass **self** into method call  
        *(Python auto-”inserts”* ***self*** *into method call)*
    - Static Method: @staticmethod
      * *self-contained code - !access anything in class*
        + *Only work w/args passed in*
      * Utility funcs - perform a task in isolation
    - Class Methods: @classmethod
      * *modify class specific details - access limited methods in the class; !access instance data*
      * *Can call @staticmethods*
      * Relies on **cls**- !need to pass **cls** into method call  
        *(Python auto-”inserts”* ***cls*** *into method call)*
        + def add\_nums(**cls**, num1, num2):
      * Can access via 2 ways:
        + *<ClassName>*.*<class\_method*>()
        + *<instance\_name>*.*<class\_method*>()
      * Can access via class instance, but !need to instantiate an instance to use it
      * Can use to instantiate a class obj from w/in the @classmethod definition - useful if need a func as part of instantiation process
        + return cls(‘Teddy’, num1 + num2)
  + help(<class> OR <instantiated obj>) - prints class attributes & methods
  + [*Class instance creation*](https://www.concentricsky.com/articles/detail/pythons-hidden-new)  
    (Note: normal instance instantiation (new\_instance = <class\_name>()) execs \_\_new\_\_() & \_\_init\_\_()
    - 1) Object instantiation - \_\_new\_\_()
    - 2) Constructor - \_\_init\_\_()
      * Customizes each instance *(i.e. data w/in each instance)*
  + Introspection
    - Examine an obj
    - Functions
      * dir(*<obj>*) - displays all attributes & methods of an instance
  + Dunder Methods
    - “Magic Methods” - allow customizing of classes
      * Modifying “base” Built-in Python functions w/in a class
        + Ex: def \_\_mul\_\_(): - overrides default “\*” functionality
    - Inherited from base Object class
    - Represent Built-in base functions
      * Ex: modify \_\_len\_\_, and len() uses its definition when executing
      * Ex: *<instance\_name>*() -- will exec \_\_call\_\_() if implemented in class
      * Built-in funcs exec dunder methods under-the-covers
        + Ex:

str() - execs \_\_str\_\_()

len() - execs \_\_len\_\_()

* + - Can be overridden in custom class
  + Multiple Inheritance - inherit from multiple super classes
    - Must accommodate diff parms for each super class
      * Create \_\_init\_\_() for each super w/in the lower class, and pass the appropriate args to each
    - MRO (Method Resolution Order)
      * Multiple inheritance “paths”
      * *Depth Search First* alg
      * <instance\_name>.mro() - displays run-time inheritance path  
        OR  
        <class>.\_\_mro\_\_
* Functional Programming
  + Isolate funcs from data
    - Pass in & return data from func
    - Func might contain/use internal data structures, vars, etc.
      * Funcs & data - !contained w/in OOP classes
  + Rules *(“Pure Funcs”)*
    - 1) Given same input, will always produce the same output
    - 2) Func should not produce side-effects
      * “*!interact w/anything in outside world*”
      * Ex: exec print() statement w/in func
        + Give control to print() func
      * All vars, data structures, etc. s/b local to func
    - Benefits
      * Less buggy code
      * Easier to test
  + Builtin Funcs
    - map(*<func>*, *<iterable>*) - “maps” (execs) func w/iterable passed in
      * Creates new iterable - !modify the *<iterable>* passed in
      * If printing map results, cast to list()
    - filter(*<func>*, *<iterable>*)- filters on supplied func & iterable
      * Recv’s True or False from supplied func for each item
    - zip(*<iterable>, <iterable>, ...*) - combines 2+ iterables into a single iterable
      * Each positional items are combined into a tuple
      * Ex: zip([1, 2, 3], [10, 20, 30]) == [(1, 10), (2, 20), (3, 30)]
      * Ex: zip([1, 2, 3], (10, 20, 30)) == [(1, 10), (2, 20), (3, 30)]
      * Ex: Combine usernames and phone #s from a DB → tuples
    - reduce(*<func>*, *<iterable>*, *<initial value>*) -
      * Must import: from functools import reduce
      * “Reduce” supplied iterable into a single value
        + e.g. summing all ints w/in a list → single int
      * map(), filter(), & zip() use reduce() internally
    - [*bytes*](https://docs.python.org/3/library/functions.html#func-bytes)*([<code>, <code>, …])*
  + lambda expressions
    - Purpose:
      * 1) “Use once” funcs
      * 2) Anonymous (nameless) funcs
    - Syntax: lambda parm: action(parm)
    - Ex: list(map(lambda item: item\*2, my\_list))
  + Comprehensions
    - Quick mechanism for creating lists, dicts, sets from existing iterables and performing action on items before inserting into created iterables
    - List Comprehension
      * Syntax: <list> = [parm for parm in <iterable> [optional conditional]]
        + Ex: my\_list = [char for char in ‘hello’]
        + 1st parm can be an expression

my\_list = [num\*2 for num in range(0,100)]

* + - * + Can add optional condition to end of comprehension

my\_list3 = [num\*\*2 for num in range(0,25) if not num % 2]  
(*Opt condition can filter data)*

* + - Set Comprehension
      * Syntax: <set> = {parm for parm in <iterable> [optional conditional]}
        + Ex: my\_list3 = {num\*\*2 for num in range(0,25) if not num % 2}
    - Dictionary Comprehension
      * Syntax: <dict> = {k,v\*\*2 for k,v in <existing dict>() if not v%2}
* Decorators  
  *(See* ***Functions*** *section above)*
  + Adds features to “normal” funcs *(“SuperCharges” funcs)*
  + Relies upon the ability to pass & exec funcs as argos
  + Higher Order Function (HOC)
    - 1) Accepts & execs a func as an arg  
      OR  
      2) func which returns a func *(presumably the returned func is defined w/in the original func)*
    - Ex: map(), filter(), reduce()
  + Use \*args & \*\*kwargs to pass in variable # of args
* Error Handling
  + See [*Built-in Exceptions*](https://docs.python.org/3/library/exceptions.html) & [*Python Built-in Exceptions*](https://www.w3schools.com/python/python_ref_exceptions.asp)
  + Can “raise” (throw) one’s own errors - can be:
    - raise Exception()
    - raise *<specific Exception>*
  + Common paradigm: embed exception catchers w/in while True: block
* Generators
  + Iterating function
    - 1) Pauses w/in loop at yield statement
      * Generator “remembers” where it paused - waits till next
    - 2) *<action>* taken on paused data
    - 3) Resumes iteration at next statement
      * “Calling” entity (*caller of generator*) issues next
      * StopIteration exception occurs if too many next statements - i.e. exceeds the generator iteration limit
        + for loop auto-detects StopIteration exception
  + Create “poofs” of individual datum through a “range” of data *(!store all results in mem as do non-gen iterators - temp stores the current iterated value)*
  + All generators are iterables (*range()*), but !all iterables are generators (*<lists>*)
  + yield - pauses the func until next
  + Benefit: faster and less mem used
* Ternary Operator
  + Syntax:
    - [on\_true] if [expression] else [on\_false]
  + Ex:
    - print("A") if a > b else print("B")
    - min = a if a < b else b
* [*Modules*](https://docs.python.org/3/tutorial/modules.html)
  + Logical groupings
  + Each \*.py file is a module (main.py, utility.py, etc.)
    - \_\_name\_\_ == *<module name>* - if !the main module
  + Naming convention: snake-case - lowercase.py or lower\_case.py
  + Imported modularized files are cached into **\_\_pycache\_\_** dir
    - <*module\_file*>.py → *<module\_file>*.cpython-*<version>*.pyc
    - Faster loading of modules
    - Eliminate re-compiling the module @ runtime
  + [*Packages*](https://docs.python.org/3/tutorial/modules.html#packages)
    - Folder containing modules
    - Syntax: import <*package\_name*>.<*module\_name*>
    - Must contain an empty \_\_init\_\_.py file
      * Tells interpreter the dir is a package
    - Can be nested
  + [*Importing modules & packages*](https://docs.python.org/3/tutorial/modules.html#packages)
    - Recommendation is to be explicit
      * Can import all via “ \* “, but !recommended
    - *“Since the name of the main module is always "\_\_main\_\_", modules intended for use as the main module of a Python application must always use absolute imports.”*
    - [*Module Search Path*](https://docs.python.org/3/tutorial/modules.html#the-module-search-path)
  + \_\_name\_\_
    - Contains the name of the file (<*main*>.py or <*module*>.py)
      * <main>.py can be any name, but is the file which is exec’d
      * \_\_name\_\_ == *<module name>* - if !the main module
    - if \_\_name\_\_ == ‘\_\_main\_\_’:  
       <*main exec code*>
      * Commonly placed into main file
  + Built-in Modules
    - “Std. Lib” - installed w/Python by default
      * Maintained by python.org
      * Must be imported
    - random
    - sys
      * argv
        + 1st arg is filename
    - argparse
      * Default arg
        + -h, --help - only default arg argparse will process
        + Other args: error: unrecognized arguments: *<argname>*
    - subprocess
      * subprocess.run()
        + Exec subproc & wait for completion (CompletedProcess instance)
        + Execs subprocess.Popen() under the hood
        + Can capture stdin, stdout, stderr

Captured as bytes obj - doesn’t decode \n, \t, etc.

Raw ASCII chars

[*bytes*](https://docs.python.org/3/library/functions.html#func-bytes)*([<code>, <code>, …])*

bytes.decode()- decodes to “normal” text

* + - * subprocess.call()
        + Interface w/older versions (predecessor to subprocess.run())
    - os
      * os.mkdir(*<path>*) - creates dir
        + FileExistsError exception - if dir exists)

Can catch parent OSError error

* + - shutil
      * Archive functions
      * Dir & File functions
        + Copy functions (.copy(), .copy2(), .copyfile(), .copytree(), .rmtree()...)
        + Move functions (.move())
      * Misc functions
        + .chown(), .copystat(), .disk\_usage(), .get\_terminal\_size(), .which()
  + PyPi (Python Package Index)
    - Community-supplied packages
    - pip *(pip3)*
      * pip3 list - displays installed packages
        + setuptools - specifies how pkgs are installed

Replaced distutils

* + - * + wheel - pre-built pkgs - drop wheel file in specified dir in PYTHONPATH
        + egg - precursor to wheels
      * Installing modules (*Pycharm - Windows & Linux*)
        + File → Settings → Project:*<proj\_name*> → Project Interpreter → “+” → search for pkgs → Install Pkg
      * Installing modules (*pip command-line*)
        + pip3 install *<pkg\_name>*
        + pip3 uninstall *<pkg\_name>*
        + pip3 --user - --user !effect w/in venv
      * Packages
        + boto3 - interact w/AWS
      * pip3 freeze [> requirements.txt]
        + Can redirect to a requirements.txt file to track pkg depends
        + Install from pip freeze:

pip3 install -y -r requirements.txt

pip3 uninstall -y -r requirements.txt

Using a reqs.txt file forces uninstall of **all** dependencies

* + - pipenv
      * Replacement for pip - single CLI
        + Uses pip & venv under-the-hood
      * Pipfile - replaces requirements.txt
      * Pipfile.lock
      * Commands
        + pipenv shell - spawn a shell in a virtenv

Args:

--three - specify system’s python v3

--python $(which python3) - specify exact vers

pipenv install flask - !specific version

pipenv install flask==0.12.1 - specific vers

* + - * + pipenv
        + pipenv
  + Useful Built-in Modules
    - collections
      * Counter, defaultdict, OrderedDict  
        (**Note:** Python has made dicts [*ordered by default*](https://softwaremaniacs.org/blog/2020/02/05/dicts-ordered/en/) by insertion order)
    - datetime
      * time, date, TZs
    - time
    - array
      * array (vs. list)
        + array(<*typecode*>, <*initializer - e.g. a list*>)
        + Faster & less mem than lists

Pre-defined size & types vs. dynamic

* + - os
    - os.path
* Debugging
  + Linting
    - Pylint
    - pep8
  + IDE &/or editor
  + pdb Debugger
    - pdb.set\_trace()
      * Interactive python debugger
* File I/O
  + Functions:
    - ~~fh = open(<~~*~~fname~~*~~>, mode=’r’)~~ - works, but...
    - with open(<*fname*>, mode=*<mode>*) as fh:
      * Preferred method of opening file
      * Auto-closes file
      * open returns an iterable
        + Can be looped (for line in fh:)
    - Modes
      * Default mode == ‘r’
      * Write mode == ‘w’
        + Overwrites or creates new
      * Read/Write mode: ‘r+’
      * Append mode: ‘a’
    - <*fh*>.read() - reads entire file
    - <*fh*>.readline() - reads next line of file
    - <*fh*>.readlines() - reads entire file into a list
  + Reading line-by-line
    - for line in fh:
  + pathlib Module *(Python 3.4+)*
    - OS-specific semantics for representing directory hierarchies
* Regex
  + “re” module
    - Search syntax:
      * re.search(*<pattern>*, *<string>*, [*options*])
    - Outputs a “match” object
      * Use match funcs() to query search results
  + Patterns
    - Regex search patterns can be “compiled”, then used
      * pattern = re.compile(‘ab\*’)
        + Creates a pattern obj
      * pattern.match(‘*<string to match w/the pattern>*’)
      * pattern.findall(‘*<string to match w/the pattern>*’)
      * pattern.fullmatch(‘*<string to match w/the pattern>*’)
* Testing
  + unittest module
  + Steps:
    - 1) Create test.py file
    - 2) import file to be tested & unittest module
    - 3) Create a test class in test.py file - inherits from unittest.TestCase
      * Methods exec tested file’s funcs
        + Both confirm & break tested file’s funcs
        + Contains

self.assertEqual(<args>)

self.assertTrue(<arg>)

…

* + - * setUp() method
        + Execs prior to each func w/in the class
      * tearDown() method
        + Execs after each function w/in the class
* Scripting with Python
  + Image Processing
    - pillow package
      * Image module
        + *<image>*.format
        + *<image>*.size
        + *<image>*.mode
        + *<image>*.convert() - convert colors, formats, etc.
        + *<image>*.thumbnail() - creates thumbnail w/identical aspect
        + dir(*<image>*)
      * ImageFilter module
        + *<image>*.save()
        + *<image>*.show()
        + *<image>*.crop()
  + OpenCV
    - “Computer Vision” library
    - Focused on real-time processing (machine learning, etc.)
  + PDF Processing Package
    - PyPDF2 package
      * Opening PDFs - must use mode = ‘rb’  
        *(Read/Binary)*
      * reader obj - *PdfFileReader creates*
      * page obj - getPage() - *creates page obj*
      * writer obj
      * merger ojb - *PdfFileMerger() creates*
  + Email Processing
  + Password Checker
    - Lessons use [*Pwned Passwords*](https://haveibeenpwned.com/Passwords) & [*SHA1 Hash Generator*](https://passwordsgenerator.net/sha1-hash-generator/)
    - hashlib module
      * sha1 method
        + Encode password to utf-8
        + Gen hexdigest()
        + Convert to upper()
        + Ex: hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
    - Response from Pnwed Password
      * *<Matching hashes>* (from 1st 5 chars):*<# of times the hash was hacked>*
        + Must split @ ‘:’ using splitlines()
  + Twitter Bot
    - tweepy module
  + SMS Processing
    - Twilio:
* Web Scraping
  + First: use site’s API(s)
    - If !usable, use Web scraping
    - If large amts. of data, use Scrapy framework
  + “robots.txt”: pg. stating the info which should be scraped on a site.  
    *<url>*/robots.txt
    - Can’t prohibit, but is honor-system
    - User-Agent: \*
      * Default group - anyone not explicitly listed (e.g. google, bing, …)
  + Example Sites
    - [Hacker News](https://news.ycombinator.com/news) *(doesn’t have an API - just download html & process)*
      * Series: “Who’s Hiring?”
    - SWAPI - Star Wars API
    - JSONPlaceholder - testing/prototyping site
  + requests module - access sites & retrieve data
  + BeautifulSoup - parse/process retrieved html, xml, etc.
    - Parses & creates a “soup” obj
    - BeautifulSoup4
      * from bs4 import BeautifulSoup
* Web Development
  + Flask
    - Commands:
      * export FLASK\_APP=*<server.py filename>*
        + Notifies flask which file is ‘\_\_main\_\_’
        + Default port: 5000
      * export FLASK\_ENV=development
        + Enables updating *<server.py>* w/out restarting it
      * flask run - starts file in FLASK\_APP
        + Only accessible to localhost
      * flask run --host=0.0.0.0 - starts file in FLASK\_APP
        + Accessible from other systems on net
    - Jinja2- templating language
      * Processes Python expressions {{ *<exp>* }}
    - URL Building
    - Favicon Processing
    - Variable Rules
      * <variable\_name>
      * Ex: @app.route(‘/user/<username>’)
    - Form Data
      * flask.request.*<methods()>*
      * request.to\_dict() - POST’d fields to a dict
    - Redirect
      * flask.redirect.*<methods()>*
    - Exercise: Web server (in vm): [*http://10.0.0.226:5000*/](http://10.0.0.226:5000/)
  + Jinga
    - Templating language used under-the-covers by Flask
    - {{...}} = Flask evals a Python statements/expressions
* Testing & Automation
  + Selenium
    - Package for exec’ing browser instance
      * Drives browser - reqs chromedriver.exe file in same dir as Python driver app
      * Update Chrome: chrome://settings/help
    - [Selenium Easy](https://www.seleniumeasy.com/test/)
      * Test site for testing code
      * “Waits” - simulate human Web site interaction timings
* Machine Learning
  + Definition:  
    *‘Machine creating its own function, based upon inputs & outputs‘*
  + ML Types
    - Supervised: *input (test) data is classified*
      * Outputs are already known
      * Ex: stock-price predictions
      * Classification & Regression
    - Unsupervised: *input data is !classified*
      * Machine must create the groupings based upon data pts.
      * Outputs are !known
      * Subtypes:
        + Clustering
        + Association / Rule Learning
    - Reinforcement
      * Teaching machines thru tria & error (real-time learning)
        + Learns via millions of executions
      * Rewards & punishment
      * Ex: Machine learns how to play game via iterations
      * Goals:
        + Skill acquisition
        + Real-time learning
  + ML Steps
    - 1) Import the data
    - 2) Clean the data
    - 3) Split data. Training Set / Test Set
    - 4) Create a model
      * Import appropriate module, then select alg
    - 5) Check the output
    - 6) Improve
  + Tools
    - Kaggle: community of ML experts
      * Free access to massive amts of data
    - NumPy: working w/lists & arrays (incl multi-dimensional arrays)
    - pandas: working w/tabular data (csv files)
      * Data frame - transforms using data frames
      * Data analysis - manipulate data in rows & cols
    - scikit-learn: create a model
      * Contains pre-built algs.
    - matplotlib: charting lib
      * Visualize data
    - seaborn: statistical data visualization
      * Runs on top of matplotlib
    - Bokeh: interactive visualization library
    - jupyter notebooks: step thru each data analysis stage & track steps & display visuals
      * Access via Anaconda installation
  + Jupyter Notebooks
    - In notebooks: click Run or Shift-Enter to exec Python
    - Data in a previous cell is available in subsequent cells
      * Ex: data\_frame = pd.read\_csv('data.csv')
        + data\_frame is avail in subsequent cells