**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Python**  **Built-in**  **data.**  **objects** | | **Goal:** use Python [built-in objects](https://docs.python.org/3/library/stdtypes.html#built-in-types) to manipulate data better than a spreadsheet and frame like a hammer.   * Why? Spreadsheets are second tier tools vs. data objects providing long-term flexibility and sustainability.   + Object data manipulating skills makes you more agile and confident with data in any form from anywhere.   + Data transformer skills with lists, tuple, string, etc improves agility skills to combine, sort, and do work now.   + These concepts help perform system design and analysis, expedite project planning, and perform solve.IT analysis work. | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Mechanics**  Mylist/mytuple = [ 'a', 'b', 'c', 10,  20,if.tuple ]   1. iterator/<index>[i] 0 1 2 3 4 2. len(mylist) |-> <-| n=5 | | | | **Description**  Create the data for list, tuple, etc   1. iteration or count; index[i] or position # 2. len() inherits total items from an object 3. iterator <**for** i in mylist=>extracts data/index | | |  |
| [**Lists**](https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range) **= []**   * organize similar\dissimilar information * mutable! (.append() ~.remove() ~.pop) * sequential with an ID# per position * contain string, list, dict., etc   mylist = ['bambam', "a+b=c", 2\_0j, [1,2,3]]  for i in mylist: print(i)  bambam, a+b=c, 20j, [1, 2, 3]  comprehension places formula before iterator to generate data  mylist =[i\*2 for i in range(0,4) ]; mylist  [0, 2, 4, 6]  mytuple = (0,1,3,4)  mylist = [i\*3 for i in mytuple]; mylist  [0, 3, 9, 12]  me1 = ['adam','carly','jackson','danny']  dict(enumerate(me1,start=100))  100:'adam',101:'carly',102:'jackson',103: 'danny'}  mylist\_values[0] => object slicing  mylist\_values[1] => grab data position 1  data pack / unpack  for i mylist[1]: newlist.append[i]  for i in mylist: print(i) a b c 10 20 .  essential-list.methods  .[append()](https://docs.python.org/3/library/stdtypes.html#common-sequence-operations)  .[pop()](https://docs.python.org/3/library/stdtypes.html#common-sequence-operations)  **.**[**remove()**](https://docs.python.org/3/tutorial/datastructures.html#data-structures)  **.**[**sort()**](https://docs.python.org/3/tutorial/datastructures.html#data-structures)  #=> all -> [built.in functions](https://docs.python.org/3/library/functions.html) | | | [**Tuples**](https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range) **= (a,b,)**   * immutable w sequential ID[x] per position * immutable! can’t add/substract data * use to reference other object data * need a trailing comma!=>(1,2,) * use type(object) to discern   mytuple = ('snhu', 2+0j, [1,2,3],)  type(mytuple)  ('snhu',(2+0j),[1,2,3]) **#note diff.data types!**  tuple  mytuple = (1,2,3,)  mytuple + mytuple **#note d**  (1, 2, 3, 1, 2, 3)  [Common Sequence Operations](https://docs.python.org/3/library/stdtypes.html#common-sequence-operations)    .enumerate() | | [**Dictionary**](https://docs.python.org/3/tutorial/datastructures.html#data-structures) **= { key:value }**   * essential for pairing related data * go-to-tool for real-world modeling * keys **immutable**, values=mutable * dict would reference your unique ID and an associated list would have the characteristic data in * returns data unordered & random   mydict= {'key\_1':['value\_1'],'key1':(1,2,3,)}  { 'key\_1':['value\_1'], 'key1':(1, 2, 3) }  if  mydict = dict(key\_1= [1,2,'z'])  mydict  {'key\_1': [1, 2, 'z']}  keytuple = ('customer\_name','age')  valuelist = [['john','doe'],[35,76]]  dict(zip(keytuple,valuelist))  {'customer\_name':['john','doe'],'age':[35, 76]}  .keys(), .values(), .items()=>  mydict={'key\_1':['value\_1'],'key2':(1,2,)}  for k,v in mydict.items(): print(mydict.keys(), mydict.values())  dict\_keys(['key\_1', 'key1']) dict\_values([['value\_1'], (1, 2)]) **#top keys,bottom value**  dict\_keys(['key\_1', 'key1'])  dict\_values([['value\_1'], (1, | | |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Python**  **Built-in**  **data.**  **objects** | **Goal:** use Python [built-in objects](https://docs.python.org/3/library/stdtypes.html#built-in-types) to manipulate data better than a spreadsheet and frame like a hammer.   * Why? Spreadsheets are second tier tools vs. data objects providing long-term flexibility and sustainability.   + Object data manipulating skills makes you more agile and confident with data in any form from anywhere.   + Data transformer skills with lists, tuple, string, etc improves agility skills to combine, sort, and do work now.   + These concepts help perform system design and analysis, expedite project planning, data uploading, and storage design. | | |
| --- | --- | --- | --- |
| **Mechanics**   1. mystring = 'python training is fun ' 2. index [i] 012345.............................23 3. len(mylist) |-> <-| **n=23**  * slicing mystring[10:] >>> 'ining is fun ' | |  |  |

| **Strings = 'abc** **'**   | **w** | **e** | **i** | **r** | **d** | | --- | --- | --- | --- | --- | | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |  * text processors quotes =! python quotes * strings factilate text and natural language processing. * a whole book in a single string   fruit = 'apple'  i = 0  myL = []  while i < len(fruit):  letter = fruit[i]  myL.append(letter)  i = i + 1  myL  ['a', 'p', 'p', 'l', 'e']  [**set(), frozenset()**](https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset)   * A set object is an unordered collection of distinct [hashable](https://docs.python.org/3/glossary.html#term-hashable) objects. * Use removing duplicates\test if have ID * Compute difference in 2 data sets: union intersection, difference, symmetric diff * [Hashability](https://docs.python.org/3/glossary.html#term-hashable) makes an object usable as a dictionary key and a set member/ | [**set(), frozenset()**](https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset)  mylist = ['a', 'p', 'p', 'l', 'e']  myset = set(mylist); myset  {'a', 'e', 'l', 'p'}    'a' in myset True ; a' not in myset False  {c for c in 'abracadabra' if c not in 'abc'}  {'d', 'r'}  # Set Operations  set1 = {'apple', 'banana'}; set2 = {'orange', 'banana'}  union\_set = set1.union(set2) # Union  print("Union:", union\_set) # Output: {'banana', 'apple', 'orange'}  intersection\_set = set1.intersection(set2) # Intersection  print("Intersection:", intersection\_set) # Output: {'banana'}  difference\_set = set1.difference(set2) # Difference  print("Difference:", difference\_set) # Output: {'apple'}  sym\_diff\_set = set1.symmetric\_difference(set2) # Symmetric Diff  print("Symm Diff:", sym\_diff\_set) # Output: {'apple', 'orange'}  # Set Comprehensions - Create from a string; exclude characters  myset = {c for c in 'abracadabra' if c not in 'abc'}  print("Set Comprehension:", myset) # Output: {'r', 'd'}  id\_list = [102, 101, 104, 102, 101]# Removing duplicate IDs  unique\_ids = set(id\_list)  print("Unique IDs:", unique\_ids) # Output: {101, 102, 104}  id\_set = {101, 102, 104} # Checking if an ID exists in data  print("ID 103 exists:", 103 in id\_set) # Output: False  # Set Comprehensions - Create set from string and exclude characters  myset = {c for c in 'abracadabra' if c not in 'abc'}  print(myset) # Output: {'r', 'd'} | [pandas](https://pandas.pydata.org/) [series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html) and [dataframe](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html)  import pandas as pd  df = pd.DataFrame(data) # 1. Create df  df = pd.read\_csv('file.csv') # 2. read csv into df  df.to\_csv('file.csv') # 3. Write df to .csv  df.head(); df.tail()  df['col\_name']  df[['column1', 'column2']]  df[df['column'] > value]  df['column'].apply(lambda x: function(x))  df.drop('column\_name', axis=1)  df.drop(df[df['column'] > value].index)  df.groupby('column').mean()  pd.merge(df1, df2, on='common\_column')  pd.concat([df1, df2])  df.sort\_values(by='column')  df.reset\_index(drop=True)  df.set\_index('column', inplace=True)  df.fillna(value)  df.dropna()  df['column'].astype('datatype')  df.rename(columns={'old\_name': 'new\_name'})  df.iloc[index]  df.loc[label]  df.dtypes  df.describe()  df['column'].unique()  df['column'].nunique()  df['column'].value\_counts()  df.isnull() |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7** 

| **Conditional**  **Statements** | [**Built-in Types**](https://docs.python.org/3/library/stdtypes.html#built-in-types)  Built-in types are truth testing logic using boolean, comparisons, (+,-,/,//,%)  Conditionals are the testing logic to evaluate whether sometime is True or False | |  |
| --- | --- | --- | --- |
| [**Boolean**](https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not) **– and, or, not**  These are the Boolean operations, ordered by ascending priority:    Notes:   1. This is a short-circuit operator, so it only evaluates the second argument if the first one is false. 2. This is a short-circuit operator, so it only evaluates the second argument if the first one is true. 3. not has a lower priority than non-Boolean operators, so not a == b is interpreted as not (a == b), and a == not b is a syntax error. | | [**Comparisons**](https://docs.python.org/3/library/stdtypes.html#comparisons)  There are eight comparison operations in Python. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, x < y <= z is equivalent to x < y and y <= z, except that *y* is evaluated only once (but in both cases *z* is not evaluated at all when x < y is found to be false). | |
| [**Numeric Type operations**](https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex)  Use constructors [int()](https://docs.python.org/3/library/functions.html#int), [float()](https://docs.python.org/3/library/functions.html#float), and [complex()](https://docs.python.org/3/library/functions.html#complex)to product specific #s | | 1. [Built-in data types](https://docs.python.org/3/library/stdtypes.html#built-in-types)    1. [boolean](https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not) <True\False>: and, or, not    2. [comparators](https://docs.python.org/3/library/stdtypes.html#comparisons): <, <=, >, >=, ==, !=, is, is not    3. python’s [data model](https://docs.python.org/3/reference/datamodel.html#data-model) <good read!> 2. [Built-in function](https://docs.python.org/3/library/functions.html) | |
|  | |  | |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Iterators**   * **for** * **range** * **while** | * Iteration is the act of looping instructions repeatably * instructions continuously execute until False or termination * such as an end of range, conditional is != * most efficient means to cycle data in lists,tuples,ranges,etc * Iterators are sequential like 0->1->2->3, and may step >1 | |  |
| --- | --- | --- | --- |
| **Mechanics**   1. mylist = [ ‘a’, ‘b’, ‘c’, 10, ] 2. iterator/index [i] 0 1 2 3 3. len(mylist) |-> <-| n=4 4. **print( mylist[i]\*3) aaa bbb ccc 30** 5. negative index [i] -4 -3 -2 -1   for i **in** mylist: print(mylist[i]\*3) | | **Mechanics Description**   * create the data for list, tuple, etc  1. iteration is the count; index is the position 2. len() inherits count of total items from mylist 3. for i **in** mylist:   print(mylist[i]\*3) #multiply each list iterate \*3   1. negative index is neg. number values for an sequence position | |

| **for i in <object>:**   * starts from 0 for all items in the object * inherits length from object * i shorthand for iterator * regularly combined with conditional statements to make decisions if-elif-else   mylist = [1,4]  for i in mylist:  print(i\*3)  3, 12  from math import log10  def myfunction(x):  return log10(x)  for i in range (2,4,1):  print("loop#{a}, value={b}".  format(a=i,b=(round(myfunction(i),2))))  loop#2, value=0.3  loop#3, value=0.48  myL = [1,2,3]  data = (round(myfunction(i),3) for i in myL)  print(list(data))   * [0.0, 0.301, 0.477] | **while i <= <value/object>:**   * use to iterate in a forward or reverse direction * slash breaks code to next line   i = 0  mylist = [] #add result to list  while i <=1:  mylist.append(i); i +=1  mylist  [0, 1]  i=1 #loop+print custom results  while i < 2:  print("loop# i={}".format\ (str(i)))  i +=1  print("final loop i is ="+str(i))  loop# i=1  final loop i is =2 | [**range**](https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range)**(start,stop,step)**   * use set a numeric range to iterator or calculate with * default start is zero and default setp is one * may inherit values form use objects, attributes   for i in range(0,2): print(i)  **0 . , 1 .**  me1=('adam','carly','jackson','danny')  for i in range(len(me1)): print(i)  0, 1,2 ,3  #see data transposition slide  me1 = ['w','e','i','r','d']  me2 = [] # (+) indexing  for i in range(0,5):  me2.append(me1[i])  ['w', 'e', 'i', 'r', 'd']  me1 = ['d','r','i','e','w']  me2 = [] # (-) indexing  for i in range(1,6):  me2.append(me1[-i])  ['w', 'e', 'i', 'r', 'd'] | **Misc**   * row for row in open (‘filepath.txt’) * generator <fix this>   sum((i\*3 for i in range(2))  with open (‘path of file.txt’, ‘r’) as data\_file:  for line in data\_file:  print(line)  -Quickly create lists or dict with-  [enumerate()](https://docs.python.org/3/library/functions.html#enumerate)adds list index #  me1 = ['adam','carly','jackson','danny']  me2 = list(enumerate(me1)); me2  [(0, 'adam'), (1, 'carly'), (2, 'jackson'), (3, 'danny')] |
| --- | --- | --- | --- |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Functions** | Functions are the workhorses helping transform, transpose, combine and just about anything else you can think of | |  |
| --- | --- | --- | --- |
| * each function has unique parameters (values it accepts) and means of operating. To figure out read the docs and when necessary look for examples on stackoverflow, jupyterform, and google but try to be selective so your time is not wasted | | dir() shows an object’s director with all constructors and methods. Use it often to learn.  dir(mylist)= | |
| | [**Built-in Functions**](https://docs.python.org/3/library/functions.html#built-in-functions) | | | | --- | --- | --- | | A  [abs()](https://docs.python.org/3/library/functions.html#abs)  [aiter()](https://docs.python.org/3/library/functions.html#aiter)  [all()](https://docs.python.org/3/library/functions.html#all)  [any()](https://docs.python.org/3/library/functions.html#any)  [anext()](https://docs.python.org/3/library/functions.html#anext)  [ascii()](https://docs.python.org/3/library/functions.html#ascii)  B  [bin()](https://docs.python.org/3/library/functions.html#bin)  [bool()](https://docs.python.org/3/library/functions.html#bool)  [breakpoint()](https://docs.python.org/3/library/functions.html#breakpoint)  [bytearray()](https://docs.python.org/3/library/functions.html#func-bytearray)  [bytes()](https://docs.python.org/3/library/functions.html#func-bytes)  C  [callable()](https://docs.python.org/3/library/functions.html#callable)  [chr()](https://docs.python.org/3/library/functions.html#chr)  [classmethod()](https://docs.python.org/3/library/functions.html#classmethod)  [compile()](https://docs.python.org/3/library/functions.html#compile)  [complex()](https://docs.python.org/3/library/functions.html#complex)  D  [delatr()](https://docs.python.org/3/library/functions.html#delattr)  [dict()](https://docs.python.org/3/library/functions.html#func-dict)  [dir()](https://docs.python.org/3/library/functions.html#dir)  [divmod()](https://docs.python.org/3/library/functions.html#divmod) | E  [enumerate()](https://docs.python.org/3/library/functions.html#enumerate)  [eval()](https://docs.python.org/3/library/functions.html#eval)  [exec()](https://docs.python.org/3/library/functions.html#exec)  F  [filter()](https://docs.python.org/3/library/functions.html#filter)  [float()](https://docs.python.org/3/library/functions.html#float)  [format()](https://docs.python.org/3/library/functions.html#format)  [frozenset()](https://docs.python.org/3/library/functions.html#func-frozenset)  G  [getattr()](https://docs.python.org/3/library/functions.html#getattr)  [globals()](https://docs.python.org/3/library/functions.html#globals)  H  [hasattr()](https://docs.python.org/3/library/functions.html#hasattr)  [hash()](https://docs.python.org/3/library/functions.html#hash)  [help()](https://docs.python.org/3/library/functions.html#help)  [hex()](https://docs.python.org/3/library/functions.html#hex)  I  [id()](https://docs.python.org/3/library/functions.html#id)  [input()](https://docs.python.org/3/library/functions.html#input)  [int()](https://docs.python.org/3/library/functions.html#int)  [isinstance(](https://docs.python.org/3/library/functions.html#isinstance)  [issubclass()](https://docs.python.org/3/library/functions.html#issubclass)  [iter()](https://docs.python.org/3/library/functions.html#iter) | L  [len()](https://docs.python.org/3/library/functions.html#len)  [list()](https://docs.python.org/3/library/functions.html#func-list)  [locals()](https://docs.python.org/3/library/functions.html#locals)  M  [map()](https://docs.python.org/3/library/functions.html#map)  [max()](https://docs.python.org/3/library/functions.html#max)  [memoryview()](https://docs.python.org/3/library/functions.html#func-memoryview)  [min()](https://docs.python.org/3/library/functions.html#min)  N  [next()](https://docs.python.org/3/library/functions.html#next)  O  [object()](https://docs.python.org/3/library/functions.html#object)  [oct()](https://docs.python.org/3/library/functions.html#oct)  [open()](https://docs.python.org/3/library/functions.html#open)  [ord()](https://docs.python.org/3/library/functions.html#ord)  P  [pow()](https://docs.python.org/3/library/functions.html#pow)  [print()](https://docs.python.org/3/library/functions.html#print)  [property()](https://docs.python.org/3/library/functions.html#property) | R  [range()](https://docs.python.org/3/library/functions.html#func-range)  [repr()](https://docs.python.org/3/library/functions.html#repr)  [reversed()](https://docs.python.org/3/library/functions.html#reversed)  [round()](https://docs.python.org/3/library/functions.html#round)  S  [set()](https://docs.python.org/3/library/functions.html#func-set)  [setattr()](https://docs.python.org/3/library/functions.html#setattr)  [slice()](https://docs.python.org/3/library/functions.html#slice)  [sorted()](https://docs.python.org/3/library/functions.html#sorted)  [staticmethod()](https://docs.python.org/3/library/functions.html#staticmethod)  [str()](https://docs.python.org/3/library/functions.html#func-str)  [sum()](https://docs.python.org/3/library/functions.html#sum)  [super()](https://docs.python.org/3/library/functions.html#super)  T  [tuple()](https://docs.python.org/3/library/functions.html#func-tuple)  [type()](https://docs.python.org/3/library/functions.html#type)  V  [vars()](https://docs.python.org/3/library/functions.html#vars)  Z  [zip()](https://docs.python.org/3/library/functions.html#zip)  \_[\_\_import\_\_()](https://docs.python.org/3/library/functions.html#import__) | | | abs(-1) = 1   * bool() -> always True, unless object is empty, like [], (), {},False, 0, None * chr(97)->a. returns string unicode character, chr(100)->d * dict()-> create a dict from object, mydict(mylist) * dir() if object has \_\_dir\_\_ returns list of attributes * divmod(numerator,denominator), result=(quotient,remainder) * x=[‘a’,’b’]->list(enumerate([x])) -> [(0, 'a'), (1, 'b')]   returns an iterable tuple object   * float(1) -> 1.0 * .format customize output, print("{a}".format(a=1.01))-> 1.01 * frozenset() -> immutable set * help() details on any function or object, help(set()) * int() -> cast to integer; x = “1”, chr(x) = 1 * isinstance()->tests if in a class * len() essential function! # items inside or across object * list()->create-> mytuple=1,2,;mylist(mytuple)->[1,2] * isinstance() -> x =”me”, isinstance(me,str) -> True * min(0,3,4) -> 0; max(0,3) -> 3 * range(start,stop,start)->for i in range(0,10,2):print(i)-> 0,2,4,6,8 * round(1.5) -> 2 * set()->create->only unique values;mutable|x=1,1,1;set(x)->{1} * slice(start,end,step)-> a=('a','b',11); x=slice(1,3); print(a[x])->('b',11) * sorted()-> * sum()-> a=100,1; sum(a)->101 * tuple()-> create -> mylist=['a',1]; tuple(mylist)->(‘a’,1) * type() -> what object is it? type(tuple())-> tuple * zip()->for item in zip([1, 2],['a','b']):print(item)->(1,'a')(2,'b'). | |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Classes are the axes** | 1. **Classes** are a framework for creating objects, functions specific to an object family, attributes, and child class via inheritance 2. **Objects** are entities that perform work. Child objects are instantiated from parents 3. **Methods** are instructions detailing “how” to perform work. Built parent or child level. 4. **Attributes** are alpha\numeric values associated with an object or class. Methods can use this values to perform work and make decisions 5. **self** <self.attribute> is the first argument in a class function self-identifying itself while processing instructions 6. **Function** – set of instructions to perform a task independent of any object. Methods are functions but associated with an object. | | . |
| --- | --- | --- | --- |
| mydict = {"training done":[], "total animals":0}  class myFarm: #create parent class object  pass  name = ""  species = ""  train = ""  def add\_train(traintype):#create a user function to count, sort  mydict["training done"].append(traintype)  mydict["total animals"] =+1  #-------------> #children instantiate from parents  a1 = myFarm() # instantiate children objects, a for animal  a2 = myFarm() # all object names are user defined   | #update attributes  a1.name = "mackenzie”  a1.species = "dog"  a1.train = "speak"  add\_train(a1.train) **#cheCK-OUT!** | <only here bc space>  a2.name = "vinny"  a2.species = "horse"  a2.train = "jumping"  add\_train(a2.train) | | --- | --- | | **function accepts attribute to update dictionary object** | |   #write a simple report using a dictionary data object format  mydict\_rpt = {a1.name:a1.species, a2.name:a2.species,"metrics=>":mydict}  mydict\_rpt  {'arnold': 'dog','vinny': 'horse','metrics=>': {'training done': ['catch', 'jumping'], 'total animals': 1}}  #use object’s constructors to view its contents  print(a1.\_\_dict\_\_,a2.\_\_dict\_\_)  {'name': 'arnold', 'species': 'dog', 'train': 'catch'} {'name': 'vinny', 'species': 'horse', 'train': 'jumping'}  #user functions built in or out of objects  def sum(a, b):  return (a + b)  a = int(input('Enter 1st number: '))  b = int(input('Enter 2nd number: '))  print(f'Sum of {a} and {b} is {sum(a, b)}'  Enter 1st number : 1  Enter 2nd number: 2  Sum of 1 and 2 is 3 | | class Animal: # Parent class  def \_\_init\_\_(self, species): # Constructor method  self.species = species # Attribute  def make\_sound(self): # Method  return f"The {self.species} makes a sound."  class Dog(Animal): # Child class inheriting from Animal  def \_\_init\_\_(self, name, species='Dog'):  super().\_\_init\_\_(species) # Call to parent constructor  self.name = name # Additional attribute specific to Dog  def bark(self): # Method specific to Dog  return f"{self.name}, the {self.species}, barks."  class Bulldog(Dog): # Subclass of Dog  def \_\_init\_\_(self, name, weight):  super().\_\_init\_\_(name, species='Bulldog') # Call to parent (Dog) constructor  self.weight = weight # Additional attribute specific to Bulldog  def growl(self): # Method specific to Bulldog  return f"{self.name}, the {self.species} weighing {self.weight}kg, growls."  def describe\_animal(animal): # Function independent of class  print(animal.make\_sound())  generic\_animal = Animal('Generic Animal') # Create objects from classes  buddy = Dog('Buddy'); max = Bulldog('Max', 30) # Bulldog Object; subclass of Dog  print(generic\_animal.make\_sound()) # Use methods of the objects  print(buddy.bark()); print(max.growl()) # Output from object  describe\_animal(max) # independent function   * Classes/Objects: Animal, Dog with generic\_animal, buddy * Inheritance: Dog inherits from Animal * Methods: Animal's make\_sound, Dog's bark * Attributes: species in Animal, name in Dog * self Keyword: Used in class methods * Functions/Methods: describe\_animal function, independent of class * Subclass: Bulldog from Dog * Constructor: Bulldog with \_\_init\_\_, weight attribute * Specific Method: Bulldog's growl; Subclass Object: max demonstrates Bulldog behavior | |

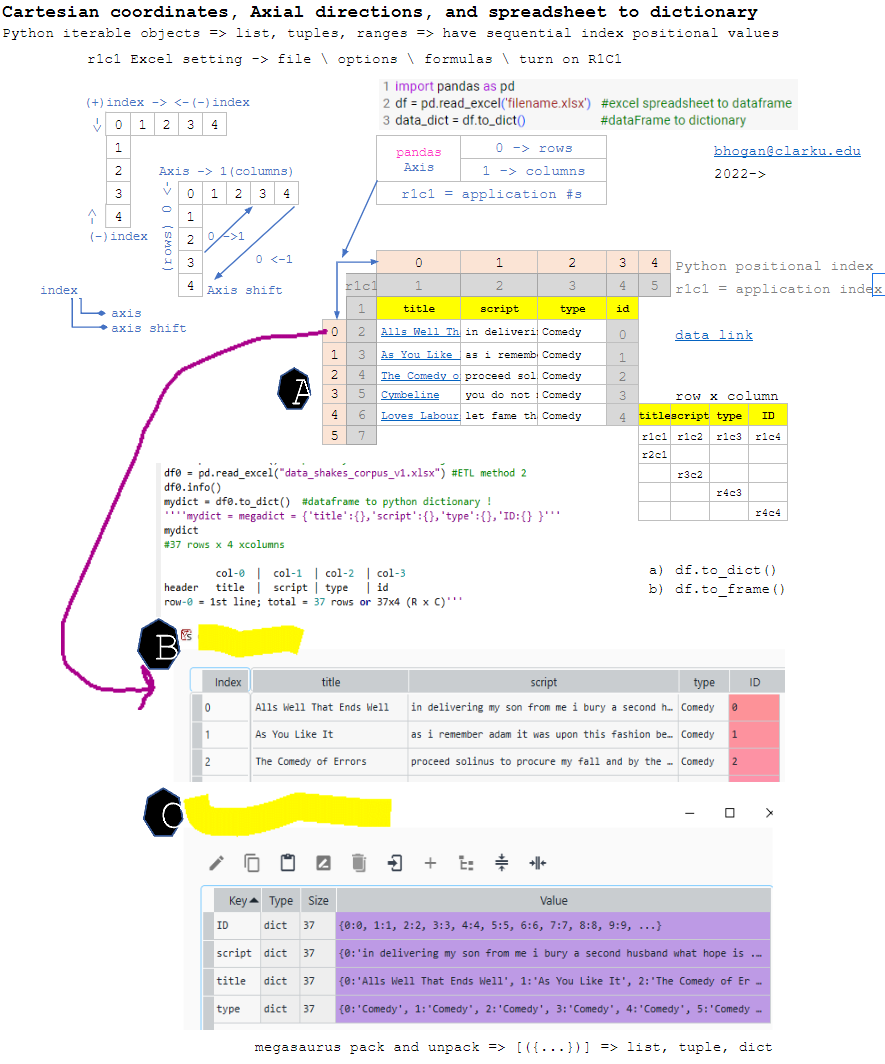
**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Transformers**  **&**  **pos/neg**  **indexing** | Moving data around is art and may require wizardry.  For starters master 2 dimensions, rows and columns, x and y like [cartesian coordinate system](https://en.wikipedia.org/wiki/Cartesian_coordinate_system)  Learn the basics of transposition  data moves left to right, right to left, &  top to bottom, bottom to top   * up\down, left\right. * down\up, right\left |  |
| --- | --- | --- |

| **Illustrates postive and negative sequential data indexing** | #Style 1 – left to right, right to left, top to bottom, bottom to top    #(+)index  me1 = ['w','e','i','r','d']  me2 = []  for i in range(0,5):  me2.append(me1[i])  me2  **#['w', 'e', 'i', 'r', 'd']**  #(-)index  me1 = ['d','r','i','e','w']  me2 = []  for i in range(1,6):  me2.append(me1[-i])  me2  **#['w', 'e', 'i', 'r', 'd']** |
| --- | --- |

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

|  | | |
| --- | --- | --- |

****

**>\_7.pillars.of.python classes1, conditionals2, data.objects3, functions4, iterators5, libraries6, transformers7**

| **Installation** |  | | * Warning <for less experienced it.minions * Take your time and read prompts | Critical source locations  [Python Package Index](https://pypi.org/project/pip/) = source repository of Python |
| --- | --- | --- | --- | --- |

**Upgrading your Jupyter labs to use share doc feature**

* <https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html>
* [Python Package Index](https://pypi.org/project/pip/) = source repository of Python software (<https://pypi.org/>)

| **Task** | | **Instructions** |
| --- | --- | --- |
| Using terminal\ command line   1. upgrade pip <[installation engine](https://pypi.org/project/pip/))    1. <https://pypi.org/project/pip/>    2. this installs pip-22.2.2 | | C:\users\17574\anaconda3\python.exe -m pip install --upgrade pip |
| 1. upgrade jupyter notebooks    1. done on command line either conda or pip | | command line:  conda install -c conda-forge jupyterlab |
| 1. add the share notebook feature    1. [github source](https://github.com/jupyterlab-contrib/jupyterlab-link-share)    2. <https://github.com/jupyterlab-contrib/jupyterlab-link-share> | | command line:  pip install jupyterlab-link-share |
| Open jupyter notebook  I GET THERE USING Anaconda Prompt  #will then open and run in browswer | | cL\Users\<your\_computer\_name>jupyter-lab |
|  | Text  Description automatically generated | |