Python Built-in Objects

Goal: use Python built-in objects 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 finding missing info.

Mechanics

```
1. mylist,mytuple  = [    'a',   'b',   'c',   10,    20,    ]
a. iterator/index [i]      0     1     2      3       4
b.        len(mylist)      |->                    <-| n=5
c. print( mylist[i])    'a'      'b'    'c'    10     20
```

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 = []
- 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]
.
```

Functions

.append()
.pop()
.remove()

Tuples = (a,b,)
- immutable w sequential ID[x] per position
- immutable! can't add/substract data
- practical reference table to other data
- need a trailing comma!=>(1,2,)
- use type(object) to know what it is

```
mytuple = ('snhu', 2+0j, [1,2,3],)
type(mytuple)
('snhu',(2+0j),[1,2,3])   #note diff.data type
s!
tuple

mytuple = (1,2,3,)
mytuple + mytuple  #note d
(1, 2, 3, 1, 2, 3)
```

Object Operations

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

Dictionary = { 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]}
```

Functions

```
.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([['val
ue_1'], (1, 2)])  #top keys,bottom value
dict_keys(['key_1', 'key1']) dict_values([['val
ue_1'], (1,
```

Core Python Objects – Part I of II: lists, tuples, dictionary, strings, sets, series, dataframes Objects

Python Built-in Objects

Goal: use Python built-in objects 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 finding missing info.

Mechanics

1. mystring = 'python training is fun '
2. index [i] 012345.....23
3. len(mylist) |-> <-| **n=23**

- slicing mystring[10:] >>> 'ining is fun '

Description

- <pending>

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 may be 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()

- A set object is an unordered collection of distinct hashable objects.
- Use removing duplicates\test if have ID
- Compute difference in 2 data sets: union intersection, difference, symmetric diff
- Hashability makes an object usable as a dictionary key and a set member/

```
mylist = ['a', 'p', 'p', 'l', 'e']
myset = set(mylist); myset
{'a', 'e', 'l', 'p'}

'a' in myset    | 'a' not in myset
True                    False
{c for c in 'abracadabra' if c not in 'abc'}
{'d', 'r'}
```

pandas series and dataframe

import pandas as pd

Conditional Statements

Built-in types are truth testing logic using boolean, comparisons, (+,-,/,//,%)
Conditionals are the testing logic to evaluate whether sometime is True or False

Boolean - and, or, not

These are the Boolean operations, ordered by ascending priority:

Operation	Result	Notes
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)

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

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

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Numeric Type operations

Use constructors `int()`, `float()`, and `complex()` to product specific #s

Operation	Result	Notes
`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*	(1)
`x % y`	remainder of `x / y`	(2)
`-x`	*x* negated	
`+x`	*x* unchanged	
`abs(x)`	absolute value or magnitude of *x*	
`int(x)`	*x* converted to integer	(3)(6)
`float(x)`	*x* converted to floating point	(4)(6)
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	(6)
`c.conjugate()`	conjugate of the complex number *c*	
`divmod(x, y)`	the pair `(x // y, x % y)`	(2)
`pow(x, y)`	*x* to the power *y*	(5)
`x ** y`	*x* to the power *y*	(5)

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,   ]
1. iterator/index [i]    0      1      2      3
2.         len(mylist)    |->                 <-|   n=4
3. print( mylist[i]*3)   aaa    bbb    ccc    30
4. 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
  4. 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(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>

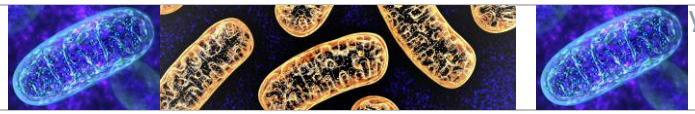
```
sum((i*3 for i in range(2))

with open ('path of file.txt',
'r') as data_file:
  for line in data_file:
    print(line)


-Quickly create lists or dict
with-
enumerate() adds list index #
me1 =
['adam','carly','jackson','danny']
me2 = list(enumerate(me1)); me2
```
[(0, 'adam'), (1, 'carly'), (2, 'jackson'), (3, 'danny')]

Essetial 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

A
abs()
aiter()
all()
any()
anext()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delatr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

__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')

..

Objects – the Actors, "memory, agent starline, is what i have instead of a view" hannibal

Building your own Object 'class'

a. **Classes** are a framework for creating objects, functions specific to an object family, attributes, and child class via inheritance
b. **Objects** are entities that perform work. Child objects are instantiated from parents
c. **Methods** are instructions detailing "how" to perform work. Built parent or child level.
d. **Attributes** are alpha\numeric values associated with an object or class. Methods can use this values to perform work and make decisions
e. **self** <self.attribute> is the first argument in a class function self-identifying itself while processing instructions
f. **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	<only here bc space>
a1.name = "mackenzie"	a2.name = "vinny"
a1.species = "dog"	a2.species = "horse"
a1.train = "speak"	a2.train = "jumping"
add_train(a1.train) #cheCK-OUT!	add_train(a2.train)
function accepts attribute to update dictionary objec	

```
#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
```

Constructor example

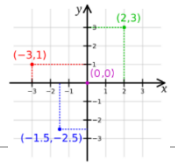
subclass example

Transposition – Part 1 of 2: left-hand side to right-hand side, RHS->LHS, top to bottom & bottom to top

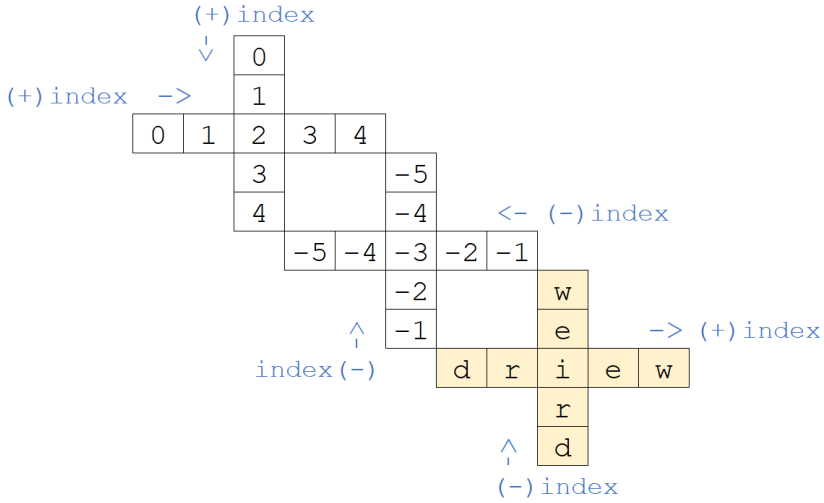
Data Transform pos/neg indexing

Moving data around is art and may require wizardry.
For starters master 2 dimensions, rows and columns, x and y like cartesian coordinate system
Learn the basics of transposition

- up\down, left\right.
- down\up, right\left



Illustrates postive and negative sequential data indexing



(+) 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']
```

#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

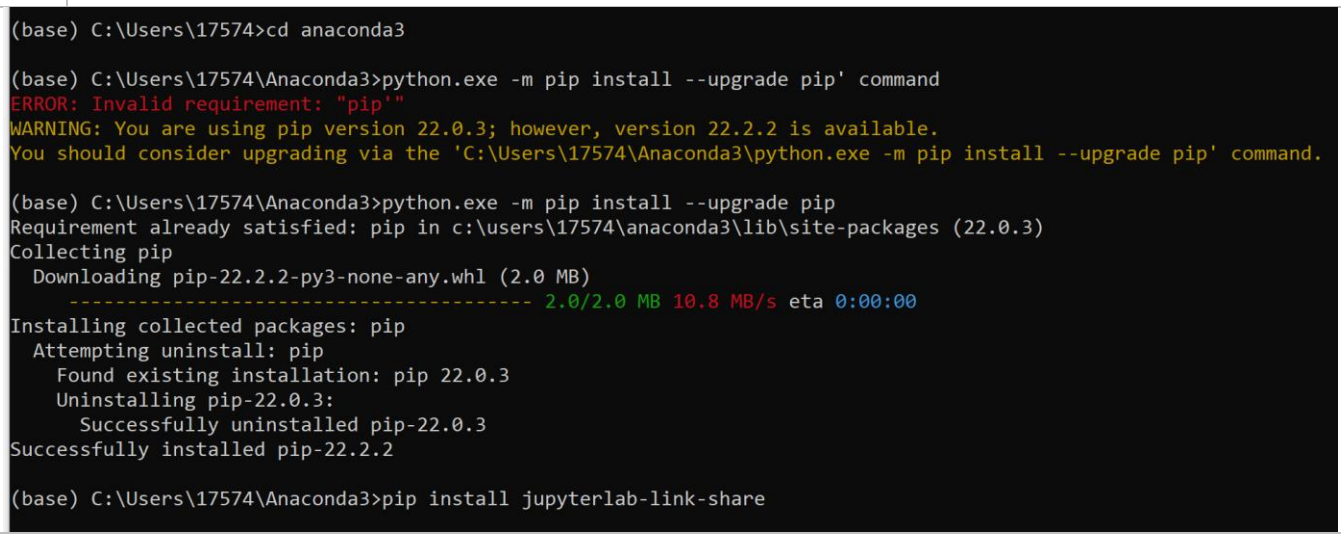
```
me1 = ['d','r','i','e','w']
me2 = []
for i in range(1,6):
    me2.append(me1[-i])
me2
#['w', 'e', 'i', 'r', 'd']
```

Installation	•	• Warning <for less experienced it.minions • Take your time and read prompts •	Critical source locations Python Package Index = source repository of Python
	Mechanics		Description

Upgrading your Jupyter labs to use share doc feature

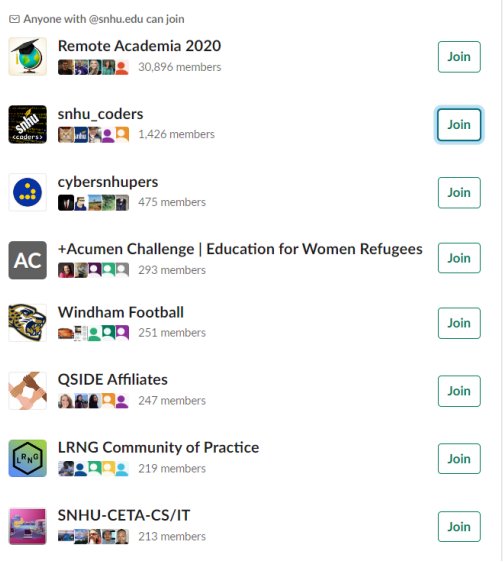
- https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html
- Python Package Index = source repository of Python software (https://pypi.org/)

Task	Instructions
Using terminal\ command line 1) upgrade pip <installation engine) a. https://pypi.org/project/pip/ b. this installs pip-22.2.2	C:\users\17574\anaconda3\python.exe -m pip install --upgrade pip
2) upgrade jupyter notebooks a. done on command line either conda or pip	command line: conda install -c conda-forge jupyterlab
3) add the share notebook feature a. github source b. 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



Fun with formatting

Lists	Tuples	Dictionary	Strings
• tbd • me1 = ['adam','carly','jackson','danny'] for i, person in enumerate(me1): print("{}st position is {}".format(i+1,person)) 1st position is adam 2st position is carly 3st position is jackson 4st position is danny	mytuple=		



future topics
randomness and variability
what libaries are and how to use them Pypi