

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\disimilar 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 = formula before iterator

```
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]
```

Tuples = (a,b,)

- immutable w sequential ID[x] per position
- **immutable!** can't add/substruct 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)
```

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]}
```

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]	lth 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

Function

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

Function

```
.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,
```