BIG DADDY'S

Data Container Quick Chart

(Attributes, Methods, Processes, Workarounds)

	Lists (sequence type, mutable)		**Tuples** (sequence type, immutable)		**Dictionaries** (maped type, mutable)		**Sets** (set type, mutable but no mutable collections)	
ACTION/EVENT	Key Term/Symbol/Method/Attribute	Example	Key Term/Symbol/Method/Attribute	Example	Key Term/Symbol/Method/Attribute	Example	Key Term/Symbol/Method/Attribute	Example
Create initially	[item, item,...]	LPreps=["of", "over","with","in","under"]	(items) ~note example tuple objects~; ("1item",) must have comma; value trailing comma issue	TupPreps=('with','with','under', ['on', 'of', 'in'], 'down', ('by', 'at'), 'among', 'about')	{key:value, key:value, ….}	DPreps={"of":"coming from", "over":"on top of","with":"accompanying"}	Setname = set (sequential item collection) ~no duplicates, unordered~ set([1,2,3,4])~	SPreps = set(LPreps) or myset = "Bannanas are nice" ↳ {'s', ' ', 'n', 'B', 'a', 'r', 'i', 'e', 'c'}
		Numlist=(47,3,12.23,96,52)		TupNum=(4,2.4, 7/3, 1,20.2) or Testtup=('a','b','c')	Newdict = dict.fromkeys(keys [,values])	Dprep2 = dict.fromkeys(LPreps,"hold")	Setname = set() <-**empty;** or ={item,item,…}	Pets = {'dog','cat','fish'}
					-	~Note: no duplicate keys allowed~	can be built with set comprehensions	a = {x for x in 'abracadabra' if x not in 'abc'}
Combine/merge containers	~concatenation~	newlist=list1 + list2	~concatenation~	newtup=tup1+tup2	dict.update(dict2) ~adds dict2 to dict1~	D1.update(D2) ~also D1.update(1=dog, 2=cat)~	.union(set2) or .update(set2) ~unique items~	newset=set1.union(set2)
Combine overlap items only	N/A		N/A		N/A		.intersection(set2)	newset=set1.intersection(set2)
Combine non-overlap items only	N/A		N/A		-> use unpack iterables function of **	a={1:'a', 3:'b'}; b={3:'b' 6:'c'}; c={**a, **b}	.difference(set2)	newset=set1.difference(set2)
Add at first position	.insert(0,item)	Lpreps[0]="about"	~tuples are "immutable" - can't add by position	none	N/A		N/A - sets are not ordered	
Add items at end	loop and append	for i in ("at","about"): ~ Lpreps.append(i)	+= can only concat tuple - not string/number	Testtup+=(1,2,3)	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Add one item at end	.append(item)	LPreps.append("behind")	+=(x,) ~note comma~	Testtup+=(4,)	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Add at position in container	.insert(position, item)	LPreps.insert(2,"of")	N/A - or programmed function	-	N/A - dictionaries are not ordered		.add(item) or add values with .union	set1.add('j); ~ set1 = set1.union(['f','g','h'])
Add somewhere inbetween	list[i]=x ~replace item i with x~	LPreps[5]="at"	N/A -or programmed function	-	dictonary[key]=value	DPreps["at"]="close to"	N/A	
Add multiple items	.extend(multiple items in a list)	LPreps.extend(["under","above"])	+=	Testtup+=(1,2,3) ↳('a', 'b', 'c', 1, 2, 3)	dict.update(dict2) ~adds dict2 to dict1~	D1.update(D2) ~also D1.update(1=dog, 2=cat)~	.union(otherset)	newset=set1.union(set2)
or simple concatenation...	list+= [item, item, ...] ~ concatenation~	LPreps+=["down","by","over"] ~add 2nd over~	+=	Testtup+=(1,2,3) ↳('a', 'b', 'c', 1, 2, 3)	see unpack iterable function of ** above	a={1:'a', 3:'b'}; b={3:'b' 6:'c'}; c={**a, **b}	N/A	
Remove a known value or key	.remove(first item x)	LPreps.remove("over")	N/A -or programmed function	-	del dictionary[key]	del DPreps["over"] ~view obj (list=keys) are dynamic!	.remove(item) or .discard(item)	myset.remove("B") ~KeyError if not present~
Remove item(s) by index	del list[index : index]	del Lpreps[2]	N/A		N/A - dictionaries are not ordered		N/A - sets are not ordered	
Remove and return the last item	.pop()	FetchedItem = LPreps.pop()	N/A -or programmed function	-	N/A - dictionaries are not ordered		N/A	
Remove and return a known item	N/A		N/A		.pop(key[,default])	DPreps.pop("of")	N/A	
Remove and return a random item	N/A		N/A		.popitem()	DPreps.popitem()	.pop()	SPreps.pop()
Remove and return item number i	.pop(i)	FetchedItem = LPreps.pop(7)	N/A -or programmed function	-	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Replace an item/pair or value	list[index]="new value"	Lpreps[2]= "among"	N/A -or programmed function		~revalue based on key~	Dpreps["with"]="possessing"	(1) create set 2 with items to be removed and use .difference, then (2) create set 3	
Replace a group of items	list[index i: index j]=new list of same length	Lpreps[2:3]=['around','by']	N/A -or programmed function		.update ~overwrites existing values~	D1.update(D2) ~also D1.update(1=dog, 2=cat)~	with the items to be added and use .union to add those back	
Retrieve sequential items	=list[from index i : to index j : step by k]	NewList=Lpreps[1:5]	tuple[i:j] ~start is 0, end is last item +1~	NewList = TupPreps[0:3]	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Retrieve values, keys, or pairs	N/A		value for each item = tuple if no [x[x]] defined	x1,x2,x3,x4,x5=TupNum [0:5] ~ 1 for1~	d.keys(), d.values(), d.items()	KeyList = list(Dpreps.keys())		
Retrieve value from known key	N/A		N/A		dictionary[key] or dictionary.get[key]	print(DPreps["of"]) ↳ "coming from"	N/A	
Retrieve all keys, values, pairs	N/A		N/A		d.keys(), d.values(), d.items()	print(DPreps.keys()) ↳ dict_keys(['over', 'with', 'of'])	N/A	
Retrieve index number of first value x	.index(x[,at or after index i [,before index j]])	MyIndex=LPreps.index("under")	.index(x[,at or after index i [,before index j]])	MyIndex = Testtup.index("c")	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Compare overlap	N/A -or programmed function						s1.isdisjoint(s2) ~True if no common elements~	Spreps.isdisjoint(myset))
Compare subset	N/A -or programmed function						s1.issubset(s2) or s1 <= s2 ~s1 contained by s2~	print(myset2 <= myset1)
*compare as true subset(not equal)	N/A -or programmed function						s1<s2 ~ both s1<=s2 and s1 !=s2 ~& not equal~	print(myset2 < myset1)
Compare superset	N/A -or programmed function						s1.issuperset(s2) or s1 >= s2	print(myset2 >= myset1)
*compare as true superset(not equal)	N/A -or programmed function						s1 > s2	print(myset2 > myset1)
Iteration (loop)	for int in list	for i in LPreps | print(i)	for int in tuple	for x in Testtup: | print (x)	TList = list(DPreps) ~ for i in range(0, len(TList)): ~ print(TList[i] + " : " + DPreps[TList[i]])			
Iteration (iter, next)	iter(list); next(itervariable, default)	x=iter(LPreps) | print(next(x,"end"))	iter(tuple) | x=iter(TupPreps) | for i in TupPreps: | print (next (x, "defalut if no value"))		iter, next | IT=iter(Dpreps); For rec in Dpreps: ~ xkey=(next(IT)); print(xkey) print(DPreps.get(xkey))		iter, next ~item delivery appears random~	
Return number of items/pairs	len(list)	len(Lpreps)	len(tuple)	len(TupPreps)	len(dictionary)	len(Dpreps)	len(setname)	print(str(len(SPreps)))
Find count of x values	.count(x) ~number of item values == x~	LPreps.count('in')	.count(x) ~number of item values == x~	TupPreps.count("with")	~can only hold unique keys - no duplicates~		can only hold unique values	
Find maximum value	.max(list)	print(max(Numlist))	max(tuple)	max(TupNum) ~TupPreps would give error	N/A		N/A	
Find minimum value	.min(list)	print(min(Numlist))	min(tuple)	min(TupNum) due to inclusion of list objects~	N/A		N/A	
Determine membership	if/in ~if then else~	if "at" in LPreps:	value in tuple name or value in (item,item,item)	BooleanVal = "xx" in Testtup ↳ False	key in , key not in ~returns boolean value~	"of" in DPreps ↳ True | "among" in DPreps ↳ False	in , not in	print(str("B" in myset)) ↳ True
-					.has_key(key)	Dpreps.has_key("with")		
Copy	.copy() ~return shallow copy~	Lcopy=LPreps.copy()	Why copy an immutable object?	-	dictionary.copy()	NewPreps = DPreps.copy()	.copy()	newset=oldset.copy()
Sort	.sort(key=None, reverse=False)~rev is low->hi~	LPreps.sort() ~key ex: key= str.lower~	sorted(tuple, reverse=False))	sorted(tup) ↳['C', 'a', 'c', 'f', 'x', 'z', 'zz']	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Reverse items	.reverse	LPreps.reverse()	tuple[::-1] ~ok, it's a slice trick but it works~	tup[::-1] ↳('zz', 'z', 'x', 'f', 'c', 'a', 'C')	N/A - dictionaries are not ordered		N/A - sets are not ordered	
Clear all	.clear or Lpreps=[]	Lcopy.clear()	=() ~clears the tuple ~	tup=()	dictionary.clear()	DPreps.clear()	.clear()	someset.clear()
Delete the object	del list	del LPreps	del tuple	del tup ~after gives name not defined error~	del dictionary	del Dpreps	del set ~after which attempted access->error~	del someset
Convert	list(tuple)	Newlist = list(SomeTuple)	tuple(list) ~convert list to tuple~	mytup = tuple(mylist)			list to set:	myset=set(mylist)
Other: setdefault					dictionary.setdefault(key[,default]) ~if key in dict return value, if not insert with value of default~			