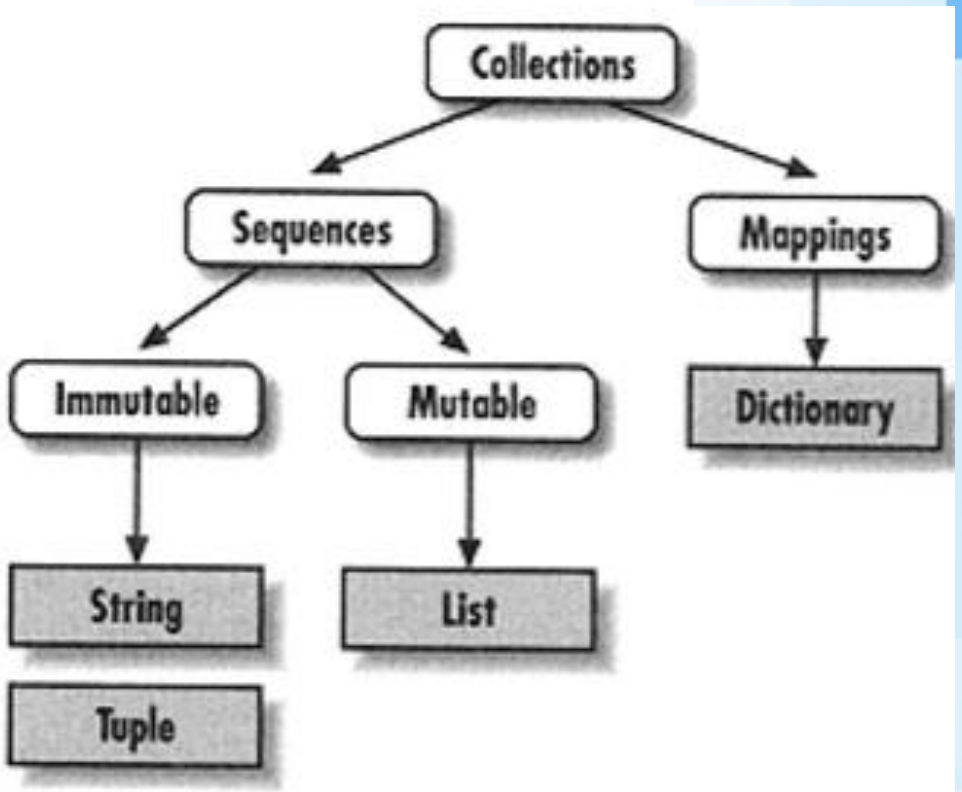
Python-List, Tuple, Dictionary

Dr. Sarwan Singh

NIELIT Chandigarh

Agenda

- Introduction – Lists , tuples, dictionaries
- Basic operations
  - indexing, slicing, matrixes
  - Concatenation, Repetition
  - Membership, Iteration



"Assignment Creates References, Not Copies"

List

- Python has six built-in **sequence types** : strings, Unicode strings, lists, tuples, buffers, and xrange objects. (source : https://docs.python.org/2.4/lib/typesseq.html) .
- List is one of the popular sequence in Python.
- List is collection of objects (ordered sequence of data similar to String except that String can only hold characters)
- List need not be homogeneous , (its heterogeneous) and it is mutable
- List is arbitrarily nestable
- Arrays of object references- lists contain zero or more references to other objects (like array of pointers in C Language)

List

- Each element of List is positioned/indexed starting from 0
- Operation on Strings like indexing, slicing, adding, multiplying, and checking for membership are all available in Lists
- E.g. studentRec = [‘Amrit’, ‘kumar’ , 21, 2000]

  recFields = [‘firstname’ , ‘lastname’, ‘rollno’, ‘fee’]

```
studentRec = ['Amrit', 'kumar' , 21, 2000]

StudentList = [2, studentRec , ['Amit','jain',10,4000]]

StudentList

[2, ['Amrit', 'kumar', 21, 2000], ['Amit', 'jain', 10, 4000]]

StudentList[1]

['Amrit', 'kumar', 21, 2000]
```

Basic operations

- Basic operation on List are similar to Strings

Expression	Description
len	Length
list1 + list2	Concatenation
List * 2	Repetition
‘elem’ in List	Membership
for x in List:	Iteration

```
StudentList
```

```
[2, ['Amrit', 'kumar', 21, 2000], ['Amit', 'jain', 10, 4000]]
```

```
StudentList[1]
```

```
['Amrit', 'kumar', 21, 2000]
```

```
StudentList[1:]
```

```
[['Amrit', 'kumar', 21, 2000], ['Amit', 'jain', 10, 4000]]
```

```
'Amrit' in StudentList
```

```
False
```

```
2 in StudentList
```

```
True
```

```
'Amrit' in StudentList[1]
```

```
True
```

Built-in functions and Methods

- Min (list)
- Max (list)
- Len (list)

```
list1 = [10,3,5,14,21,9,13]
print(list1)
```

```
[10, 3, 5, 14, 21, 9, 13]
```

```
list1[5:7]
```

```
[9, 13]
```

```
del list1[5]
list1[5:7]
```

```
[13]
```

```
list1 = [10,3,5,14,21,9,13]
print(list1)
```

```
[10, 3, 5, 14, 21, 9, 13]
```

```
list1[5:7] #slicing
```

```
[9, 13]
```

```
list1[2:4] = [] #shrinking list
```

```
print(list1)
```

```
[10, 3, 21, 13]
```

```
StudentList.
```

StudentList.append
StudentList.clear
StudentList.copy
StudentList.count
StudentList.extend
StudentList.index
StudentList.insert
StudentList.pop
StudentList.remove
StudentList.reverse

```
list("34Amrit") #converting String to List
```

```
['3', '4', 'A', 'm', 'r', 'i', 't']
```

Zip

- The purpose of zip() is to **map the similar index of multiple containers** so that they can be used just using as single entity.
- passing two iterables, like lists, zip() enumerates them together

- <u>Practical use:</u> student database or scorecard or any other utility that requires mapping of groups.

```
StudentList[1]
```

```
['Amrit', 'kumar', 21, 2000]
```

```
recFields = ['firstname' , 'lastname', 'Rollno', 'fee']
```

```
StudentRecPrint = zip(recFields, StudentList[1]) #zip to map values
stuList = list(StudentRecPrint) #converting to list
print(stuList) #print list
```

```
[('firstname', 'Amrit'), ('lastname', 'kumar'), ('Rollno', 21), ('fee', 2000)]
```

```
header, sturecord= zip(*stuList) #unzipping values
print (header, '\n', sturecord)
```

```
('firstname', 'lastname', 'Rollno', 'fee')
 ('Amrit', 'kumar', 21, 2000)
```

LIST Equivalence/reference

- == equality operator determines if two lists contain the same elements
- is operator determines if two variables alias the same list
- The association of a variable with an object is called a reference
- Aliase : An object with more than one reference has more than one name

```
a=[10,20,30,40]
```

```
b=a
c=[10,20,30,40]
```

```
print (" List a: " ,a , " id(a): ", id(a))
print (" List b: " ,b , " id(b): ", id(b))
print (" List c: " ,c , " id(c): ", id(c))
```

```
List a:  [10, 20, 30, 40]  id(a):  1326451643144
List b:  [10, 20, 30, 40]  id(b):  1326451643144
List c:  [10, 20, 30, 40]  id(c):  1326450352200
```

```
b[2] = 35
c[2] = 35
print (" List a: " ,a , " id(a): ", id(a))
print (" List b: " ,b , " id(b): ", id(b))
print (" List c: " ,c , " id(c): ", id(c))
```

```
List a:  [10, 20, 35, 40]  id(a):  1326451643144
List b:  [10, 20, 35, 40]  id(b):  1326451643144
List c:  [10, 20, 35, 40]  id(c):  1326450352200
```

```
a==b
```

```
True
```

```
a is b
```

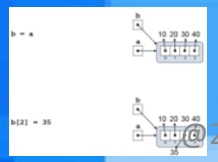
```
True
```

```
b==c
```

```
True
```

```
b is c
```

```
False
```



Repetition adds one-level deep

- sequence repetition is like adding a sequence to itself a number of times
- When mutable sequences are nested, effect is different

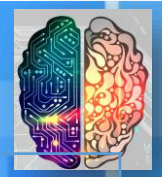
```
list1 = [1,2,3,4]
A= list1*4
```

```
B=[list1] *4
print('list1 *4 = ',A) ; print('[list1] *4 = ',B)
```

```
list1 *4 =  [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
[list1] *4 =  [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
```

```
list1[1] = 0
print('list1 *4 = ',A) ; print('[list1] *4 = ',B)
```

```
list1 *4 =  [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
[list1] *4 =  [[1, 0, 3, 4], [1, 0, 3, 4], [1, 0, 3, 4], [1, 0, 3, 4]]
```

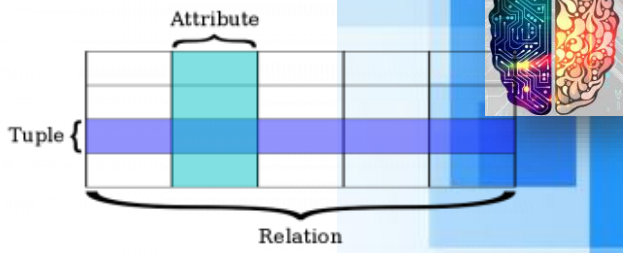


FUNCTION	DESCRIPTION
Append()	Add an element to the end of the list
Extend()	Add all elements of a list to the another list
Insert()	Insert an item at the defined index
Remove()	Removes an item from the list
Pop()	Removes and returns an element at the given index
Clear()	Removes all items from the list
Index()	Returns the index of the first matched item
Count()	Returns the count of number of items passed as an argument
Sort()	Sort items in a list in ascending order
Reverse()	Reverse the order of items in the list
copy()	Returns a copy of the list

Source : Geeksforgeeks.org

FUNCTION	DESCRIPTION
round()	Rounds off to the given number of digits and returns the floating point number
reduce()	apply a particular function passed in its argument to all of the list elements stores the intermediate result and only returns the final summation value
sum()	Sums up the numbers in the list
ord()	Returns an integer representing the Unicode code point of the given Unicode character
cmp()	This function returns 1, if first list is “greater” than second list
max()	return maximum element of given list
min()	return minimum element of given list
all()	Returns true if all element are true or if list is empty
any()	return true if any element of the list is true. if list is empty, return false
len()	Returns length of the list or size of the list
enumerate()	Returns enumerate object of list
accumulate()	apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
filter()	tests if each element of a list true or not
map()	returns a list of the results after applying the given function to each item of a given iterable
lambda()	This function can have any number of arguments but only one expression, which is evaluated and returned

Source : Geeksforgeeks.org



Python-Tuples

- Another type of sequence like list
- Immutable
- Uses ( )
- comma-separated list of values

- Tuples are immutable, cannot update or change the values
- Tuples can be concatenated (+) , deleted using del
- Other basic operation like list are same : indexing, slicing, matrixes

```
tpl = () #empty tuple
tpl
```

```
()
```

```
tpl = (10)
```

```
tpl[0]
```

```
----------------------------------------
TypeError                                 Tr
<ipython-input-91-20e03974e213> in <module>(
----> 1 tpl[0]

TypeError: 'int' object is not subscriptable
```

```
tpl = (10,)
tpl[0]
```

```
10
```

```
tpl[0]=20
```

```
----------------------------------
TypeError
<ipython-input-93-2d7cb66a897d> in
----> 1 tpl[0]=20

TypeError: 'tuple' object does not support
```

```
tpl1=(1,2)
```

```
tpl2 = tpl + tpl1
tpl2
```

```
(10, 1, 2)
```

sequence packing-unpacking

- packing always creates tuple
- unpacking works for any sequence
- Parentheses is optional while packing

```
tpl = (10,'amrit', 2000.50)
```

```
rno, name, fee = tpl #unpacking
```

```
print("tuple-tpl : ", tpl)
print('Rno  :',rno)
print('Name :',name)
print('fee  :',fee)
```

```
tuple-tpl :  (10, 'amrit', 2000.5)
Rno  : 10
Name : amrit
fee  : 2000.5
```

```
tpl2 = rno, name, fee # packing
```

```
tpl2
```

```
(10, 'amrit', 2000.5)
```

Changing element of a tuple

- Immutable Types Can't Be Changed in Place

```
T = (1, 2, 3)
T[2] = 4                # error!
```

```
----------------------------------------------------------
TypeError                                 Traceback (
<ipython-input-11-7bca93914e13> in <module>()
      1 T = (1, 2, 3)
----> 2 T[2] = 4                # error!
      3 T = T[:2] + (4,)        # okay: (1, 2, 4)
      4 print(T)

TypeError: 'tuple' object does not support item assig
```

```
T = T[:2] + (4,)        # okay: (1, 2, 4)
print(T)
```

```
(1, 2, 4)
```

Tuple assignment

- swap a and b

```
Temp = a
a =  b
b = temp
```

- tuple assignment is more elegant

```
a, b = b, a
```

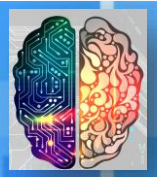
```
a, b = 1, 2, 3  # error
```

ValueError: too many values to unpack

```
email = 'monty@python.org'  un
user, domain = email.split('@')
```

Comparing tuple

```
(0, 1, 2) < (0, 3, 4)
True
```



Python-Dictionary

- Key : value pair separated with :
- Uses curly brackets { }
- Keys are unique in a dictionary, values may not
- values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples

Updation

- dict2['school']='DPS Delhi'

Deletion

- del dict1 ['name']; # remove entry with key 'Name'
- dict1.clear(); # remove all entries in dict1
- del dict1 ; # delete entire dictionary

```
dict1= {} #empty dictionary
```

```
print(len(dict1)); print(dict1);
```

```
0
{}
```

```
dict2 = {'rno':10,'name':'amrit', 'fee':2000.50 }
```

```
dict2
```

```
{'fee': 2000.5, 'name': 'amrit', 'rno': 10}
```

```
dict2['name']
```

```
'amrit'
```

Uses of Dictionary

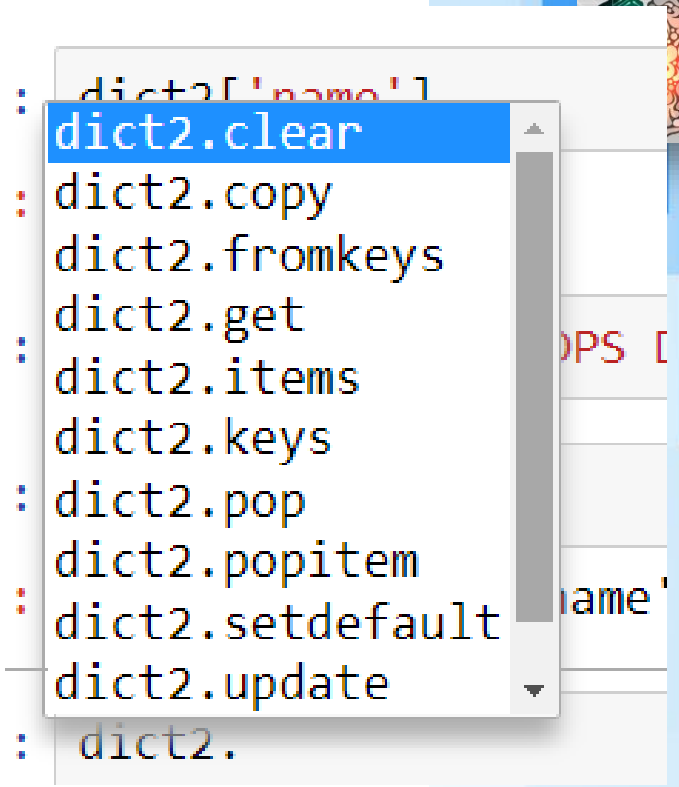
- Web crawlers use dictionary for storing data
- Store database settings
- Representing application templates
- In unit tests, sample data for web forms (mapping field name to value)
- Mapping objects on game field – literally maps
- Building indexes of contents – phone book
- Exchanging data over Internet – JSON

- **Lists:** used for collection of similar/non similar items.
- **Tuples** are generally used for smaller groups of similar items

As today, more and more web apps are moving to the RESTFul design pattern, and JSON is key to the client/server data exchange.

Methods

- Exercise :
- Write a python function to get all the string elements inside tuple passed as an argument (nested tuple)
  - Without recursion
  - With recursion
- Redefined method to accept list as an argument



METHODS	DESCRIPTION
copy()	They copy() method returns a shallow copy of the dictionary.
clear()	The clear() method removes all items from the dictionary.
pop()	Removes and returns an element from a dictionary having the given key.
popitem()	Removes the arbitrary key-value pair from the dictionary and returns it as tuple.
get()	It is a conventional method to access a value for a key.
dictionary_name.values()	returns a list of all the values available in a given dictionary.
str()	Produces a printable string representation of a dictionary.
update()	Adds dictionary dict2's key-values pairs to dict
setdefault()	Set dict[key]=default if key is not already in dict
keys()	Returns list of dictionary dict's keys
items()	Returns a list of dict's (key, value) tuple pairs
has_key()	Returns true if key in dictionary dict, false otherwise
fromkeys()	Create a new dictionary with keys from seq and values set to value.
type()	Returns the type of the passed variable.
cmp()	Compares elements of both dict.

Jupyter Notebook Link

- https://colab.research.google.com/drive/1jUAhD0VvIz23RA8Hr25pDzK__srpTcsm?usp=sharing

Github Repository Link

- https://github.com/sarwansingh/Python/tree/master/ORD