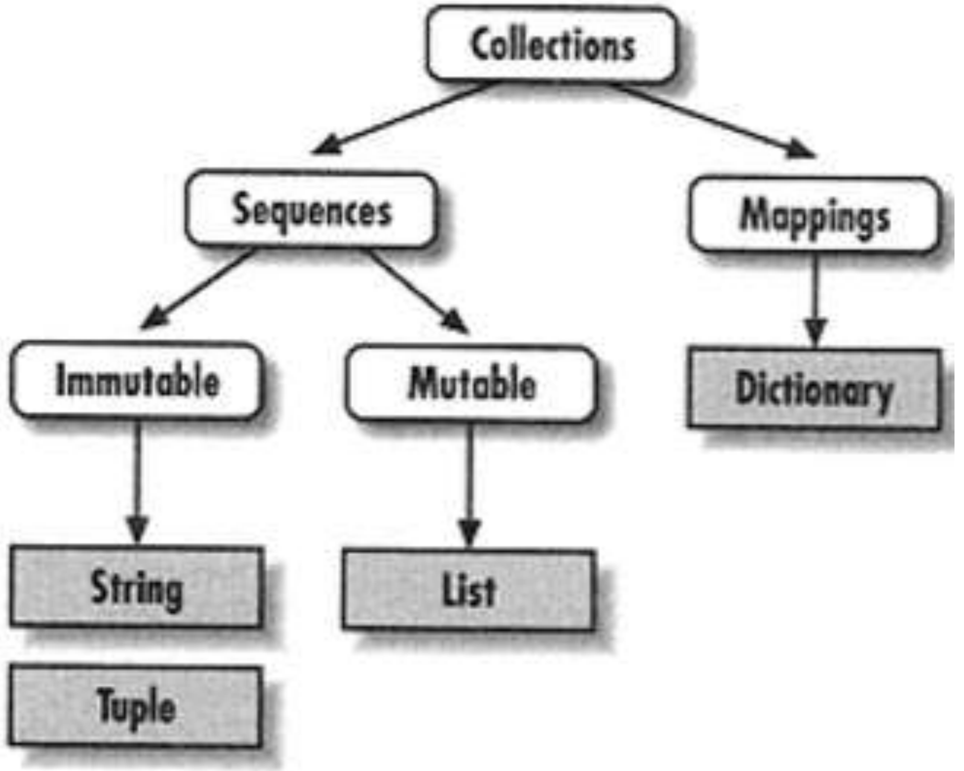
Python-List, Tuple, Dictionary

Dr. Sarwan Singh

Agenda

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



“Assignment Creates References, Not Copies”

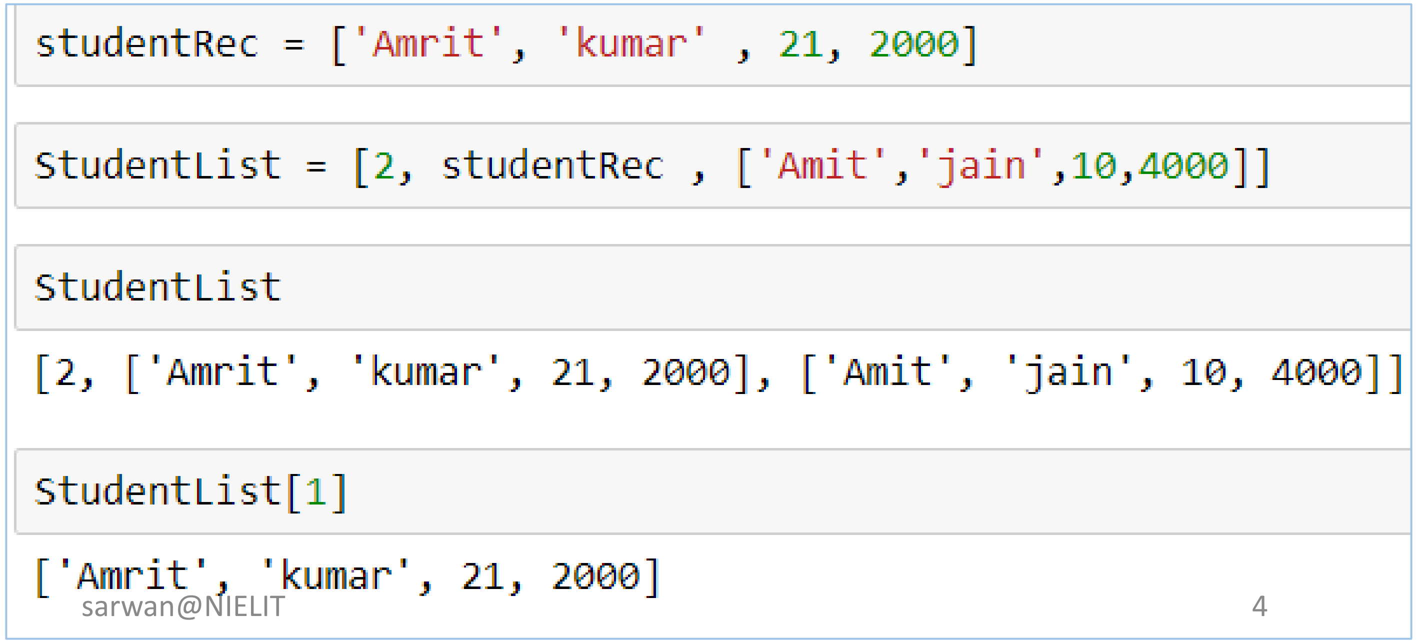
List

- 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]]
```

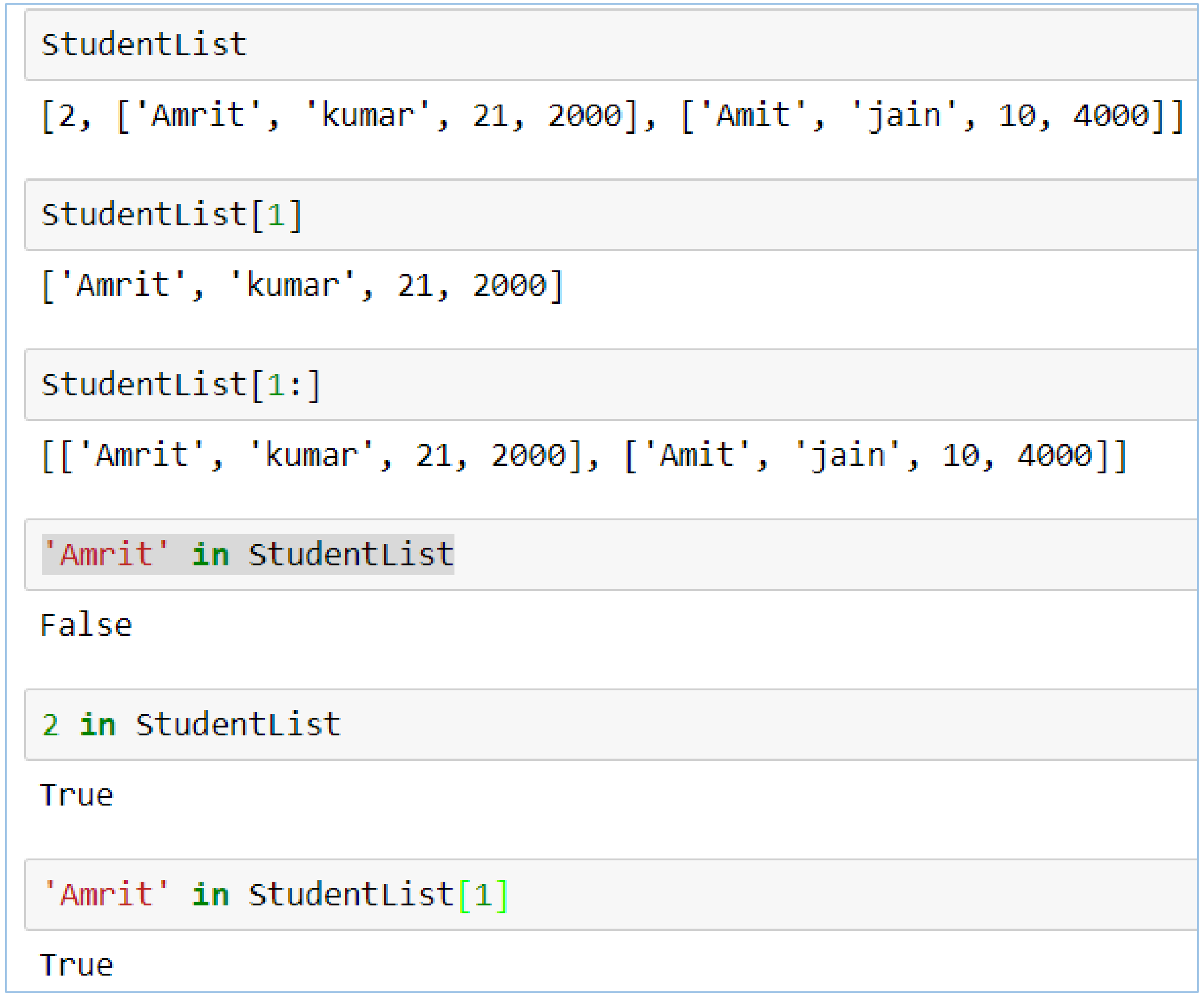
```
StudentList[1]
```

```
['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
```

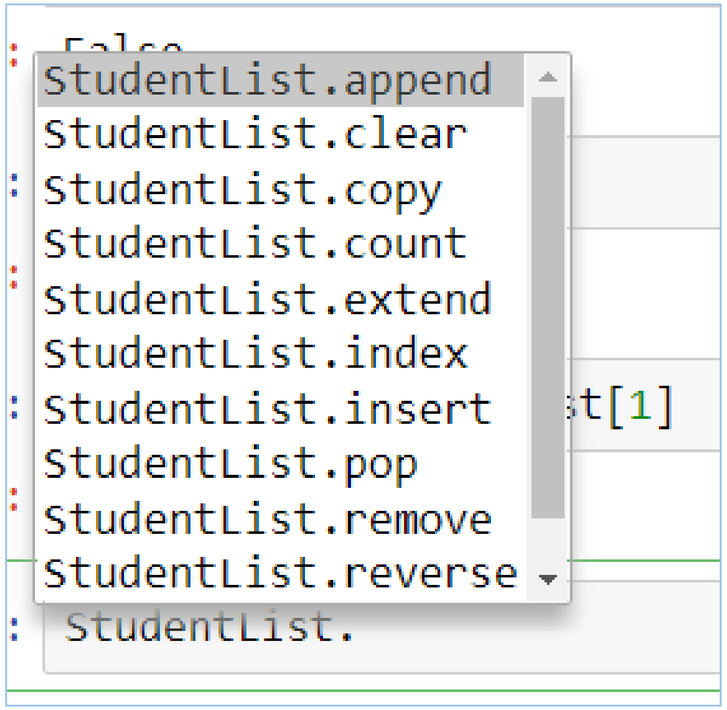
```
True
```

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

```
True
```

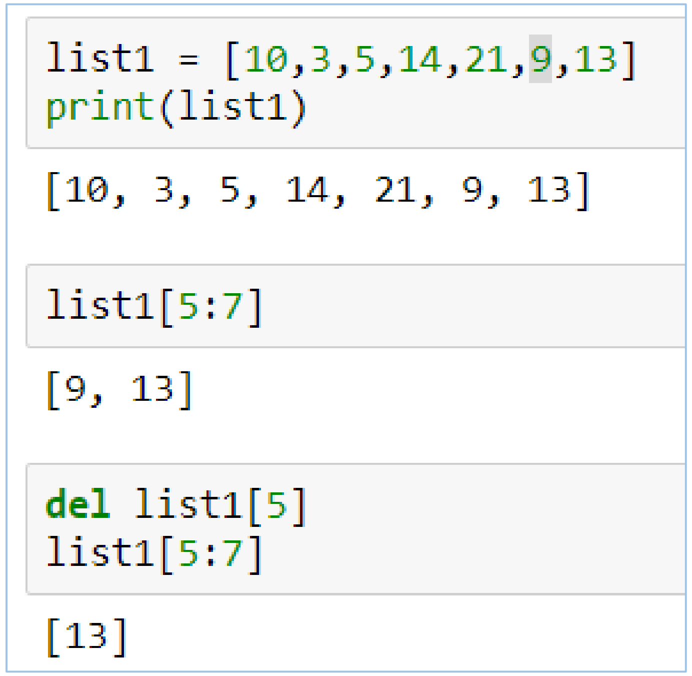
Built-in functions and Methods

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



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

```
StudentList.
```



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

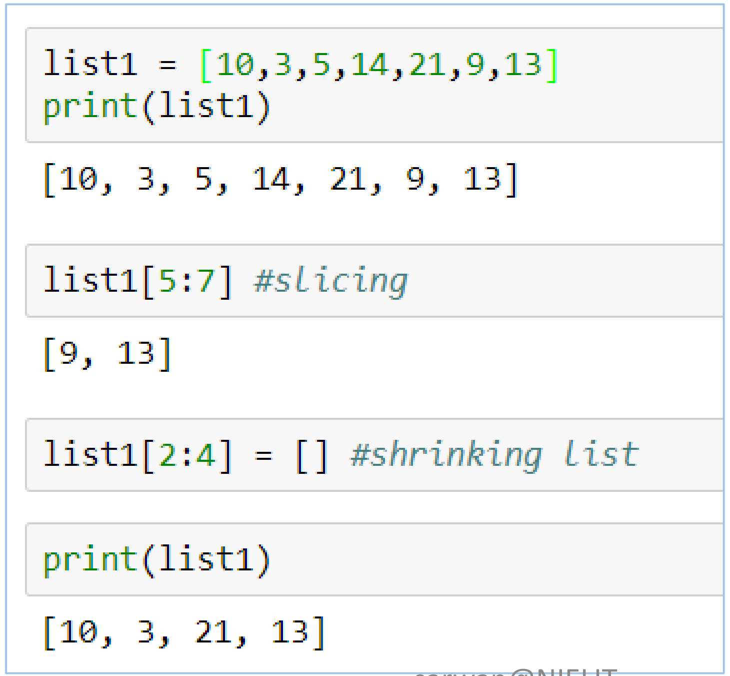
```
[10, 3, 5, 14, 21, 9, 13]
```

```
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]
```

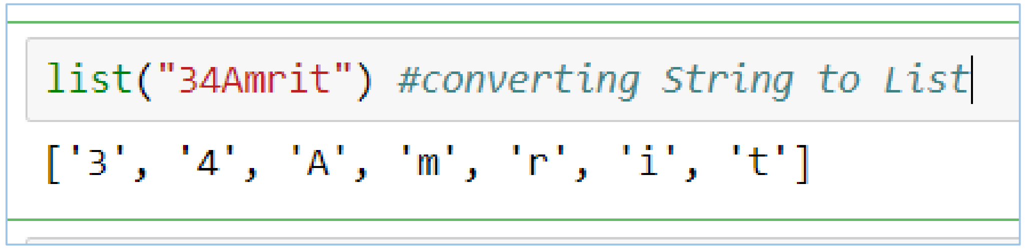
```
list1[5:7] #slicing
```

```
[9, 13]
```

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

```
print(list1)
```

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



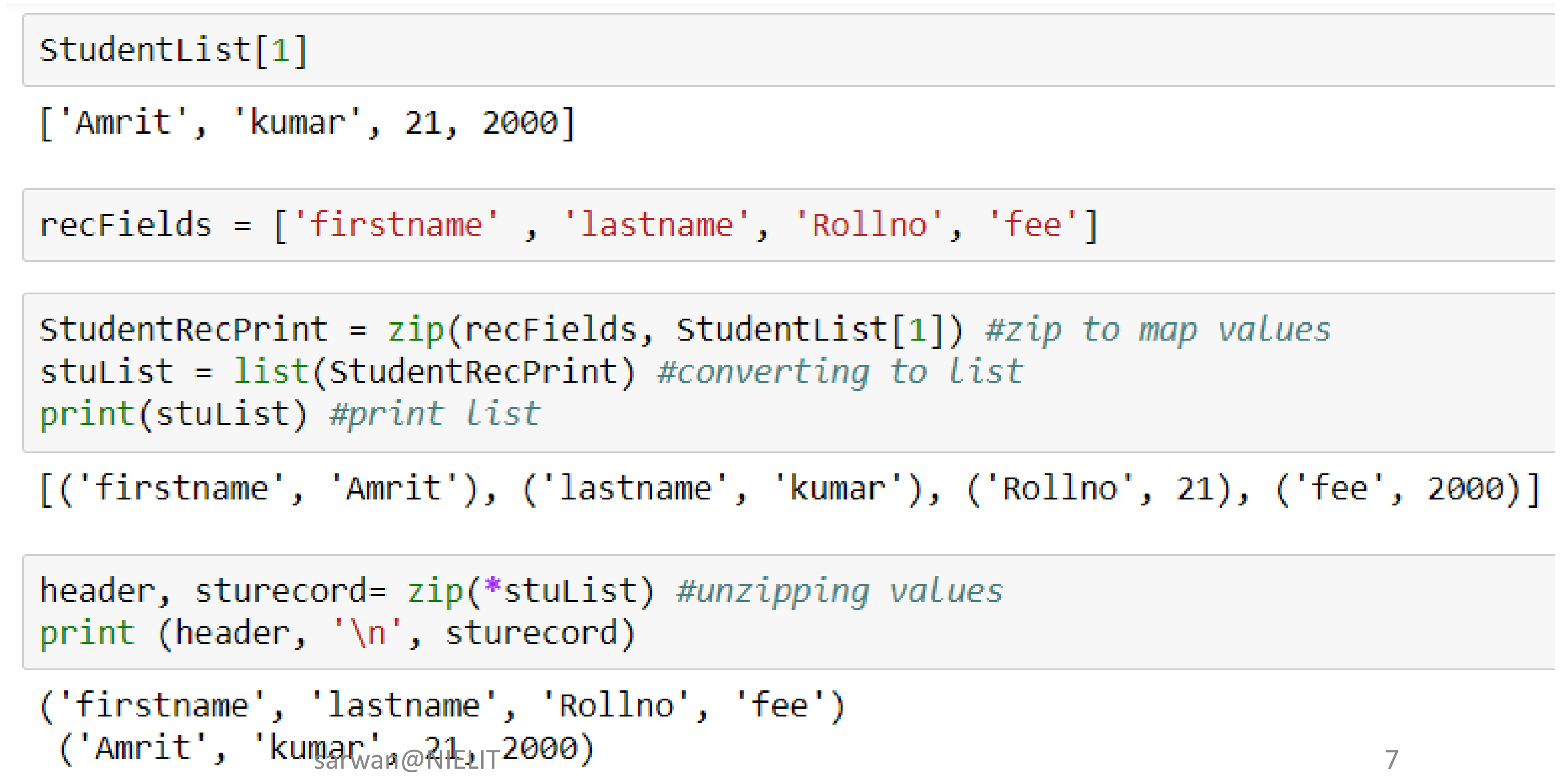
```
list("34Amrit") #converting String to List
```

```
['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
```

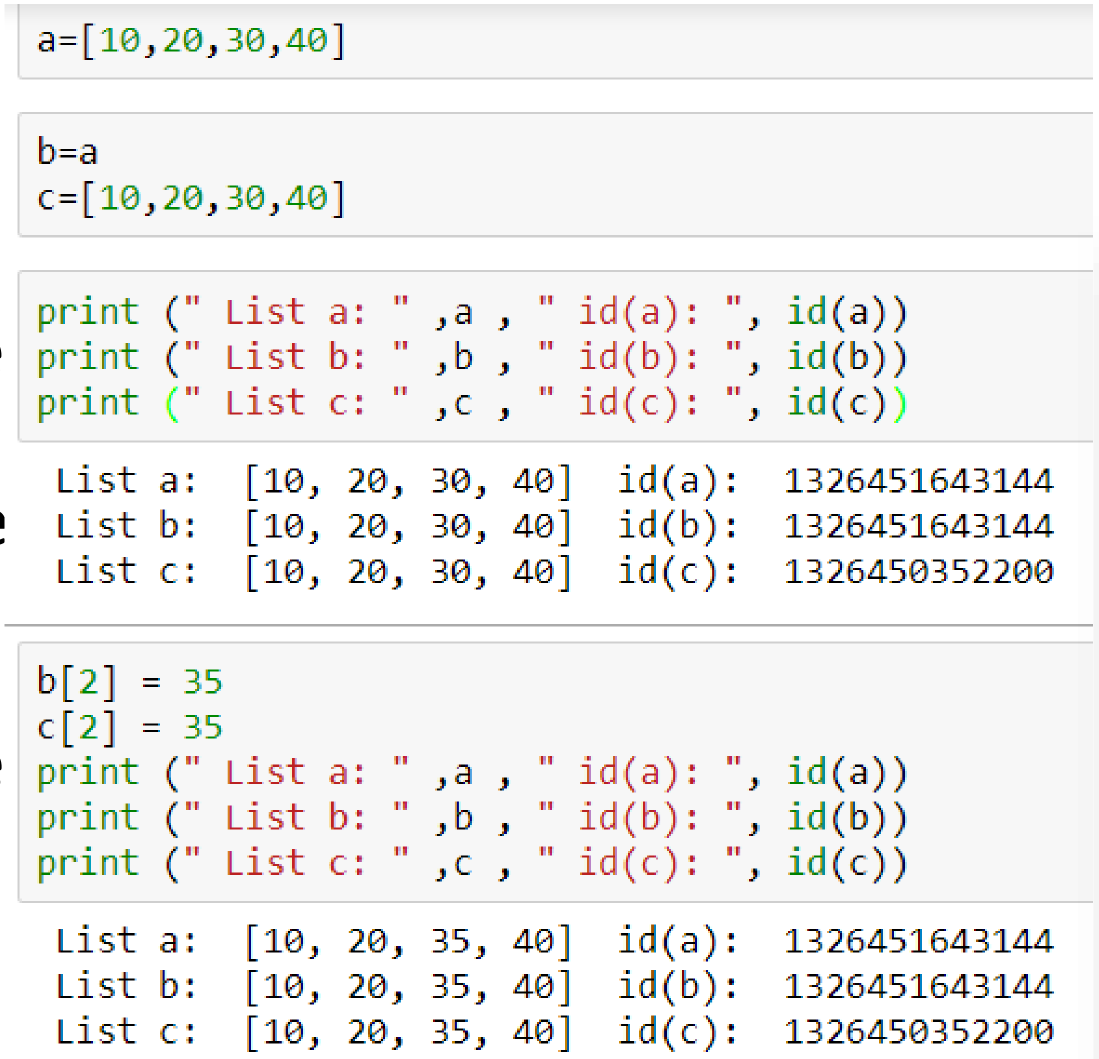
```
[('firstname', 'Amrit'), ('lastname', 'kumar'), ('Rollno', 21), ('fee', 2000)]
```

```
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]
```

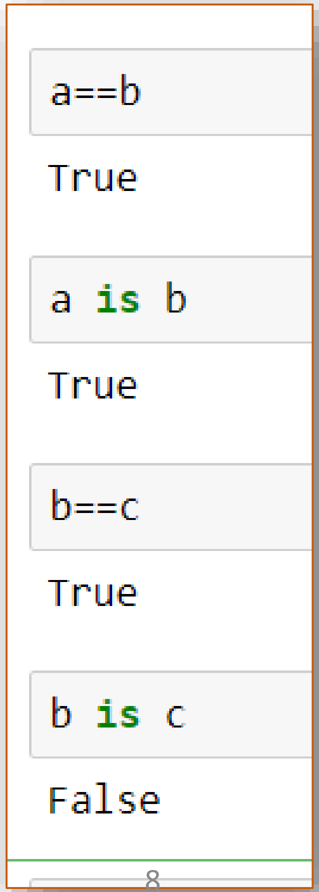
```
b=a
c=[10,20,30,40]
```

```
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
```

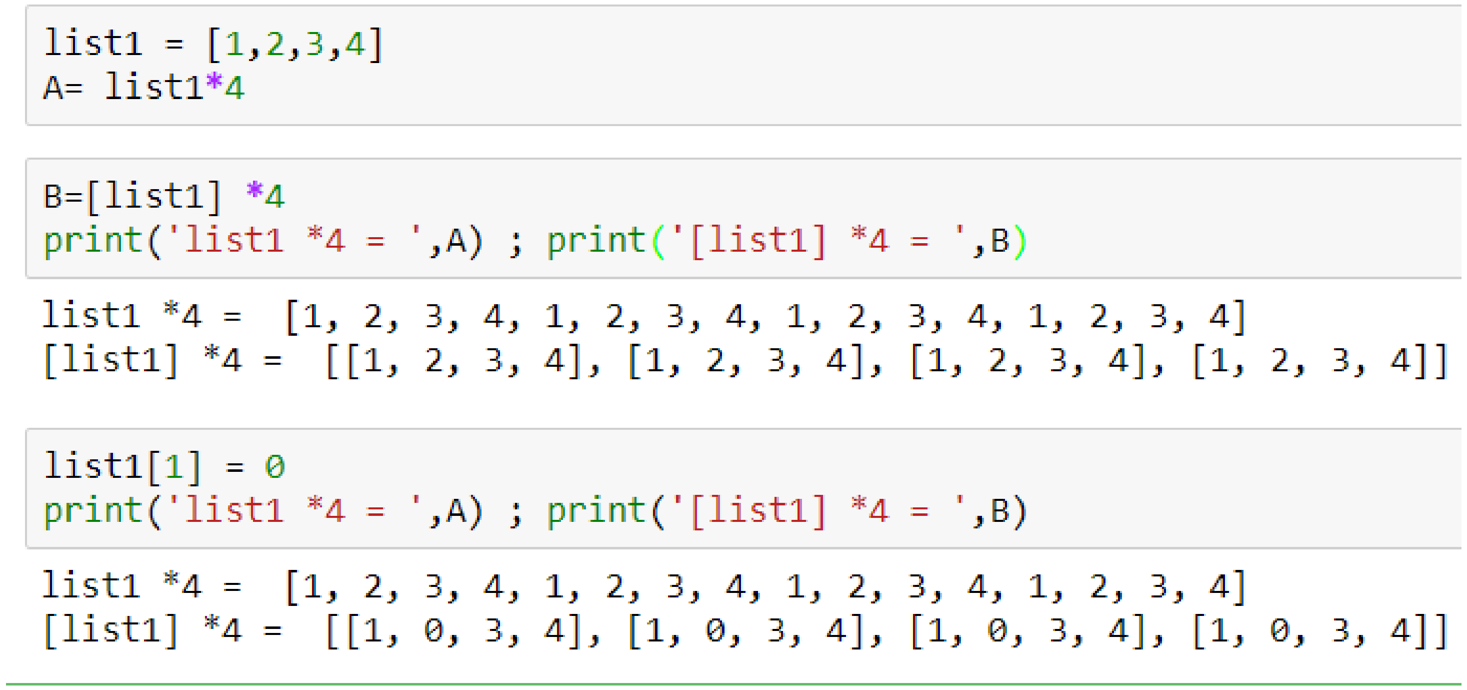
```
True
```

```
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



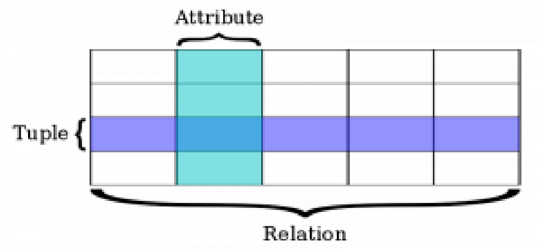
```
list1 = [1,2,3,4]
A= list1*4
```

```
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)
```

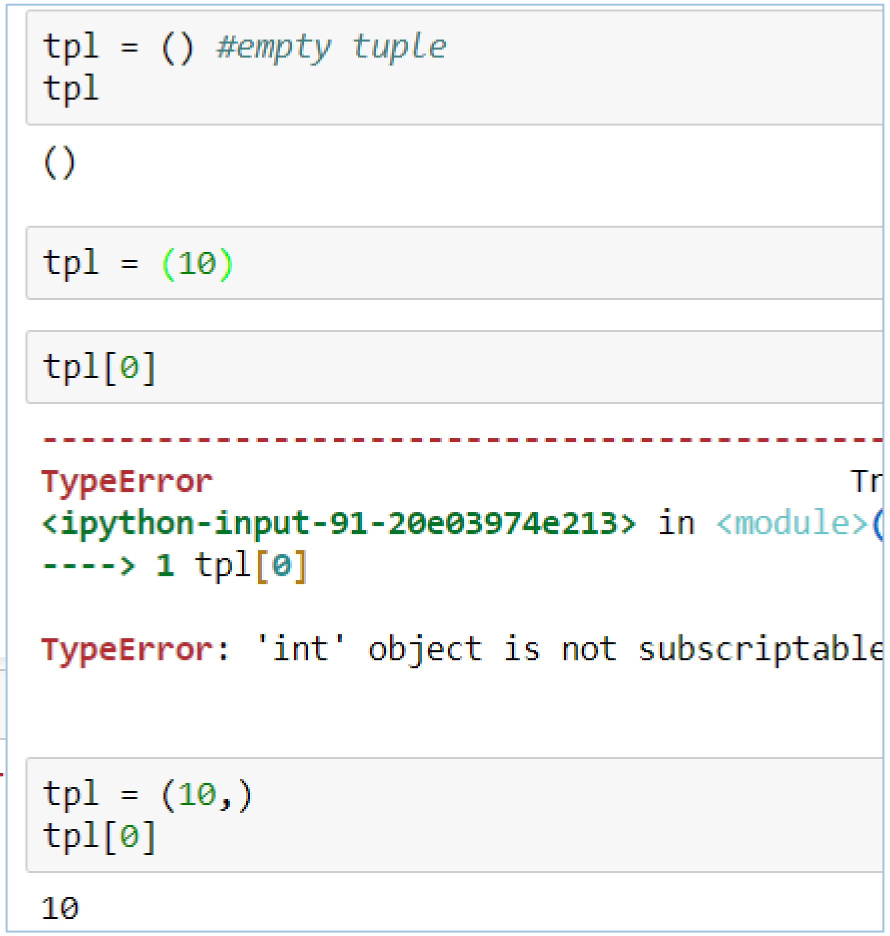
```
list1 *4 =  [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
[list1] *4 =  [[1, 0, 3, 4], [1, 0, 3, 4], [1, 0, 3, 4], [1, 0, 3, 4]]
```



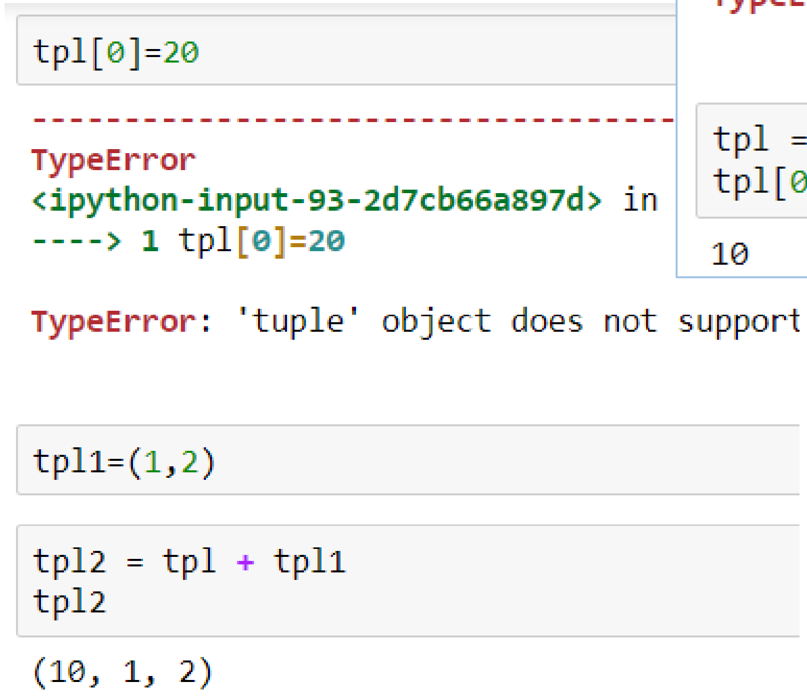
Python-Tuples

- 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



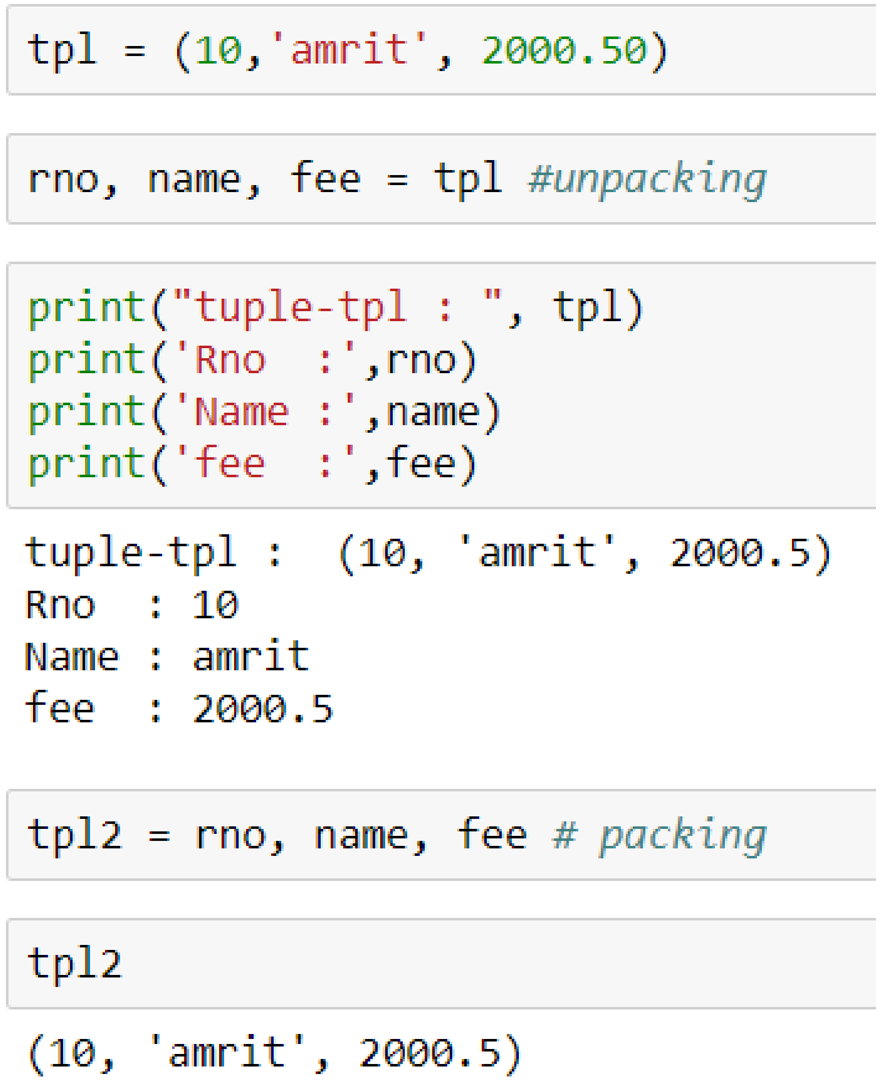
```
tpl = () #empty tuple
tpl

()

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
```

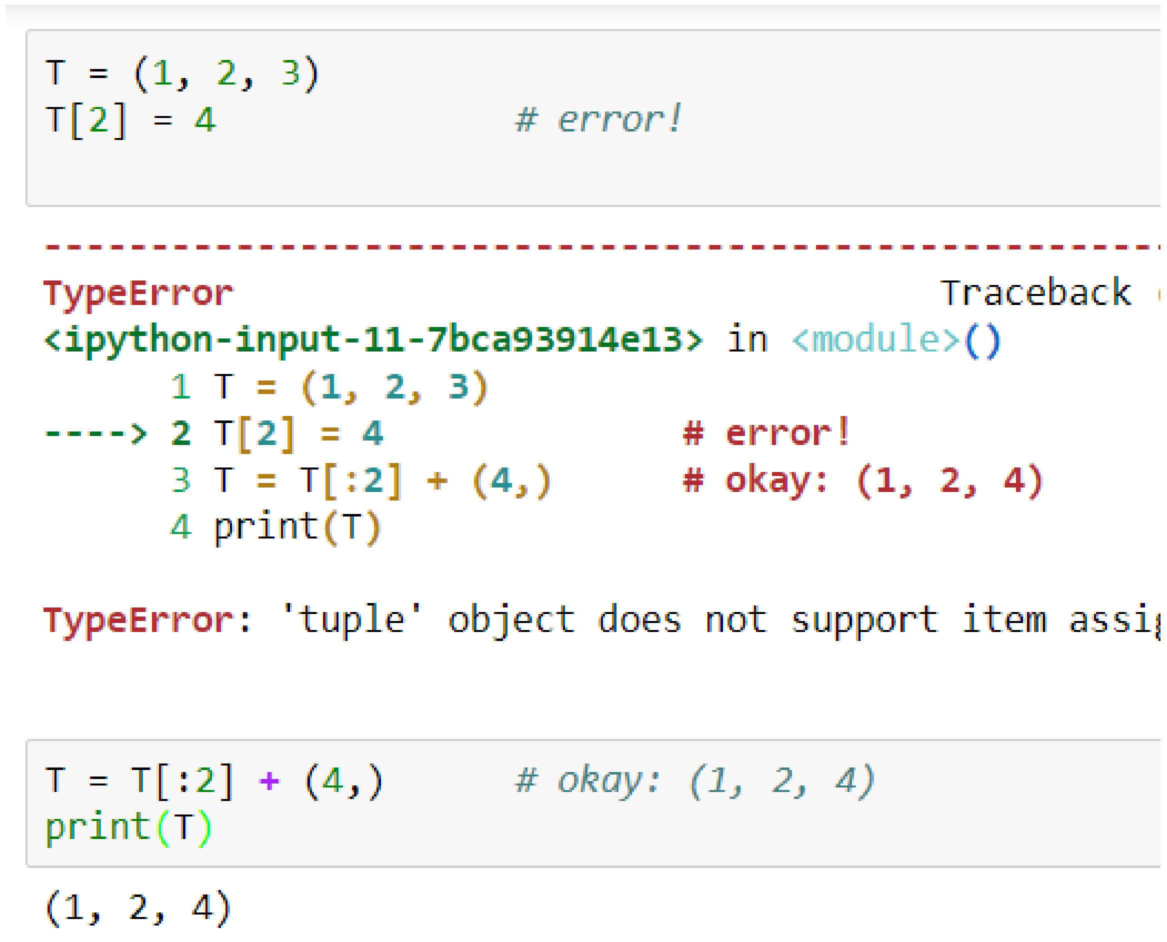
```
tpl2 = rno, name, fee # packing
```

```
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 assi
```

```
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

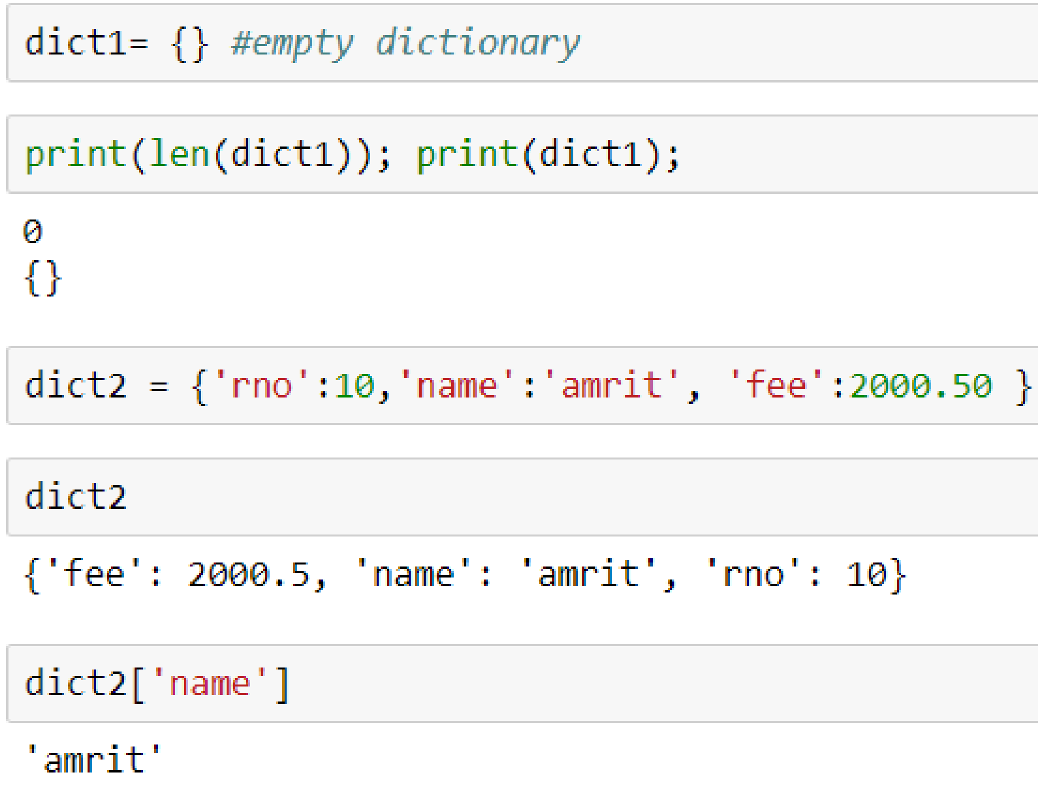
- Key : value pair separated with :
- Uses curly brackets { }
- Keys are unique in a dictionary, values may not
- values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples

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
```

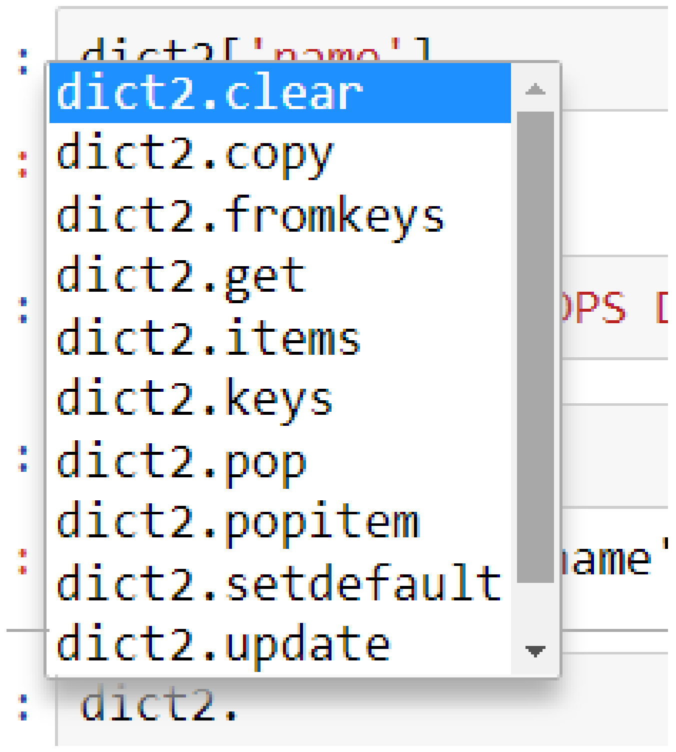
```
{'fee': 2000.5, 'name': 'amrit', 'rno': 10}
```

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

```
'amrit'
```

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



```
dict2.
```