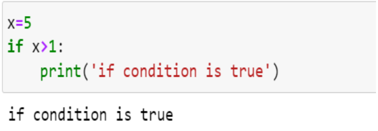
Data Science Workshop-1

ITER, SOA University

Centre for Data Science
SOA University

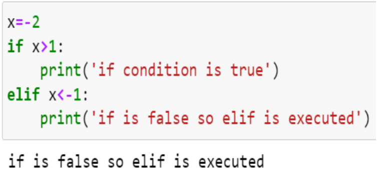
if-elif-else

- We use the if statement to run a block code only when a certain condition is met.
- There are three forms of the if...else statement
  - if statement
  - if...else statement
  - if...elif...else statement
- nested if when we write an if statements inside an if or elif or else statement.



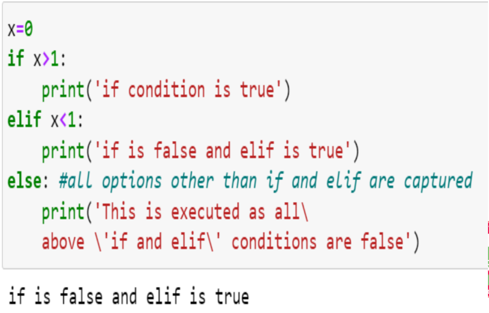
```python
x=5
if x>1:
    print('if condition is true')
```

```
if condition is true
```



```python
x=-2
if x>1:
    print('if condition is true')
elif x<-1:
    print('if is false so elif is executed')
```

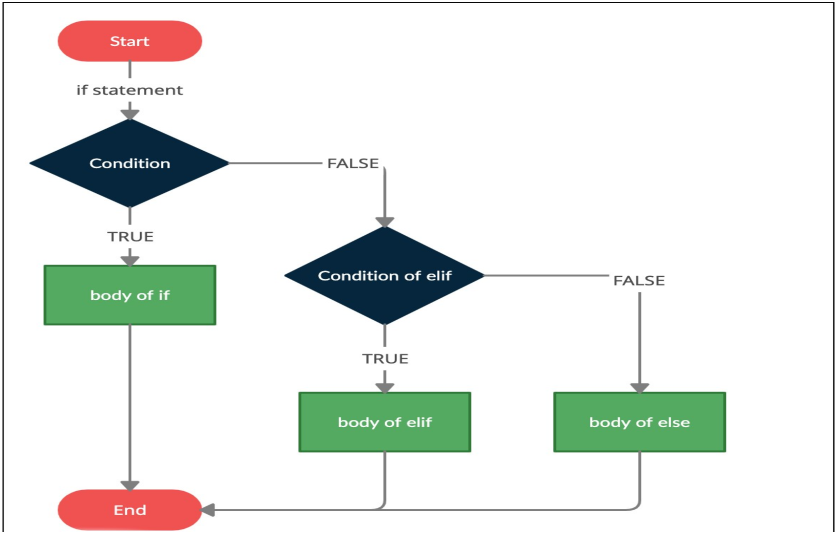
```
if is false so elif is executed
```



```python
x=0
if x>1:
    print('if condition is true')
elif x<1:
    print('if is false and elif is true')
else: #all options other than if and elif are captured
    print('This is executed as all\
    above \'if and elif\' conditions are false')
```

```
if is false and elif is true
```

working rule for if-elif-else



Loops

- Run a single statement or set of statements repeatedly using a loop command.
- while loop repeats a statement or group of statements while a given condition is TRUE.
- for loop is used to iterate over sequences such as lists, tuples, string, and range function, or any other iterable objects.
- we can use (optional) else statement after for loop and while loop, when the loop terminates smoothly(without break), the else statements gets executed.
- If the condition of a loop is always True, the loop runs for infinite times (until the memory is full)
- range(n) will produce values between 0 and n-1. (n numbers)
- range function takes 3 arguments. range(start, stop, step size). If the step size is not specified, it defaults to 1.

for loop

syntax

```
for i in iterables:
        statements
```

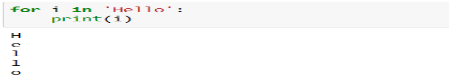
- The indented statements inside the for loops are executed once for each item in an iterable.

for loop

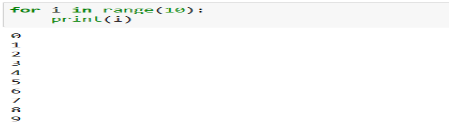
syntax

for i in iterables:
        statements

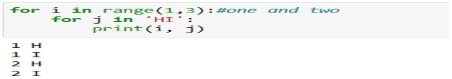
- The indented statements inside the for loops are executed once for each item in an iterable.
- The variable i takes the value of the next item of the iterable each time through the loop.



```
for i in 'Hello':
    print(i)

H
e
l
l
o
```



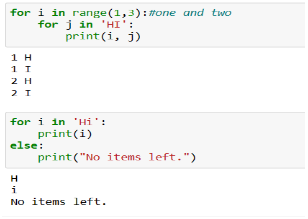
```
for i in range(10):
    print(i)

0
1
2
3
4
5
6
7
8
9
```



```
for i in range(1,3):#one and two
    for j in 'HI':
        print(i, j)

1 H
1 I
2 H
2 I
```

for loop

- nested for loops to iterate over two ranges of numbers. The inner loop is executed for each value of i in the outer loop.



```
for i in range(1,3):#one and two
    for j in 'HI':
        print(i, j)
```

```
1 H
1 I
2 H
2 I
```

```
for i in 'Hi':
    print(i)
else:
    print("No items left.")
```

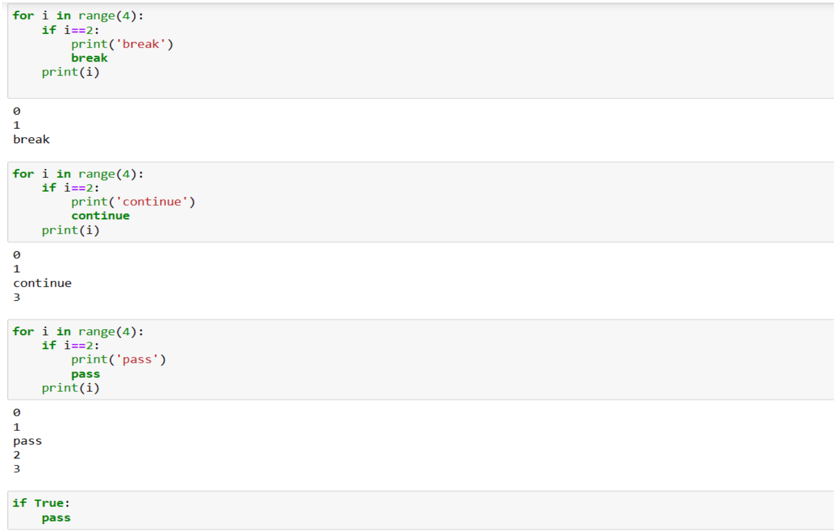
```
H
i
No items left.
```

- The else part is executed when the loop is exhausted (after the loop iterates through every item of a sequence).(This is optional)

Break vs Continue

- With the break statement we can stop the loop before it has looped through all the items. Terminates the loop's execution and transfers the program's control to the statement next to the loop.
- Skips the current iteration of the loop. The statements following the continue statement are not executed once the Python interpreter reaches the continue statement.

Continue statement



```
for i in range(4):
    if i==2:
        print('break')
        break
    print(i)
```

```
0
1
break
```

```
for i in range(4):
    if i==2:
        print('continue')
        continue
    print(i)
```

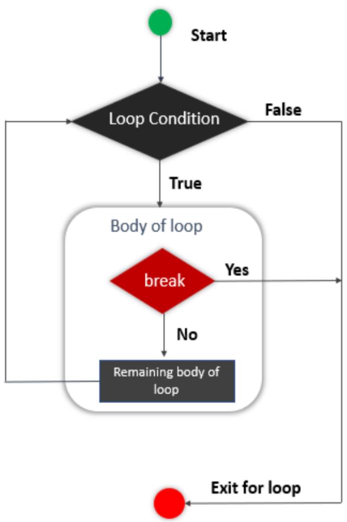
```
0
1
continue
3
```

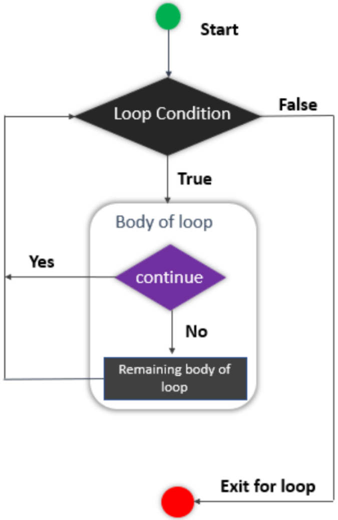
```
for i in range(4):
    if i==2:
        print('pass')
        pass
    print(i)
```

```
0
1
pass
2
3
```

```
if True:
    pass
```

break vs continue





Pass statement

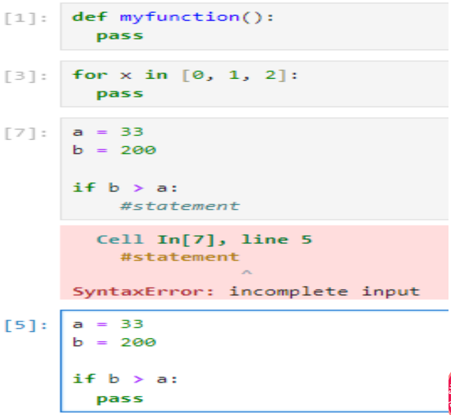
- Pass statement It is used when a statement is syntactically necessary, but no code is to be executed.

Pass statement

- Pass statement It is used when a statement is syntactically necessary, but no code is to be executed.(If it does nothing why do we need it?)

Pass statement

- Pass statement It is used when a statement is syntactically necessary, but no code is to be executed.(If it does nothing why do we need it?)
- The statement can be used to fulfill a place in a block of code that needs at least 1 statement.



```
[1]: def myfunction():
       pass

[3]: for x in [0, 1, 2]:
       pass

[7]: a = 33
     b = 200

     if b > a:
         #statement

       Cell In[7], line 5
         #statement
                   ^
     SyntaxError: incomplete input

[5]: a = 33
     b = 200

     if b > a:
       pass
```

Summary of Loop control statements

Loop Control statements: Statements used to control loops and change the course of iteration

- Break statement
- Continue statement

Summary of Loop control statements

Loop Control statements: Statements used to control loops and change the course of iteration

- Break statement
- Continue statement
- The else block will not execute if the for loop is stopped by a break statement.

List

- An ordered sequence of values
- Different data types, can be stored.
- Enclosed in square brackets.
- >>>L=[1,'Hello', 5.7, True]
- Accessing listing elements exactly same way as strings.
- But Lists are mutable. One individual inside a list can be changed.
- >>>L[0]=2
- >>>L
- >>>L=[2,'Hello', 5.7, True]

Loops and lists

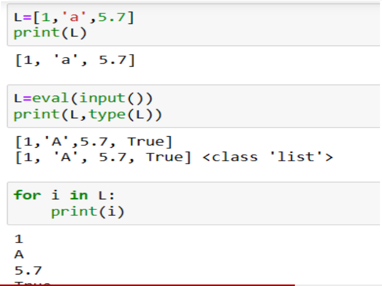
- >>>for i in L:
  >>>   print(i)
  >>>2
  >>>'Hello'
  >>>5.7
  >>>True

Creating a list

- Using assignment operator.
- Using eval() function.
- list() method.
- Using append, extend and insert methods.

Check Point:-

Create a list of 5 integers, using assignment operator. Find the sum of all the elements present in the list.
Find the largest element in the above list, and find it's index.



```
L=[1,'a',5.7]
print(L)
```

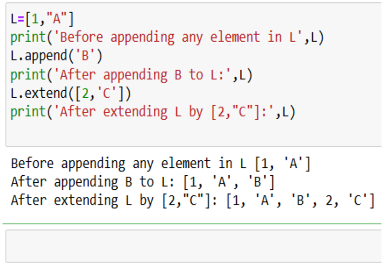
```
[1, 'a', 5.7]
```

```
L=eval(input())
print(L,type(L))
```

```
[1,'A',5.7, True]
[1, 'A', 5.7, True] <class 'list'>
```

```
for i in L:
    print(i)
```

```
1
A
5.7
```



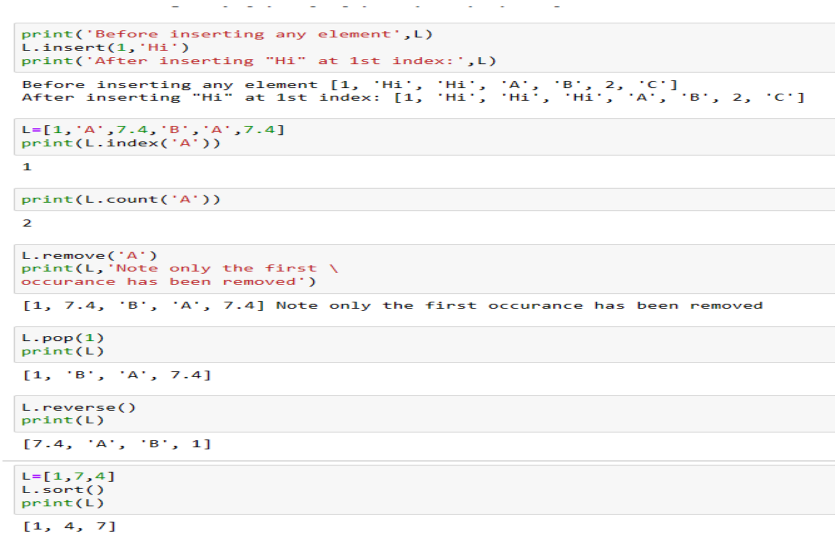
```
L=[1,"A"]
print('Before appending any element in L',L)
L.append('B')
print('After appending B to L:',L)
L.extend([2,'C'])
print('After extending L by [2,"C"]:',L)
```

```
Before appending any element in L [1, 'A']
After appending B to L: [1, 'A', 'B']
After extending L by [2,"C"]: [1, 'A', 'B', 2, 'C']
```

List method

L.append(e)	Inserts the element e at the end of the list L.
L.extend(L2)	Inserts in the items in the sequence L2 at the end of the elements of the List L.
L.remove(e)	Removes the first occurrence of the element e in the List L.
L.insert(i,e)	Inserts element e at index i in the list L
L.pop(i)	Returns the element from the list L at index i, while removing it from the list.
L.index(e)	Returns index of an object e, if present in the list L.
L.count(e)	Returns count of occurrences of the object e in the List L.
L.sort()	Sorts the elements of the list L.
L.reverse()	Reverses the order of elements in the list L.

Examples



```
print('Before inserting any element',L)
L.insert(1,'Hi')
print('After inserting "Hi" at 1st index:',L)
```

```
Before inserting any element [1, 'Hi', 'Hi', 'A', 'B', 2, 'C']
After inserting "Hi" at 1st index: [1, 'Hi', 'Hi', 'Hi', 'A', 'B', 2, 'C']
```

```
L=[1,'A',7.4,'B','A',7.4]
print(L.index('A'))
```

```
1
```

```
print(L.count('A'))
```

```
2
```

```
L.remove('A')
print(L,'Note only the first \
occurance has been removed')
```

```
[1, 7.4, 'B', 'A', 7.4] Note only the first occurance has been removed
```

```
L.pop(1)
print(L)
```

```
[1, 'B', 'A', 7.4]
```

```
L.reverse()
print(L)
```

```
[7.4, 'A', 'B', 1]
```

```
L=[1,7,4]
L.sort()
print(L)
```

```
[1, 4, 7]
```

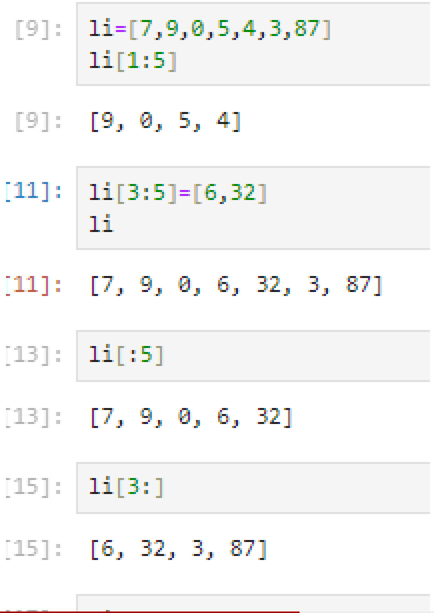
List

Check Point

Create a list NAMES containing the names of 10 students and in another list ROLL keep their roll numbers. Given a particular roll no, you remove the roll no from the list ROLL and and the corresponding name from the lists.

- The del operator is used to remove a sub sequence of elements (start:end:increment) from a list.
- if we simply write del L, then the list will be deleted and we can't access the list L.
- sorted() function sorts the given sequence. This function doesn't effect the original sequence.
- Syntax: sorted (iterable, key, reverse=False)
- Key(optional): a basis of comparison.
- Reverse(optional): If set True, then the iterable would be sorted in reverse (descending) order, by default it is set as False.
- sorted functions returns a list object.

Slicing in Lists



```
[9]:  li=[7,9,0,5,4,3,87]
      li[1:5]

[9]:  [9, 0, 5, 4]

[11]: li[3:5]=[6,32]
      li

[11]: [7, 9, 0, 6, 32, 3, 87]

[13]: li[:5]

[13]: [7, 9, 0, 6, 32]

[15]: li[3:]

[15]: [6, 32, 3, 87]
```

2-D lists

- List comprehension: A shorter syntax to create a new list based on the values of an existing list.
- newlist = [expression for item in iterable if condition == True]
- Two dimensional list: A 2D list is a list of lists, which represents a table-like structure with rows and columns.
- >>>L=[[i1,i2],[j1,j2]]
- The above list of dimension 2×2
  - >>>L[0]=[i1,i2]   >>>L[1]=[j1,j2]
  - >>>L[0][0]=i1   >>>L[0][1]=i2
  - >>>L[1][0]=j1   >>>L[1][1]=j2

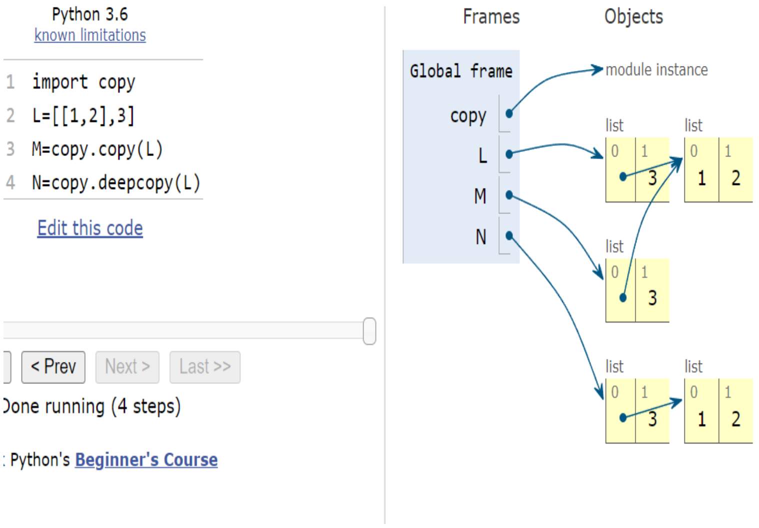
Check Point

Create a 2-D list of dimension 2×5. First row containing first five odd numbers, and second row containing first five even numbers.
L=[[1,3,5,7,9],[0,2,4,6,8]].

Deepcopy vs copy

- When we use the = operator, It only creates a new variable that shares the reference of the original object. This happens in case of list also.
- So in the above method, change will one list will change the other list.
- In Python, there are two ways to create real copies:
  - Copy(Sometimes called shallow copy)
  - Deep Copy
- To make these copy work, we use the copy module.
- shallow copy creates a new object which stores the reference of the original elements.
- shallow copy doesn't create a copy of nested objects, instead it just copies the reference of nested objects.
- deep copy creates a new object and recursively adds the copies of nested objects present in the original elements.

Deepcopy vs copy



Python 3.6
known limitations

```
import copy
L=[[1,2],3]
M=copy.copy(L)
N=copy.deepcopy(L)
```

Edit this code

< Prev   Next >   Last >>

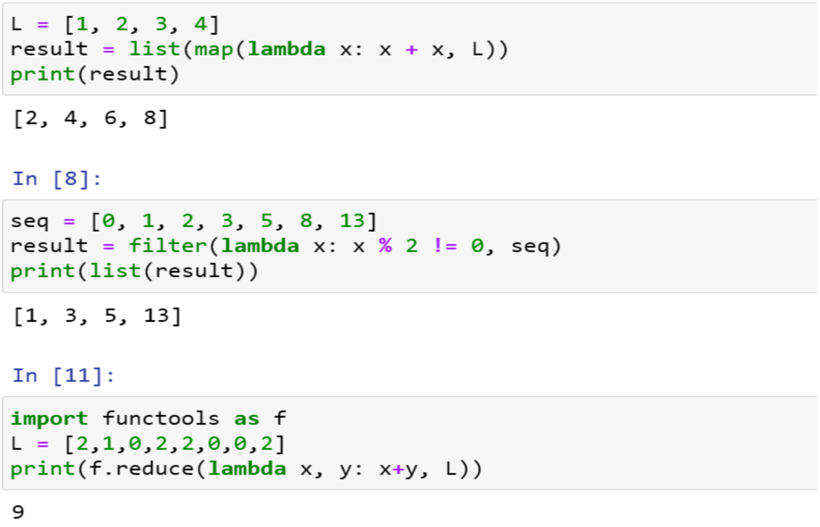
Done running (4 steps)

Python's Beginner's Course

map,reduce, and filter

- Map: Returns a list of the results after applying the given function to each item of a given sequence.
- Syntax: map(fun, sequence)
- return is a map object, which has to be converted to list.
- filter(fun,sequence) method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.
- Syntax: filter(function, sequence)
- The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements
- This function is defined in "functools" module.
- first two elements of sequence are picked and the result is obtained. Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.

Example



```
L = [1, 2, 3, 4]
result = list(map(lambda x: x + x, L))
print(result)
```

```
[2, 4, 6, 8]
```

In [8]:

```
seq = [0, 1, 2, 3, 5, 8, 13]
result = filter(lambda x: x % 2 != 0, seq)
print(list(result))
```

```
[1, 3, 5, 13]
```

In [11]:

```
import functools as f
L = [2,1,0,2,2,0,0,2]
print(f.reduce(lambda x, y: x+y, L))
```

```
9
```

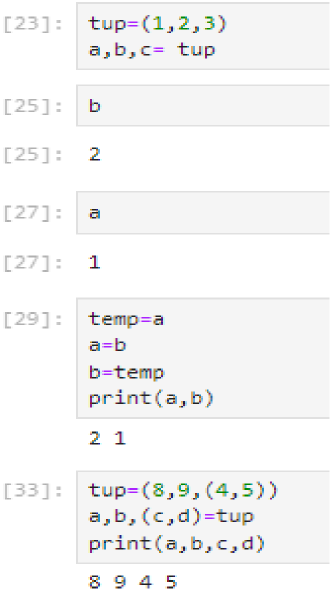
tuples

- tuples: order sequence of objects, contains different data types. Immutable version of lists.
- An element of a tuple may be a mutable object like list.
- comma separated elements inside parentheses.
- indexing and slicing same way as in a list
- parentheses are optional but at least one comma need to be there.
- can create tuples by using assignment operator or using tuple() method on a sequence.
- If an object inside a tuple is mutable, such as a list, you can modify it in place.
- unpacking tuples: extract the values back into variables
- In unpacking of tuple number of variables on left-hand side should be equal to number of values in given tuple

tuples operations

Multiplication operator tuple* k	creates k copies of the tuple
concatenation operator +	concatenates two tuples
len(tuple)	gives length of the tuple
min(tuple)	gives minimum element of the tu
sum(tuple)	sums all elements of the tuple
tuple.index(e)	returns index of the first occurre
tuple.count(e)	returns count of occurrence of element e

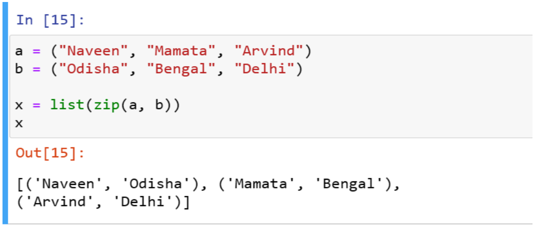
Unpacking Tuples



```
[23]: tup=(1,2,3)
      a,b,c= tup

[25]: b

[25]: 2

[27]: a

[27]: 1

[29]: temp=a
      a=b
      b=temp
      print(a,b)

      2 1

[33]: tup=(8,9,(4,5))
      a,b,(c,d)=tup
      print(a,b,c,d)

      8 9 4 5
```

zip function on tuple

- zip() is used to produce an zip object(iterable object).
- syntax: zip(iterable1,iterable2)
- The i-th element of the zip object is a tuple containing i-th element from each iterable object passed as argument to the zip function.



```
In [15]:

a = ("Naveen", "Mamata", "Arvind")
b = ("Odisha", "Bengal", "Delhi")

x = list(zip(a, b))
x

Out[15]:

[('Naveen', 'Odisha'), ('Mamata', 'Bengal'),
('Arvind', 'Delhi')]
```

Check Point

Create two lists one with the name of the chief ministers and another list with the corresponding states. Use zip() to run a for loop over the two lists simultaneously, and your output should be, "x is the chief minister of the state y". Print this line for all the states.

set

- set is unordered collection data type that is iterable, mutable and has no duplicate elements.
- Curly braces and elements separated by commas
- creating a set using set() method
- set elements has to be immutable.
- indexing and slicing, + and * does not work in case of sets
- min, max, sum and len work for sets in the same manner as defined for lists

set functions

S.add()	add a single 'element' to S
S.update()	adding multiple elements to the set the argument can be list,set, or tuple
S.remove()	removes the element which is passed as an argument
S.pop()	removes an element arbitrarily
S.clear()	remove all the elements of a set, without deleting the object
S.copy()	create a copy of the set
S1.union(S2)	Returns union of sets S1 and S2
S1.intersection(S2)	Returns a set containing elements common in the sets S1 and S
S1.difference(S2)	Returns a set containing elements in the set S1 but not in the set S2
S1.symmetric_difference(S2)	Returns a set containing elements that are in one of the two sets S1 and S2,
S1<=S2	Returns True if S1 is a subset of S2
S1>=S2	Returns True if S1 is a super set of S2

Dictionary

- Unordered sequence of key-value pairs
- As list have indices, in case of dictionaries, we name the indices and call it keys.
- The corresponding element to each key is called it's value.
- key and value are separated by colon. Each key:value are separated by commas. Enclosed in curly braces.
- Accessing elements D[key_name]
- keys of dictionary has to be unique.
- Keys in a dictionary are immutable. Lists can be used as keys.
- values in a dictionary can be mutable, can be repeated, even can be dictionary also.

Dictionary

- dict.keys() returns an object of keys.
- dict.values() returns an object of values.
- dict.items() returns an object of (key,value) tuples
- loops over dictionary
  - for i in dict:#loop will run over all the keys of dictionary
  - for i in dict.values():#loop will run over all the values of the dictionary
- Membership operator 'in', and functions min,max,sum apply only to the keys in a dictionary. So applying them on 'dict' is equivalent to apply them on 'dict.keys()'

Check Point

Write a function that takes an integer as input argument and returns the integer using words. For example if the input is 4721 then the function should return the string "four seven two one".

functions in Dictionary

D.clear()	Removes all key:value pairs from the dictionary D
del D[5]	Removes the key:value pair corresponding to the key 5
D.copy()	creates a copy of the dictionary D
D1.update(D2)	Adds the key value pairs of dictionary D2 to the dictionary D1
D.get(key,default)	For the specified key, the function returns the associat value. Returns the default value in case key is not present in the dictionary D.

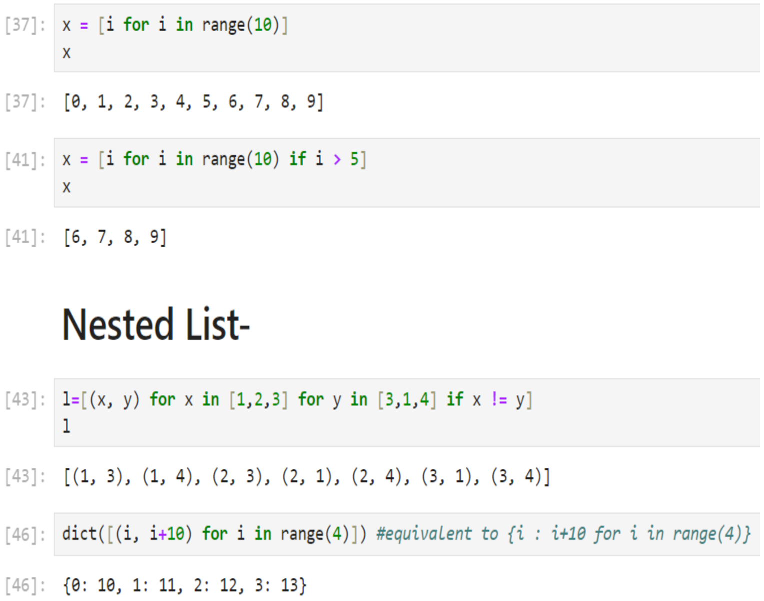
Check point

Let D={'A':1,'B':2,'C':1,'D':2} be given. Create another dictionary D1 such that D1={1:['A','c'],2:['B','D']}. Invert the dictionary, make the values to be keys and store the keys in a list.

List, Set, Dictionary Comprehensions

- Python comprehension is a set of looping and filtering instructions for evaluating expressions and producing sequence output.
- List and dictionary comprehensions are a powerful substitute to for-loops and also lambda functions.
- List comprehensions are constructed from brackets containing an expression, followed by a for clause, that is [item-expression for item in iterator], can then be followed by further for or if clauses: [item-expression for item in iterator if conditional].

List, Set, Dictionary Comprehensions Examples



```
[37]: x = [i for i in range(10)]
      x

[37]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[41]: x = [i for i in range(10) if i > 5]
      x

[41]: [6, 7, 8, 9]
```

Nested List-

```
[43]: l=[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
      l

[43]: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

[46]: dict([(i, i+10) for i in range(4)]) #equivalent to {i : i+10 for i in range(4)}

[46]: {0: 10, 1: 11, 2: 12, 3: 13}
```