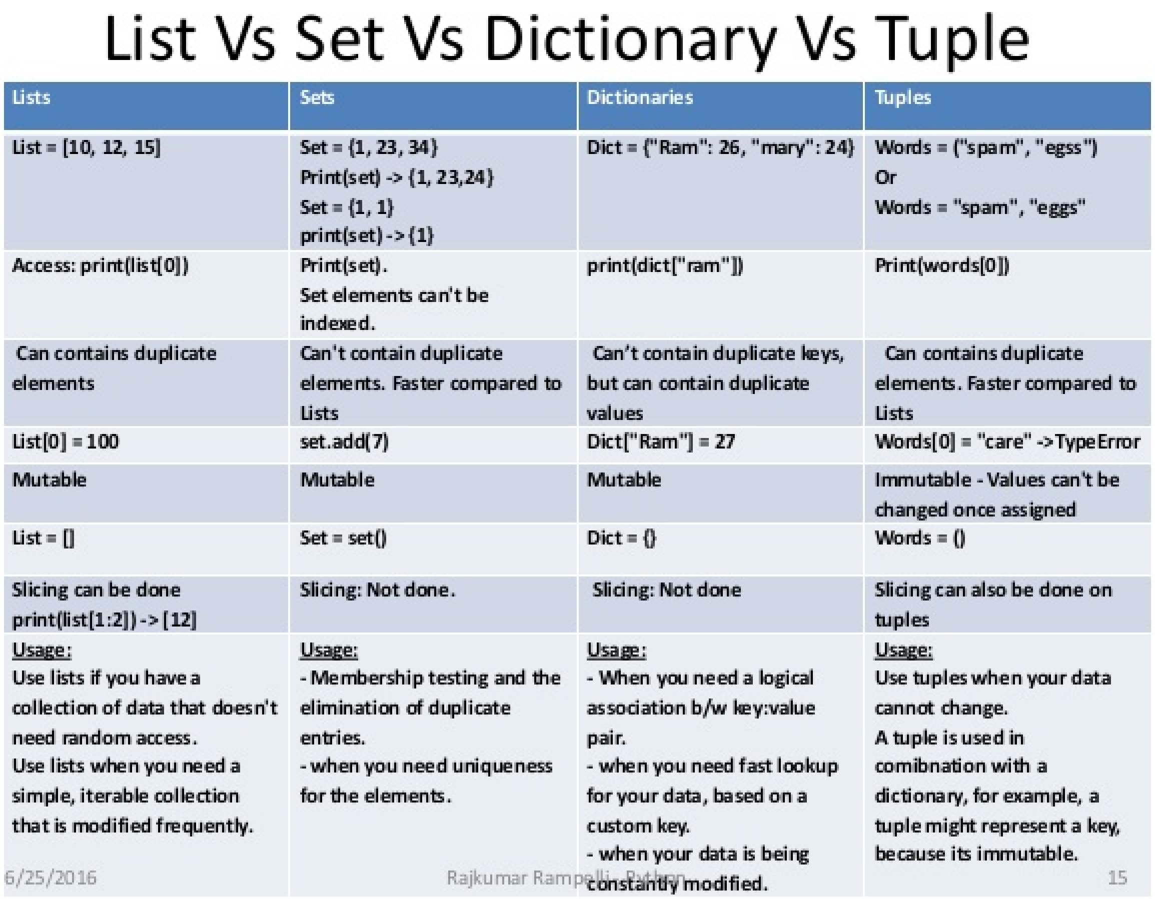


List Vs Set Vs Dictionary Vs Tuple

Lists	Sets	Dictionaries	Tuples

List = [10, 12, 15]	Set = {1, 23, 34}
	Print(set) -> {1, 23,24}
	Set = {1, 1}
	print(set) -> {1}
		Dict = {"Ram": 26, "mary": 24}	Words = ("spam", "egss")
			Or
			Words = "spam", "eggs"

Access: print(list[0])	Print(set). Set elements can't be indexed.	print(dict["ram"])	Print(words[0])
Can contains duplicate elements	Can't contain duplicate elements. Faster compared to Lists	Can't contain duplicate keys, but can contain duplicate values	Can contains duplicate elements. Faster compared to Lists
List[0] = 100	set.add(7)	Dict["Ram"] = 27	Words[0] = "care" ->TypeError
Mutable	Mutable	Mutable	Immutable - Values can't be changed once assigned
List = []	Set = set()	Dict = {}	Words = ()
Slicing can be done print(list[1:2]) -> [12]	Slicing: Not done.	Slicing: Not done	Slicing can also be done on tuples
Usage: Use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	Usage: - Membership testing and the elimination of duplicate entries. - when you need uniqueness for the elements.	Usage: - When you need a logical association b/w key:value pair. - when you need fast lookup for your data, based on a custom key. - when your data is being constantly modified.	Usage: Use tuples when your data cannot change. A tuple is used in comibnation with a dictionary, for example, a tuple might represent a key, because its immutable.

6/25/2016 Rajkumar Rampelli - Python 15

List Data Type

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets: [ ]
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- List items are ordered, changeable, and allow duplicate values.

    - **Ordered:** When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.
    - **Changeable:** The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
    - **Allow duplicate values:** Since lists are indexed, lists can have items with the same value:
- A list is not merely a collection of objects, it is an ordered collection of objects. The order in which you specify the elements when you define a list is an innate characteristic of that list and is maintained for that list’s lifetime.
- A list can contain any assortment of objects. The elements of a list can all be the same or different type. Lists can even contain complex objects, like functions, classes, and modules.
- A list can contain any number of objects, from zero to as many as your computer’s memory will allow.

In [1]:

```
# creating a list:

mylist = ["Ram", "Shyam", "Sita", 7, 12.5]
print(mylist)
```

```
['Ram', 'Shyam', 'Sita', 7, 12.5]
```

In [2]:

```
# creating an empty list:

a = []
print(type(a))
print(a)
```

```
<class 'list'>
[]
```

In [3]:

```
# We can also create a new list using list constructor: list()

mylist = list(("apple", "banana", "cherry"))
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

We cannot assign values to a list like **a[0] = "Hi"**, we have to use list methods for that.

In [4]:

```python
# Length of the list can be determined using len() function

b = ["apple", "banana", "cherry"]
print(type(b))
print(len(b))
```

```
<class 'list'>
3
```

In [5]:

```python
# A list can contain different data types:

a = ["abc", 34, True, 40.5, "male"]
print(type(a))
print(a)
```

```
<class 'list'>
['abc', 34, True, 40.5, 'male']
```

In [6]:

```python
# A list can contain complex objects too

def fun():
    pass

def m1():
    pass

l = [fun, m1, "10"]
print(type(l))
print(l)
```

```
<class 'list'>
[<function fun at 0x7faf2c4ccee0>, <function m1 at 0x7faf2c4ccd30>, '10']
```

In [7]:

```python
# List objects needn’t be unique. A given object can appear in a list multiple times

marks= [34, 54, 67, 87, 22, 54]
print(marks)
```

```
[34, 54, 67, 87, 22, 54]
```

Access list items

List items are indexed and you can access them by referring to the index number.

In [8]:

```
a = ["abc", 34, True, 40.5, "male"]
print(a[2])
```

```
True
```

List also allows negative indexing

In [9]:

```
# Negative indexing
print(a[-1])
```

```
male
```

Iterating over a list

In [10]:

```
# Method 1: Using for loop

list = [1, 3, 5, 7, 9]
for i in list:
    print(i)
```

```
1
3
5
7
9
```

In [11]:

```
# Method 2: For loop and range()

list = [1, 3, 5, 7, 9]
length = len(list)

for i in range(length):             # range(length) can also be replaced with range(len(list))
    print(i, list[i])
```

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

In [12]:

```
# Method 3: Using while loop
list = [1, 3, 5, 7, 9]
length = len(list)
i = 0
while i < length:                         # Use len(list) instead of length to reduce LOC
    print(list[i])
    i += 1
```

```
1
3
5
7
9
```

In [13]:

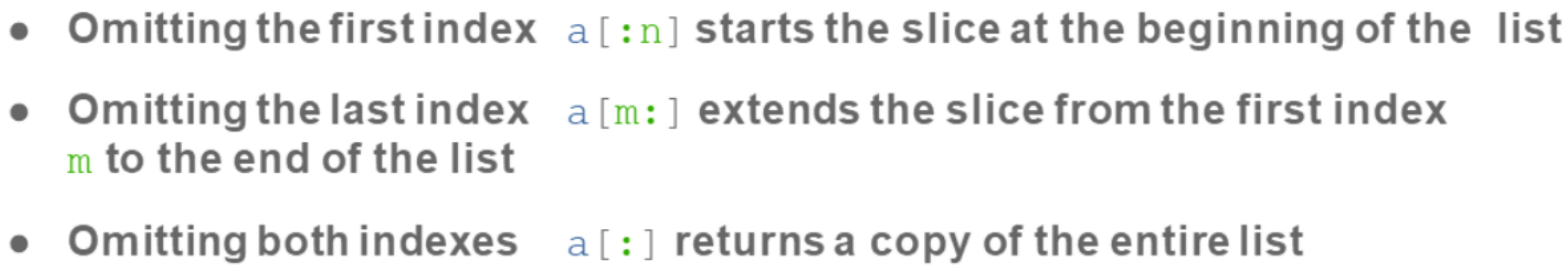
```
# Method 4: Using list comprehension: Covered in further topics
```

In [14]:
```
# Method 5: Using enumerate()
# If we want to convert the list into an iterable list of tuples (or get the index based on a co
# for example in linear search you might need to save the index of minimum element), you can use
# enumerate() function.

list = [1, 3, 5, 7, 9]
for i, val in enumerate(list):
    print (i,",",val)
```

```
0 , 1
1 , 3
2 , 5
3 , 7
4 , 9
```

List slicing

You can specify a range of indexes by specifying where to start and where to end the range. If a is a list, the expression a[m:n] returns the portion of a from index m to, but not including, index n: Both positive and negative indices can be specified.



- Omitting the first index a[:n] starts the slice at the beginning of the list
- Omitting the last index a[m:] extends the slice from the first index m to the end of the list
- Omitting both indexes a[:] returns a copy of the entire list

In [15]:
```
a = ["abc", 34, True, 40.5, "male"]
print(a[1:4])
```

```
[34, True, 40.5]
```

In [16]:

```
print(a[-5:-1])
```

```
['abc', 34, True, 40.5]
```

In [17]:

```
marks= [34, 54, 67, 87, 22, 54]
```

In [18]:

```
marks[:len(marks)]
```

Out[18]:

```
[34, 54, 67, 87, 22, 54]
```

In [19]:

```
marks[:]
```

Out[19]:

```
[34, 54, 67, 87, 22, 54]
```

You can specify a stride—either positive or negative. The syntax for reversing a list works the same way it does for strings: **a[::-1]**

In [20]:

```
# Reversing a list
marks[::-1]
```

Out[20]:

```
[54, 22, 87, 67, 54, 34]
```

In [21]:

```
marks[len(marks):0:-2]
```

Out[21]:

```
[54, 87, 54]
```

In [22]:

```
marks[::-2]
```

Out[22]:

```
[54, 87, 54]
```

In [23]:

```
a=[0,11,22,33,44,55,66,77,88,99]
```

In [24]:

```
a[0:7:2]
```

Out[24]:

```
[0, 22, 44, 66]
```

```
In [25]: a[0:7]

Out[25]: [0, 11, 22, 33, 44, 55, 66]
```

```
In [26]: a[2:7:1]

Out[26]: [22, 33, 44, 55, 66]
```

```
In [27]: a[7:2:1]

Out[27]: []
```

```
In [28]: a[7:2:-1]

Out[28]: [77, 66, 55, 44, 33]
```

```
In [29]: a[:7:1]

Out[29]: [0, 11, 22, 33, 44, 55, 66]
```

```
In [30]: a[5::1]

Out[30]: [55, 66, 77, 88, 99]
```

```
In [31]: a[:-5:-1]

Out[31]: [99, 88, 77, 66]
```

```
In [32]: a[-5::-1]

Out[32]: [55, 44, 33, 22, 11, 0]
```

```
In [33]: a[5:-1:1]

Out[33]: [55, 66, 77, 88]
```

In [34]:

```
a[-1:5:-1]
```

Out[34]:

```
[99, 88, 77, 66]
```

in and not in are membership operators and can be used with lists. A membership operator used on a list:

1. The in and not in operators: Returns True if the first operand is contained within the second Returns False otherwise
2. The concatenation (+) and replication () *operators: The concatenation (+) operator concatenates the operands. The replication ()* operator creates multiple concatenated copies.
3. len() returns the length of the list. min() returns the object from the list with the smallest value. max() returns the object from the list with the highest value.

In [35]:

```
# To determine if a specified item is present in a list use the 'in' keyword:
fruits = ["apple", "papaya", "banana", "cherry"]
print("apple" in fruits)
print("orange" in fruits)
```

```
True
False
```

In [36]:

```
fruits = ["apple", "papaya", "banana", "cherry"]
if "apple" not in fruits:
    print("Present")
else:
    print("not Present")
```

```
not Present
```

In [37]:

```
new = fruits + ['kiwi','orange']
print(fruits)
print(new)
```

```
['apple', 'papaya', 'banana', 'cherry']
['apple', 'papaya', 'banana', 'cherry', 'kiwi', 'orange']
```

In [38]:

```python
print(fruits * 2)
fruits
```

```
['apple', 'papaya', 'banana', 'cherry', 'apple', 'papaya', 'banana', 'cherry']
```

Out[38]:

```
['apple', 'papaya', 'banana', 'cherry']
```

In [39]:

```python
fruits = ["apple", "papaya", "banana", "cherry"]
```

In [40]:

```python
len(fruits), max(fruits), min(fruits)
```

Out[40]:

```
(4, 'papaya', 'apple')
```

In [41]:

```python
# ord(c,/): Returns the Unicode code point for a one-character string.
ord('a'), ord('p')
# ord('hello')
```

Out[41]:

```
(97, 112)
```

In [42]:

```python
fruits= fruits + [10]
fruits
```

Out[42]:

```
['apple', 'papaya', 'banana', 'cherry', 10]
```

In [43]:

```python
# Cannot compare int with a string
max(fruits)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-bb1367d0578f> in <module>
      1 # Cannot compare int with a string
----> 2 max(fruits)

TypeError: '>' not supported between instances of 'int' and 'str'
```

Change List Items

In [44]:

```
# To change the value of a specific item, refer to the index number
thislist = ["apple", "banana", "cherry"]
print(thislist)

thislist[1] = "orange"
print(thislist)
```

```
['apple', 'banana', 'cherry']
['apple', 'orange', 'cherry']
```

In [45]:

```
# To change the value of items within a specific range, define a list with the new values, and
# refer to the range of index numbers where you want to insert the new values:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
print(thislist)

thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange', 'kiwi', 'mango']
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

List Methods: Part 1

No return value, change the original list

- mylist.insert(<index>,<obj>)
- mylist.append(<obj>)
- mylist.extend(<iterable>)
- mylist.remove(<obj>)
- mylist.clear()
- mylist.sort(<key=None>,<reverse=False>)

In [46]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
fruits
```

Out[46]:

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

In [47]:

```
fruits.insert?
```

In [48]:

```
# To insert element at any index of list use insert()
fruits.insert('banana')
fruits
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-998d74c18500> in <module>
      1 # To insert element at any index of list use insert()
----> 2 fruits.insert('banana')
      3 fruits

TypeError: insert expected 2 arguments, got 1
```

In [49]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
fruits.insert(2,'banana')
fruits
```

Out[49]:

```
['apple', 'blackcurrant', 'banana', 'watermelon', 'orange', 'kiwi', 'mango']
```

In [50]:

```
fruits.append?
```

In [51]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
fruits.append('new entry')
fruits
```

Out[51]:

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'new entry']
```

In [52]:

```
l = []
for i in range(5):
    l.append(input())
    print(l)
```

```
one
['one']
two
['one', 'two']
three
['one', 'two', 'three']
four
['one', 'two', 'three', 'four']
five
['one', 'two', 'three', 'four', 'five']
```

In [53]:

```
print(l)
```

```
['one', 'two', 'three', 'four', 'five']
```

In [54]:

```
fruits.extend?
```

In [55]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
mylist = ['Hello','World','Hi']
fruits.extend(mylist)
fruits
```

Out[55]:

```
['apple',
 'blackcurrant',
 'watermelon',
 'orange',
 'kiwi',
 'mango',
 'Hello',
 'World',
 'Hi']
```

In [56]:

```
fruits.remove?
```

In [57]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
fruits.remove('orange')
fruits
```

Out[57]:

```
['apple', 'blackcurrant', 'watermelon', 'kiwi', 'mango']
```

In [58]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango','watermelon']
fruits.remove('watermelon')
fruits
```

Out[58]:

```
['apple', 'blackcurrant', 'orange', 'kiwi', 'mango', 'watermelon']
```

In [59]:

```
fruits.clear?
```

In [60]:

```
fruits.clear()
fruits
```

Out[60]:

```
[]
```

In [61]:

```
fruits.sort?
```

In [62]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango','watermelon']
fruits.sort()
fruits
```

Out[62]:

```
['apple',
 'blackcurrant',
 'kiwi',
 'mango',
 'orange',
 'watermelon',
 'watermelon']
```

In [63]:

```
# You cannot sort a list that contains BOTH string values AND numeric values.
fruits = ['apple', 'blackcurrant', 'watermelon', 10, 'kiwi', 'mango',40.5]
fruits.sort()
fruits
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-63-7e0f512794ea> in <module>
      1 # You cannot sort a list that contains BOTH string values AND numeric values.
      2 fruits = ['apple', 'blackcurrant', 'watermelon', 10, 'kiwi', 'mango',40.5]
----> 3 fruits.sort()
      4 fruits

TypeError: '<' not supported between instances of 'int' and 'str'
```

In [64]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango','watermelon']
fruits.sort(reverse=True)
fruits
```

Out[64]:

```
['watermelon',
 'watermelon',
 'orange',
 'mango',
 'kiwi',
 'blackcurrant',
 'apple']
```

In [65]:

```
fruits = ['mango','apple', 'blackcurrant', 'watermelon','orange', 'kiwi', 'watermelon']
fruits.sort(key=len)
fruits
```

Out[65]:

```
['kiwi',
 'mango',
 'apple',
 'orange',
 'watermelon',
 'watermelon',
 'blackcurrant']
```

In [66]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango','watermelon']
fruits.sort(key=len, reverse=True)
fruits
```

Out[66]:

```
['blackcurrant',
 'watermelon',
 'watermelon',
 'orange',
 'apple',
 'mango',
 'kiwi']
```

List Methods: Part 2

With return values:

- mylist.pop(<index=-1>): Returns the item removed
- mylist.index(<obj>,<start>[,<end>]])
- mylist.count(<obj>)
- mylist.copy(): Returns a shallow copy

In [67]:

```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
fruits
```

Out[67]:

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

In [108]:

```
fruits.pop?
```

In [69]:

```
fruits.pop()
```

Out[69]:

```
'mango'
```

In [70]:
```
fruits
```

Out[70]:
```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi']
```

In [71]:
```
fruits.pop(2)
```

Out[71]:
```
'watermelon'
```

In [72]:
```
fruits
```

Out[72]:
```
['apple', 'blackcurrant', 'orange', 'kiwi']
```

In [73]:
```
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
```

In [74]:
```
fruits.index?
```

In [75]:
```
fruits.index('orange')
```

Out[75]:
```
3
```

In [76]:
```
fruits.index('mango')
```

Out[76]:
```
5
```

In [77]:
```
# Find index of multiple occurrences of an object
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
for i, value in enumerate(fruits):
    if value=='orange':
        print(i, value)
```

```
3 orange
6 orange
```

In [78]:
```
fruits.count?
```

In [79]:

```python
fruits.count('watermelon')
```

Out[79]:

```
1
```

In [80]:

```python
fruits.count('orange')
```

Out[80]:

```
2
```

In [81]:

```python
fruits.copy?
```

In [82]:

```python
newlist = fruits.copy()
```

In [83]:

```python
print(newlist)
print(fruits)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
```

Nested Lists

An element in the list can be of any type, which means it can be a list too. A list can contain sublist which in turn can have another sublist, and so on to arbitrary depth.

In [84]:

```python
l = [['Ram', 78, 12], ['Shyam', 21, 44], ['Sita', 46, 97]]
```

In [85]:

```python
for i in l:
    print(i)
```

```
['Ram', 78, 12]
['Shyam', 21, 44]
['Sita', 46, 97]
```

In [86]:

```
for i in l:
    for j in i:
        print(j)
    print('\n')
```

```
Ram
78
12


Shyam
21
44


Sita
46
97
```

In [87]:

```
for i in range(len(l)):
    print(l[i][2])
```

```
12
44
97
```

In [88]:

```
sum = 0
for i in range(len(l)):
    sum += l[i][2]
print(sum)
```

```
153
```

In [89]:

```
x = ['a',['bb',['ccc','ddd'], 'ee', 'ff'], 'g', ['hh','ii'], 'jj']
```

```
In [90]: # ddd
         x[1][1][1]

Out[90]: 'ddd'
```

```
In [91]: # ee
         x[1][2]

Out[91]: 'ee'
```

```
In [92]: # hh
         x[3][0]

Out[92]: 'hh'
```

```
In [93]: # ccc
         x[1][1][0]

Out[93]: 'ccc'
```

```
In [94]: # ii
         x[3][1]

Out[94]: 'ii'
```

```
In [95]: # g
         x[2]

Out[95]: 'g'
```

sort() vs sorted()

list.sort(key=None, reverse=False)

- works only for list data structure
- in place sorting (modifies the existing list)

- efficient than sorted() if we do not need existing list

sorted(iterable, key=None, reverse=False)

- works for any iterable
- creates a new modified list

In [96]:

```python
# sorting objects on the basis of their length:
fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
fruits.sort(key=len)
fruits
```

Out[96]:

```
['kiwi', 'apple', 'mango', 'orange', 'orange', 'watermelon', 'blackcurrant']
```

In [97]:

```python
# sorting objects on the basis of their length:
def myfun(s):
    return len(s)

fruits = ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango', 'orange']
fruits.sort(key=myfun,reverse=True)
fruits
```

Out[97]:

```
['blackcurrant', 'watermelon', 'orange', 'orange', 'apple', 'mango', 'kiwi']
```

In [98]:

```python
# Sorting on the basis of marks:
def myfun1(l):
    return l[1]

marks = [['Ram', 78, 12], ['Shyam', 21, 44], ['Sita', 46, 97]]
# marks.sort(key=myfun1)
# print(marks)
newmarks = sorted(marks,key=myfun1)
print(newmarks)
```

```
[['Shyam', 21, 44], ['Sita', 46, 97], ['Ram', 78, 12]]
```

In [99]:

```python
# Sorting on the basis of marks using lambda function:

marks = [['Ram', 78, 12], ['Shyam', 21, 44], ['Sita', 46, 97]]
# marks.sort(key=lambda l: len(l[0]))
# print(marks)
new_marks = sorted(marks,key= lambda l: len(l[0]))
print(new_marks)
```

```
[['Ram', 78, 12], ['Sita', 46, 97], ['Shyam', 21, 44]]
```

In [100]:

```python
d = {'Ram': 45, 'Shyam': 12, 'Arjun': 45}
d_new = sorted(d)
print(d_new, type(d_new))

# To sort on basis of an criteria, we use items() function. Will learn in dictionary topic.
```

```
['Arjun', 'Ram', 'Shyam'] <class 'list'>
```

Operator Module Functions

Python provides convenience functions to make key functions easier and faster. The operator module has itemgetter(), attrgetter() and methodcaller() function.

In [101]:

```python
from operator import itemgetter, attrgetter
```

In [102]:

```python
marks = [['Ram', 78, 12], ['Shyam', 21, 44], ['Sita', 46, 97]]
d_new = sorted(marks, key = itemgetter(2))
print(d_new)
```

```
[['Ram', 78, 12], ['Shyam', 21, 44], ['Sita', 46, 97]]
```

In [103]:

```
class Student:
    def __init__(self,name,marks,age):
        self.name = name
        self.marks = marks
        self.age = age
    def __repr__(self):
        return repr((self.name, self.marks, self.age))
```

In [104]:

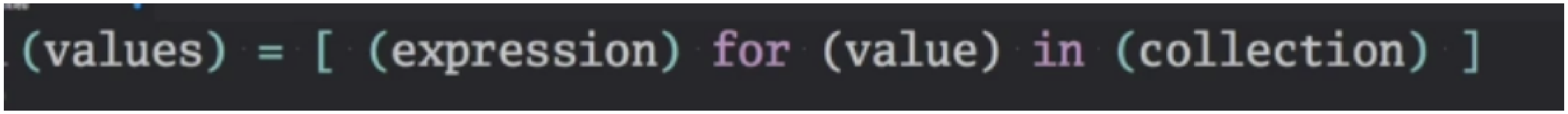
```
student = [ Student('Ram',45,16),
            Student('Arjun',97,17),
            Student('Suresh', 67, 15)]
new = sorted(student,key = attrgetter('marks'))
print(new)
```

```
[('Ram', 45, 16), ('Suresh', 67, 15), ('Arjun', 97, 17)]
```

List Comprehensions

list comprehensions are powerful if used correctly and can lead to more concise and readable code.

https://dbader.org/blog/list-dict-set-comprehensions-in-python (https://dbader.org/blog/list-dict-set-comprehensions-in-python)



```
(values) = [ (expression) for (value) in (collection) ]
```

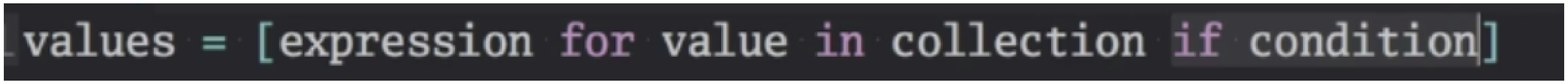
In [105]:

```
# Find square of every number and store it in a new list

l = [1,2,3,4,5,6,7,8]
# new = list()
# for i in l:
#     new.append(i*i)
# new

new = [(i*i) for i in l]
new
```

Out[105]:

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

Conditional statements can be added to Python list comprehensions in order to filter out data.



```
values = [expression for value in collection if condition]
```

In [107]:

```
# Find square of every even number and store it in a new list

l = [1,2,3,4,5,6,7,8]
# new = list()
# for i in l:
#     if(i%2==0):
#         new.append(i*i)
# new

new = [(i*i) for i in l if (i%2==0)]
new
```

Out[107]:

```
[4, 16, 36, 64]
```

In [ ]:

```

```