Chp04 Lists, Tuples and Dictionaries

Python Collections

There are four built-in collection data types in the Python programming language

- **Lists**
- **Tuples**
- **Dictionaries**
- **Sets**

In this course, the first three will be covered (i.e. **Lists, tuples** and **dictionaries**)

Chp04A Lists

References

1. https://www.w3schools.com/python/python_lists.asp w3school Python Tutorial (lists)

2. https://www.tutorialspoint.com/python3/python_lists.htm Tutorialspoint (lists)

3. Learning Python, 5th (mark Lutz Oreilly 2013) – chapter 4 - Introducing Python Object Types

4. Python for Everybody – Chapters 8 Lists

5. https://docs.python.org/3/tutorial/index.html Python Tutorial - check Table of Contents

Definition

A list is a collection which is **ordered** and **changeable** **(mutable)**. In Python list items are enclosed in square brackets **[ ]** and are separated by comma. Example:

```
names=["Kimario","Asha","George", "Hamis"]
print(names)
```

Results:

```
['Kimario', 'Asha', 'George', 'Hamis']
```

`print(type(names))` – results into **`<class 'list'>`**

Note: Items in a list are not necessarily have the same type. Example:

```
list2=["Juma", 93.5, 34, True]
print(list2)
```

Results: `['Juma', 93.5, 34, True]`

`chp04Aex01List01`

Creating a new list using list() constructor

The **list** constructor can be used to make a list.

Example: chp04Aex02List02

```
countries=list(["Tanzania","Kenya","Uganda","
Rwanda","Burundi","Somalia"])
print(countries)
['Tanzania', 'Kenya', 'Uganda', 'Rwanda', 'Burundi',
'Somalia']

print(list("abcd")) #['a', 'b', 'c', 'd‘]
```

- Can also use range to create list

```
    numbers=list(range(10)) # [0,1,2,...9]
```

To create empty list, use:

```
listName=[]  or listName=list()
```

- To access the list item, use index number The first index is 0. To determine the number of items in the list use **len** function: `len(listName)`
- The last index is **len(listName)-1** otherwise you get IndexError: IndexError: list index out of range

```
names=["Kimario","Asha","George", "Hamis"]
print(names[0])    # Kimario
print(names[3])    #Hamis
print(names[-1])   # Hamis
print(names[len(names)-1)  #Hamis
```

List slicing

Syntax: `list_name[start : end: step]`

```
names=["Kimario","Asha","George", "Hamis"]
print(names[0:3]) #1st 3 elements
```

NOTE: Invalid indexes do not cause slicing expressions to raise an exception.

- If the end index specifies a position beyond the end of the list, Python will use the **length** of the list instead.
- If the start index specifies a position before the beginning of the list, Python will use **0** instead.
- If the start index is greater than the end index, the slicing expression will return an empty list.

Since a list can contain items of different types, a list can also contain a list. Example:

```
lst1=[1,2,3,[4,5]]
print(lst1[3]) # [4, 5]
```

The values in these lists of lists

can be accessed using multiple indexes

```
print(lst1[3][0]) #4 print(lst1[3][1]) #5
```

`Note:` Other languages they use **arrays** which are similar to **lists**, but are much more limited in their capabilities compared to lists.

Looping through a list

- To loop through a list, you can use **for** or **while** loop. **Example**: Print all items in the names list, one by one each on a separate line

```
names=["Kimario","Asha","George", "Hamis"]
for name in names:  #pythonic way
    print(name)
```

OR

```
for index in range(len(names)):
    print(names[index])
```

- To **change** the value of a specific item, refer to the index number and specify a new value.

```
names[2]="Tumaini" # list is changeable
print(names)
```

Example: chp04Aex03Lists03

List methods

https://www.w3schools.com/python/python_ref_list.asp Full list of list methods.

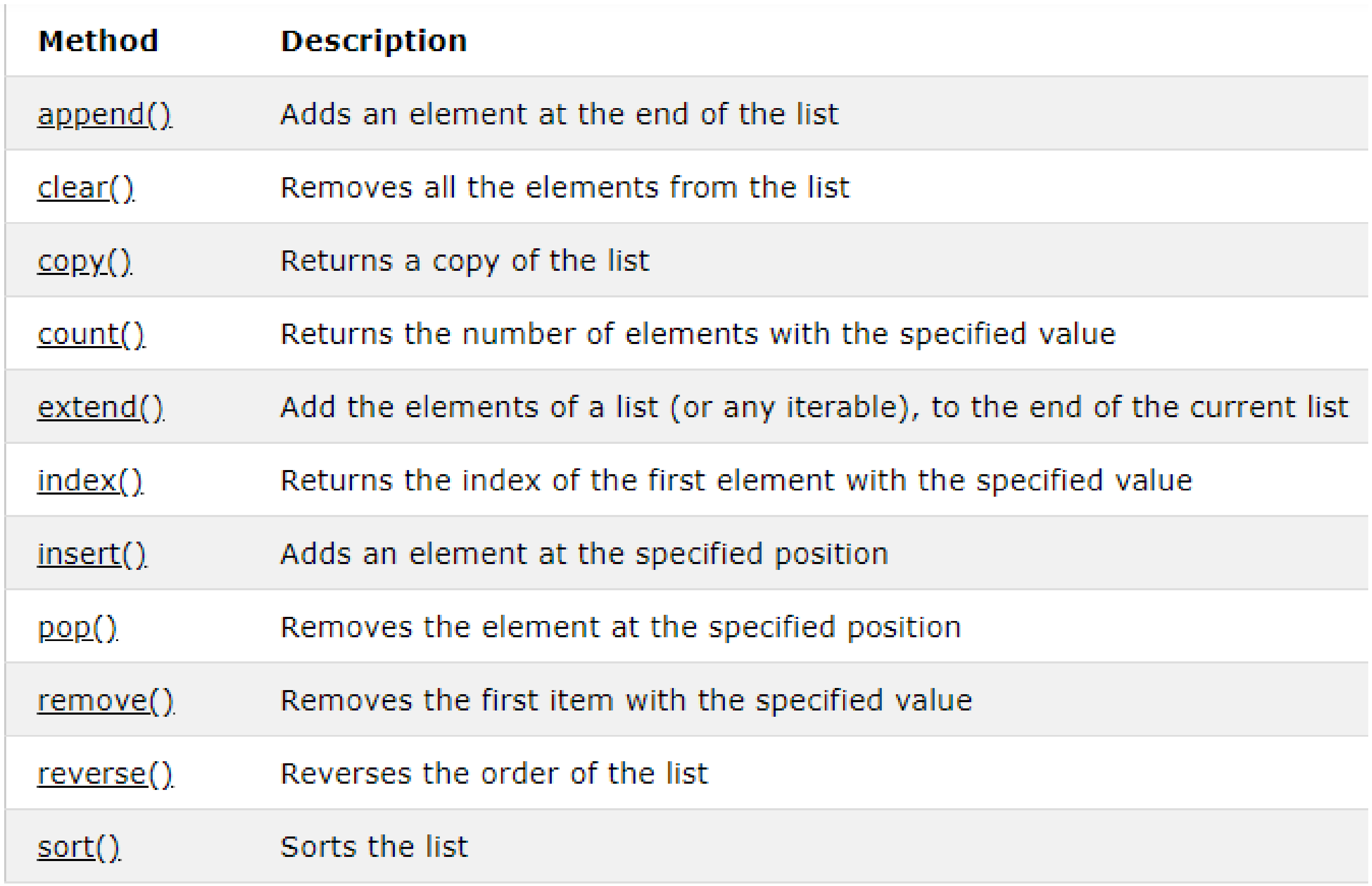
Click on each link of the method to get corresponding examples

Other links:

- https://www.tutorialspoint.com/python3/python_lists.htm Python 3 – Built-in List Functions and Methods
- https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

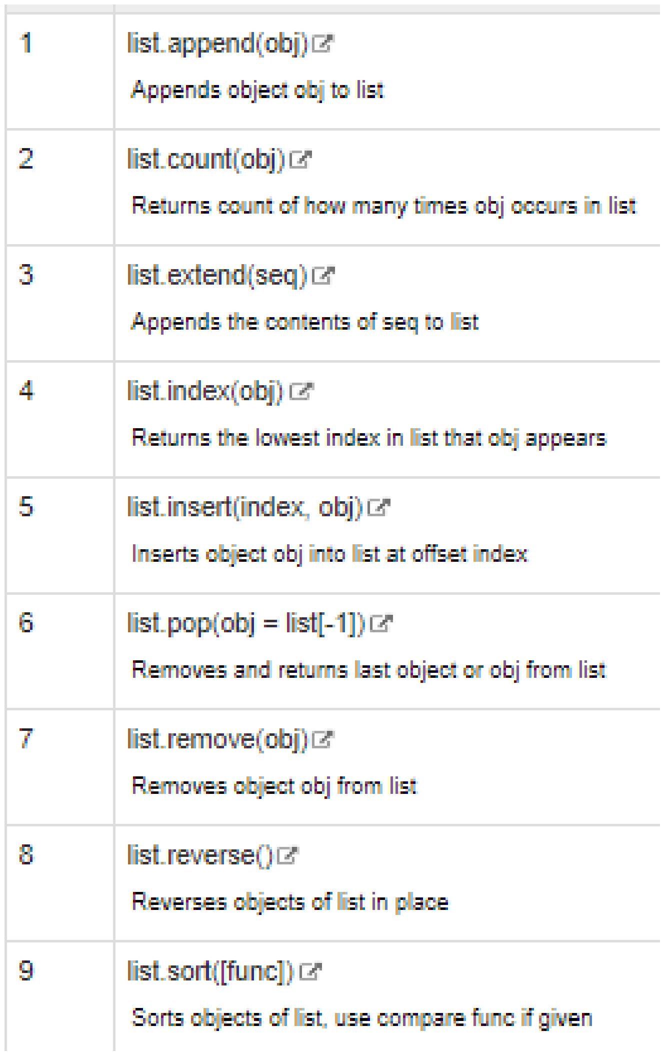
- help("list") – display methods associated with list
- help("list.methodName") – help for a methodName
- In **jupyter notebook**: type listname followed by dot (.) then press TAB to bring up a list of options

https://www.w3schools.com/python/python_ref_list.asp



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

https://www.tutorialspoint.com/python3/python_lists.htm



1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj = list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func if given

Click on the arrow to get details about the method

index method - I

- To get the index of the **first occurrence** of item with the specified value, use index method:

Syntax: `listName.index(x[, start[, end]])`

Return zero-based index in the list of the first item whose value is equal to *x*.

start and ***end*** are optional and are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. **end** is not counted. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

```
names=["Kimario","Asha","George", "Hamis"]
pos=names.index("George") #pos=2
```

Note:ValueError occurs if the item does not exist. First check if the list contains the item you want to search.

index method - II

To check if item exist, use **in** keyword. It is case sensitive

```
#check if specified name exists in the list
```

Example: chp04Aex04indexMethod

```
names=["Kimario","Asha","George", "Hamisi"]
name=input("Enter name to search: ")
if name in names:
    pos=names.index(name)
    print(f"{name} is at index {pos}")
else:
    print(f"{name} does not exist in the 
list")
```

insert method - I

To add item at specific position use **insert** method.

```
Syntax: listName.insert(index,item)
```

chp04Aex05insertMethod

```
names=["Kimario","Asha","George", "Hamisi"]
print("Names before inserting new name")
print(names)
newName=input("Enter name to insert: ")
pos=int(input("Enter position to insert 0-
"+str(len(names))+" "))
names.insert(pos,newName)
print("Names after inserting new name")
print(names)
```

insert method - II

- When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list.
- No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
- The two arguments must be specified. If **`index>=len(listName)`**, will be inserted at the end. If index**`<=-len(listName)`** will be inserted at the beginning.

extend method

- Can extend the list by using **extend** method. The **extend()** method extends the specified list elements at the end of the current list. Example:

chp04Aex06extendMethod

```
names1=["Kimario", "Hamis", "George"]
names2=["Salome", "Zaina"]
names1.extend(names2) # names1=names1+names2
print(f"Extended list is:\n {names1} ")
```

Output:

```
['Kimario', 'Hamis', 'George', 'Salome', 'Zaina']
```

If extending a list with a single item, then enclose item in []

```
names1.extend(["Charles"]) # include []
```

append method-I

The **append** method adds an item to the end of the existing list. append takes only one argument

```
names1=["Kimario", "Hamis", "George"]
names1.append("Zawadi")
print(names1)

#['Kimario', 'Hamis', 'George',
'Zawadi']
```

append method-II

Take note here:

```
names1=["Kimario", "Hamis", "George"]
names1.append(["Zawadi"])
print(names1)
# ['Kimario', 'Hamis', 'George',
['Zawadi']]
print(type(names1[3])) # gives <class
'list'>
names1=["Kimario", "Hamis", "George"]
names1.append(["Zawadi", "Naomi"])
# ['Kimario', 'Hamis', 'George',
['Zawadi', 'Naomi']]
```

append method-III

Use append method to fill a list by specifying the values from the keyboard. **Example**: chp04Aex07appendMethod

```
names=[]  # empty list
numItems=int(input("How many items to add: "))
for index in range(numItems):
    name=input("Enter name "+str(index+1)+": ")
    names.append(name)
    #names.extend([name]) #note use of []
print("\nEntered names are:")
print(names)
```

Sample output on next slide.

append method-IV

Sample output of chp04Aex07appendMethod

```
How many items to add: 3
Enter name 1: Kimario
Enter name 2: Hamis
Enter name 3: Leonard

Entered names are:
['Kimario', 'Hamis', 'Leonard']
```

extend vs append (difference) -EXTRA

```
lst=[1,2,3]
lst.extend([9, 8, 7])
print(lst)  # [1, 2, 3, 9, 8, 7]

lst=[1,2,3]
lst.append([9, 8, 7])
print(lst)  # [1, 2, 3, [9, 8, 7]]
```

Difference:

- extend takes a list as an argument.
- append takes a singleton as an argument.

Deleting items - **remove** method

There are several methods to remove items from a list.
chp04Aex08removeMethod

- The **remove** method removes the **first** matching element (which is passed as an argument) from the list. It is **case sensitive**. Remove method returns None

Syntax: `listname.remove(item)`

- **Note**: error occurs if the specified item to remove does not exist. You can check if the item you want to remove exist before removing.

```
if "George" in names:
    names.remove("George")
    print("George has been removed from the list")
else:
    print("George does not exists in the list")
print(names)
```

pop method

The **pop** method removes and returns the item on a specified index, (or the last item if index is not specified, i.e. index=-1). Error occurs if the list is empty. Popped value can be assigned to a variable.

```
names=["Kimario", "Bernard", "Tumaini",
"Hamisi", "Zawadi"]
```

`names.pop()` – removes the last index (-1)

print(names)

```
=["Kimario", "Bernard", "Tumaini", "Hamisi”]
```

`names.pop(0)` – removes the 1st item

print(names)

```
=["Bernard", "Tumaini", "Hamis"]
```

Note: You can check if the list is not empty by using **len** function. **`if len(listName) !=0:`** ……….

chp04Aex08removeMethod

del statement and clear method

- The **del** statement removes the item(s) at specified index/indexes/indices. Must be within the range.

```
del names[1] # remove item at index=1
del names[1:3] # item 2 and 3
```

- The **del** keyword can also delete the whole list completely:

```
del names   # dangerous, be careful!
print(names) #this will cause an error because this list no longer exists.
```

- The **clear()** method empties the list

```
names.clear()
print(names)   # prints []
```

chp04Aex08removeMethod

reverse method

To reverse the elements in a list use **reverse** method.

```
names=["Kimario","Asha","George", "Hamis"]
names.reverse() # no assignment on left.
That means inplace effect
print(names)
```

Output:

```
['Hamis', 'George', 'Asha', 'Kimario']
```

Example: chp04Aex09reverseMethod

sort method - I

Use **sort** method to sort the list

Syntax: **`sort(reverse=True|False)`**

By default reverse is False which means ascending order. If reverse it is **True** then sort in descending order

Note: You cannot sort a list that contains BOTH string values AND numeric values.

```
names=['Hamis', 'George', 'Asha', 'Kimario']
names.sort()  # in ascending order
print(names)
#['Asha', 'George', 'Hamis', 'Kimario']
names=['Hamis', 'George', 'Asha', 'Kimario']
names.sort(reverse=True) # in descending order
# ['Kimario', 'Hamis', 'George', 'Asha']
```

sort method - II

Example chp04Aex10sortMethod demonstrates use of sort (ascending and descending). **Output is:**

```
Original list:
['Hamis', 'George', 'Asha', 'Kimario']
Sort in ascending order:
['Asha', 'George', 'Hamis', 'Kimario']

Original list:
['Hamis', 'George', 'Asha', 'Kimario']
Sort in descending order:
['Kimario', 'Hamis', 'George', 'Asha']
```

sorted function I

Can use **sorted()** function to assign sorted list to new variable **without** affecting the original list.

Note: You cannot sort a list that contains BOTH string values AND numeric values.

Syntax:

```
newList=sorted(listName, reverse=True|False)
```

Example: chp04Aex10sortedFunction

sorted function II

Example: chp04Aex11sortedFunction

```
names=['Hamis', 'George', 'Asha', 'Kimario']
print(f"Original list: \n {names} ")
lstAscending=sorted(names) #ascending
lstDescending=sorted(names,reverse=True)
print(f"\nList Sorted in ascending order:\n
{lstAscending} ")
print(f"\nList Sorted in descending order:\n
{lstDescending} ")
print(f"\nOriginal list is not affected\n
{names}")
```

Methods which returns None

Most list methods are **void**; they modify the list and return None. Example:

```
t = ['d', 'c', 'e', 'b', 'a']
t = t.sort()
print(t)
```

Results: None

So be careful with assignment of the results for methods which returns None!

copy method

To create a copy of a list use **copy** method.

chp04Aex12CopyMethod

```
names1=["Kimario", "Hamis", "George"]
names2=names1.copy()    #names2=names1[:]
                        #names2=list(names1)
                        #names2=names1 +[]
```

```
names1=["Kimario", "Hamis", "George"]
names2=names1.copy()
print(f"names1 list: {names1}")
print(f"\nnames2 list: {names2}")
```

Output: The two list contains same elements

Caution: Do not use names2=names1. See example in the notebook

min,max and sum methods

The **max** and **min** python built-in methods returns item from the list with max and min value respectively. **sum** returns the sum of elements (numerical ones).

Example: chp04Aex13max_min_sumMethods

```
numbers=[56, 45.5, 65, 12.5, 99.5, 34.5]
print(f"Numbers are: {numbers}")
print(f"sum(numbers): {sum(numbers)}") #313.0
print(f"min(numbers): {min(numbers)}") #12.5
print(f"max(numbers): {max(numbers)}") #99.5
print(f"Average of numbers
{sum(numbers)/len(numbers):.2f}") #52.17
```

Mean and stdev using statistics module

Statistics module provides functions for calculating statistics of data, including average, variance, standard deviation, etc. help('statistics').

```
Example: chp04Aex14mean_stdev_statisticsModule
import statistics
numbers=[56, 45.5, 65, 12.5, 99.5, 34.5]
print(f"Average of numbers
{statistics.mean(numbers):.2f}")
#standard deviation of sample data print(f"Std.
dev of numbers
{statistics.stdev(numbers):.2f}“)
#Average of numbers 52.17
#Std. dev of numbers 29.50
```

count method

The **count** method returns the number of occurrence of an item in a list. If the element does not exist, it returns zero. Example:chp04Aex15countMethod

```
numbers=[1,3,2,5,6,7,2,3,6]
print(f"numbers.count(3)):{numbers.count(3)}")
# 2 occurrences
print(f"numbers.count(6)):{numbers.count(6)}")
# 2 occurrences
print(f"numbers.count(7)):{numbers.count(7)}")
# 1 occurence
print(f"numbers.count(12)):{numbers.count(12)}"
) # 0 occurence
```

enumerate function

`range(len(listName))` – already discussed.

`enumerate(listName)`- enumerate function is used to get a counter and the value from the iterable at the same time. It return two values: the **index** of the item in the list, and the **item** in the list itself. Useful if you need both the item and the item's index in the loop's block

Example: chp04Aex16enumerateFunction

Concatenating lists - I

- Can concatenate lists using + to extend the list

```
list1=[12,34,56]
list2=[32,67]
list3=list1+list2
print(list3)
    #[12, 34, 56, 32, 67]

#can also extend the existing list
list1=list1+list2
print(list1)
#Results: [12, 34, 56, 32, 67]
```

Note: You can concatenate lists **ONLY** with other lists, otherwise you get an error

Concatenating lists - II

```
Example: chp04Aex17concatenateLists
numbers=[] # integer numbers list
while True:
   num=input("Enter number " + str(len(numbers)
+ 1) + " (Or enter nothing to stop.):")
if num == "": break #Sentinel to stop inputs
    numbers += [int(num)] # list concatenation
print("Entered numbers are")
for num in numbers:
    print(num) print(f"List elements are:
{numbers}")
```

Slicing a list -I

Assignment to slices is also possible, and this can even change the size of the list or clear it entirely.

Example:

```
chp04Aex18SliceList
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print(letters)
#['a', 'b', 'c', 'd', 'e', 'f', 'g']
# replace some values
letters[2:5] = ['C', 'D', 'E']
print(letters)
# ['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

Slicing a list -II

```
# now remove added items
letters[2:5] = []
print(letters)    #['a', 'b', 'f', 'g']
# clear the list => empty list
letters[:] = []  #equivalent to: letters=[]
print(letters)  # []
```

* operator

The operator ***** is used to repeat a list as it is in the string.

```
numbers = [1, 2, 3]
print(f"numbers*2: {numbers*2}")

# gives [1, 2, 3 , 1, 2, 3]


numbers=[0]*5
# Result: [0, 0, 0, 0, 0]
```

list unpacking I

Suppose: `lst=[23,45,67]`

List can be unpacked using:

```
x,y,z=lst
```

The values of x,y and z becomes: 23, 45 and 67 respectively.

Note: list unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the list otherwise results into an error.

list unpacking II

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a **list**.

chp04Aex19listUnpacking

```
numbers = [1, 2, 3, 4, 5]
x,y,*z=numbers
print(f"x= {x}")     #1
print(f"y= {y}")     #2
print(f"z= {z}")     #[3,4,5]
```

Implicit Line Joining in the editor

In the editor, expressions in parentheses, square brackets or curly braces can be split over more than one physical line without using backslashes (\). For example:

```
months=["January", "February", "March",
        "April", "May", "June", #comment


               .....]
```

- Implicitly continued lines can carry comments as shown above. The indentation of the continuation lines is not required.

mutable and immutable - I

mutable – that can change

Immutable – that can not change

```
a=1; b=a
a=a+1
print("a=",a) # a=2
print("b=",b) # b=1
#================================
str1="Jumanne";
str2=str1
str1=str1+" George"
print(str1) # str1="Jumanne George"
print(str2) #str2= "Jumanne"
```

mutable and immutable - II

```
c=[1,2,3]; d=c
c.extend([7])
print("c=",c) # c= [1, 2, 3, 7]
print("d=",d) # d= [1, 2, 3, 7]
#===========================
e=[1,2,3]
f=e.copy()  # f=e[:] – colon must be there
e.extend([7])
print("e=",e)   # e= [1, 2, 3, 7]
print("f=",f)   # f= [1, 2, 3]
```

Refer chp04Aex20Mutability for more details and how to use id() built-in function to return identity of an object.

Classes and mutability

Class	**Description**	**Mutable?**
bool	Boolean value	NO
int	Integer	NO
float	Floating-point number	NO
str	String	NO
tuple	Tuple	NO
List	List	YES
dict	Dictionary	YES
set	Set	YES

string – split method - I

str.split() function splits a string into a list. Then can use list methods to process splitted string. Example: chp04Aex21strSplitFunction

```
date1 = "20/11/2021"
splittedDate=date1.split("/")
print(type(splittedDate)) #<class 'list'>
print("splittedDate:",splittedDate)
print(type(splittedDate))
print("Day is: ",splittedDate[0])
print("Month is: ",splittedDate[1])
print("Year is :",splittedDate[2])
```

string – split method - II

Enter values as string and then use split to create a list. Example:

```
valuesStr=input('Enter values: ') print(f"valuesStr:
{valuesStr}") valuesList=valuesStr.split()
print(f"valuesList: {valuesList}") valuesListNums=[]
# numerical values
for i in range(len(valuesList)):
  valuesListNums.append(int(valuesList[i]))
print(valuesListNums)
```

chp04Aex21strSplitFunction

Output:

```
Enter values: 34 56 78 98 43
valuesStr: 34 56 78 98 43
valuesList: ['34', '56', '78', '98', '43']
valuesListNums: [34, 56, 78, 98, 43]
```

String - join method

The **join()** method takes all items in an iterable and joins them into one string. A string must be specified as the separator.

chp04Aex22strJoinFunction

```
date1 = "20/11/2021"
splittedDate=date1.split("/")
print("type of splittedDate is: ",
type(splittedDate))
date2="/".join(splittedDate)
print("type of date2 is: ", type(date2))
print("date1 is ", date1)
print("date2 is ", date2)
```

keywords in compact form
Example: chp04Aex23Keywords

```
import keyword,platform # platform used to
determine python version
kwords=keyword.kwlist
print(f"type(kwords): {type(kwords)} \n")
print(f"keywords in Python
{platform.python_version()} are:\n {kwords}",)
print(f"\nThere are {len(kwords)} keywords")
```

Compare results with those obtained using
`help("keywords")`

Two dimensional list -I

Lists of lists are also known as nested lists, or two-dimensional lists. Two-dimensional list has other lists as its elements.

In Mathematics, a 2D 3x3 matrix is defines as:

$$A = \begin{bmatrix} 12 & 45.5 & 23.5 \\ 56 & 98.5 & 90 \\ 45 & 56.5 & 34 \end{bmatrix}$$

Two dimensional list -II

List of 4 regions in Tanzania with their capitals:

```
regions=[ ['Ruvuma','Songea'],
['Kilimanjaro','Moshi'], ['Mara','Musoma'],
['Mwanza','Mwanza']]
```

	Column 0	**Column 1**
Row 0	'Ruvuma'	'Songea'
Row 1	'Kilimanjaro'	'Moshi'
Row 2	'Mara'	'Musoma'
Row 4	'Mwanza'	'Mwanza'

Two dimensional list -III

	Column 0	**Column 1**
Row 0	Regions[0][0]	Regions[0][1]
Row 1	Regions[1][0]	Regions[1][1]
Row 2	Regions[2][0]	Regions[2][1]
Row 4	Regions[3][0]	Regions[3][1]

Refer to examples: chp04Aex24_2DLists01, chp04Aex25_2DLists02 and chp04Aex26_2DLists03 on how to access 2D lists data.
H/W: Think also about processing marks of students in a number of courses.

Advanced – Not in the syllabus

If interested, study how to use list comprehension.
Prepare your own lecture notes if interested.
This is advanced stuff BUT knowing how to use it can be an extra advantage.

- https://docs.python.org/3/tutorial/datastructures.html#nested-list-comprehensions
- https://www.w3schools.com/python/python_lists_comprehension.asp
- Learning Python, 5th (mark Lutz Oreilly 2013) – chapter 4 - Introducing Python Object Types
- Google `list comprehension` including youtube videos

==End of chapter 4A ==

Next: Chapter 4B Tuples