SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name – Computational Thinking and Python Programming

I Year / I Semester

Unit 4-LISTS, TUPLES, DICTIONARIES

Lists

- List is an ordered sequence of items. Values in the list are called elements / items.
- It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- Items in the lists can be of different data types.

Operations on list:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Updating
6. Membership
7. Comparison

operations	examples	description

create a list	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> print(a)
	[2, 3, 4, 5, 6, 7, 8, 9, 10]
		in this way we can create a list at compile time

Indexing	>>> print(a[0])
	2
	>>> print(a[8])
	10
	>>> print(a[-1])
	10
		Accessing the item in the position 0
		Accessing the item in the position 8
		Accessing a last element using negative indexing.

Slicing	>>> print(a[0:3])
	[2, 3, 4]
	>>> print(a[0:])
	[2, 3, 4, 5, 6, 7, 8, 9, 10]
		Printing a part of the list.

Concatenation	>>>b=[20,30] >>> print(a+b) [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30]	Adding and printing the items of two lists.
Repetition	>>> print(b*3) [20, 30, 20, 30, 20, 30]	Create a multiple copies of the same list.

Updating	>>> print(a[2])
	4
	>>> a[2]=100
	>>> print(a)
	[2, 3, 100, 5, 6, 7, 8, 9, 10]
		Updating the list using index value.

Membership	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in list. Otherwise returns false.

Comparison	>>> a=[2,3,4,5,6,7,8,9,10]
	>>>b=[2,3,4]
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

List slices:

List slicing is an operation that extracts a subset of elements from an list and packages them as another list.

Syntax:
Listname[start:stop]
Listname[start:stop:steps]

- v default start value is 0
- v default stop value is n-1
- v [:] this will print the entire list
- v [2:2] this will create a empty slice

slices	example	description

a[0:3]	>>> a=[9,8,7,6,5,4]
	>>> a[0:3]
	[9, 8, 7]
		Printing a part of a list from 0 to 2.

a[:4]	>>> a[:4] [9, 8, 7, 6]	Default start value is 0. so prints from 0 to 3
a[1:]	>>> a[1:] [8, 7, 6, 5, 4]	default stop value will be n-1. so prints from 1 to 5
a[:]	>>> a[:] [9, 8, 7, 6, 5, 4]	Prints the entire list.
a[2:2]	>>> a[2:2] [ ]	print an empty slice
a[0:6:2]	>>> a[0:6:2] [9, 7, 5]	Slicing list values with step size 2.
a[::-1]	>>> a[::-1] [4, 5, 6, 7, 8, 9]	Returns reverse of given list values

List methods:

-

v Methods used in lists are used to manipulate the data quickly.
v These methods work only on lists.
v They do not work on the other sequence types that are not mutable, that is, the values they contain cannot be changed, added, or deleted.

syntax:
list name.method name(element/index/list)

	syntax	example	description

1	a.append(element)	>>> a=[1,2,3,4,5]
		>>> a.append(6)
		>>> print(a)
		[1, 2, 3, 4, 5, 6]
			Add an element to the end of the list

2	a.insert(index,element)	>>> a.insert(0,0)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6]
			Insert an item at the defined index

3	a.extend(b)	>>> b=[7,8,9]
		>>> a.extend(b)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8,9]
			Add all elements of a list to the another list

4	a.index(element)	>>> a.index(8) 8	Returns the index of the first matched item

5	a.sort()	>>> a.sort()
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8]
			Sort items in a list in ascending order

6	a.reverse()	>>> a.reverse()
		>>> print(a)
		[8, 7, 6, 5, 4, 3, 2, 1, 0]
			Reverse the order of items in the list

7	a.pop()	>>> a.pop() 0	Removes and returns an element at the last element
8	a.pop(index)	>>> a.pop(0) 8	Remove the particular element and return it.

9	a.remove(element)	>>> a.remove(1)
		>>> print(a)
		[7, 6, 5, 4, 3, 2]
			Removes an item from the list

10	a.count(element)	>>> a.count(6) 1	Returns the count of number of items passed as an argument

11	a.copy()	>>> b=a.copy()
		>>> print(b)
		[7, 6, 5, 4, 3, 2]
			Returns a shallow copy of the list

12	len(list)	>>> len(a) 6	return the length of the length
13	min(list)	>>> min(a) 2	return the minimum element in a list
14	max(list)	>>> max(a) 7	return the maximum element in a list.

15	a.clear()	>>> a.clear()
		>>> print(a)
		[ ]
			Removes all items from the list.

16	del(a)	>>> del(a)
		>>> print(a)
		Error: name 'a' is not defined
			delete the entire list.

List loops:

1. **For loop**
2. **While loop**
3. **Infinite loop**

1. List using For Loop:

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

Iterating over a sequence is called traversal.

Loop continues until we reach the last item in the sequence.

The body of for loop is separated from the rest of the code **using indentation**.

Syntax:

for val in sequence:

Accessing element	output
a=[10,20,30,40,50] for i in a: print(i)	1 2 3 4 5
Accessing index	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(i)	0 1 2 3 4
Accessing element using range:	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(a[i])	10 20 30 40 50

2. List using While loop

v The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

v When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Syntax:
while (condition):
body of while

Sum of elements in list

```
a=[1,2,3,4,5]
i=0
sum=0
while i<len(a):
sum=sum+a[i]
i=i+1
print(sum)
```

Output:

15

3. Infinite Loop

A loop becomes infinite loop if the condition given never becomes false. It keeps on running. Such loops are called infinite loop.

Example

```
a=1
while (a==1):
n=int(input("enter the number"))
print("you entered:" , n)
```

Output:

```
Enter the number 10
you entered:10
Enter the number 12
you entered:12
Enter the number 16
you entered:16
```

Mutability:

v Lists are mutable. (can be changed)

v Mutability is the ability for certain types of data to be changed without entirely recreating it.

v An item can be changed in a list by accessing it directly as part of the assignment statement.

v Using the indexing operator (square brackets[ ]) on the left side of an assignment, one of the list items can be updated.

Example	description

>>> a=[1,2,3,4,5]
>>> a[0]=100
>>> print(a)
[100, 2, 3, 4, 5]
	changing single element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[100,100,100]
>>> print(a)
[100, 100, 100, 4, 5]
	changing multiple element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[ ]
>>> print(a)
[4, 5]
	The elements from a list can also be removed by assigning the empty list to them.

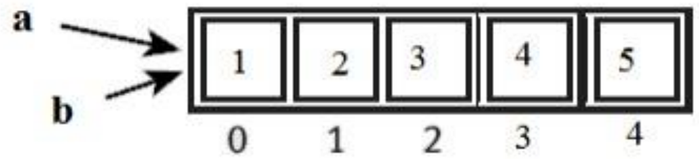
>>> a=[1,2,3,4,5]
>>> a[0:0]=[20,30,45]
>>> print(a)
[20,30,45,1, 2, 3, 4, 5]
	The elements can be inserted into a list by squeezing them into an empty slice at the desired location.

Aliasing(copying):

v Creating a copy of a list is called aliasing. When you create a copy both list will be having same memory location. changes in one list will affect another list.

v **Alaising refers to having different names for same list values.**

Example	Output:
a= [1, 2, 3 ,4 ,5] b=a print (b) a is b a[0]=100 print(a) print(b)	 [1, 2, 3, 4, 5] True [100,2,3,4,5] [100,2,3,4,5]



- In this a single list object is created and modified using the subscript operator.
- When the first element of the list named "**a**" is replaced, the first element of the list named "**b**" is also replaced.

v This type of change is what is known as a **side effect**. This happens because after the assignment **b=a**, the variables **a** and**b** refer to the exact same list object. v They are **aliases** for the same object. This phenomenon is known as **aliasing**.

v To prevent aliasing, a new object can be created and the contents of the original can be copied which is called **cloning.**

-

Clonning:

v To avoid the disadvantages of copying we are using cloning. creating a copy of a same list of elements with two different memory locations is called cloning.

v Changes in one list will not affect locations of aother list.

v Cloning is a process of making a copy of the list without modifying the original list.

1. Slicing
2. list()method
3. copy() method

clonning using Slicing

```
>>>a=[1,2,3,4,5]
>>>b=a[:]
>>>print(b)
[1,2,3,4,5]
>>>a is b
False
```

clonning using List( ) method

```
>>>a=[1,2,3,4,5]
>>>b=list
>>>print(b)
[1,2,3,4,5]
>>>a is b
false
>>>a[0]=100
>>>print(a)
>>>a=[100,2,3,4,5]
>>>print(b)
>>>b=[1,2,3,4,5]
```

clonning using copy() method

```
a=[1,2,3,4,5]
>>>b=a.copy()
>>>  print(b) [1, 2, 3, 4, 5]
>>>  a is b
False
```

List as parameters:

v In python, arguments are passed by reference.

v If any changes are done in the parameter which refers within the function, then the changes also reflects back in the calling function.

v When a list to a function is passed, the function gets a reference to the list.

v Passing a list as an argument actually passes a reference to the list, not a copy of the list.

Since lists are mutable, changes made to the elements referenced by the parameter change the same list that the argument is referencing.

Example 1:

```
def remove(a):
a.remove(1)
a=[1,2,3,4,5]
remove(a)
print(a)
```

Output

[2,3,4,5]

Example 2:

```
def inside(a):
for i in range(0,len(a),1):
a[i]=a[i]+10
print(“inside”,a)
a=[1,2,3,4,5]
inside(a)
print(“outside”,a)
```

Output

```
inside [11, 12, 13, 14, 15]
outside [11, 12, 13, 14, 15]
```

Example 3

```
def insert(a):
a.insert(0,30)
a=[1,2,3,4,5]
insert(a)
print(a)
```

output

```
[30, 1, 2, 3, 4, 5]
```

Tuple:

- A tuple is same as list, except that the set of elements is enclosed in parentheses instead of square brackets.
- A tuple is an immutable list. i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- But tuple can be converted into list and list can be converted in to tuple.

methods	example	description

list( )	>>> a=(1,2,3,4,5)
	>>> a=list(a)
	>>> print(a)
	[1, 2, 3, 4, 5]
		it convert the given tuple into list.

tuple( )	>>> a=[1,2,3,4,5]
	>>> a=tuple(a)
	>>> print(a)
	(1, 2, 3, 4, 5)
		it convert the given list into tuple.

Benefit of Tuple:

- Tuples are faster than lists.
- If the user wants to protect the data from accidental changes, tuple can be used.
- Tuples can be used as keys in dictionaries, while lists can't.

Operations on Tuples:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Membership
6. Comparison

Operations	examples	description
Creating a tuple	>>>a=(20,40,60,"apple","ball")	Creating the tuple with elements of different data types.
Indexing	>>>print(a[0]) 20 >>> a[2] 60	Accessing the item in the position 0 Accessing the item in the position 2
Slicing	>>>print(a[1:3]) (40,60)	Displaying items from 1st till 2nd.
Concatenation	>>> b=(2,4) >>>print(a+b) >>>(20,40,60,"apple","ball",2,4)	Adding tuple elements at the end of another tuple elements
Repetition	>>>print(b*2) >>>(2,4,2,4)	repeating the tuple in n no of times

Membership	>>> a=(2,3,4,5,6,7,8,9,10)
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in tuple. Otherwise returns false.

Comparison	>>> a=(2,3,4,5,6,7,8,9,10)
	>>>b=(2,3,4)
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

Tuple methods:

Tuple is immutable so changes cannot be done on the elements of a tuple once it is assigned.

methods	example	description

a.index(tuple)	>>> a=(1,2,3,4,5)
	>>> a.index(5)
	4
		Returns the index of the first matched item.

a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple
min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

Tuple Assignment:

- Tuple assignment allows, variables on the left of an assignment operator and values of tuple on the right of the assignment operator.
- Multiple assignment works by creating a tuple of expressions from the right hand side, and a tuple of targets from the left, and then matching each expression to a target.

- Because multiple assignments use tuples to work, it is often termed tuple assignment.

Uses of Tuple assignment:

- It is often useful to swap the values of two variables.

Example:

Swapping using temporary variable:

```
a=20
b=50
temp = a
a = b
b = temp
print("value after swapping is",a,b)
```

Swapping using tuple assignment:

```
a=20
b=50
(a,b)=(b,a)
print("value after swapping is",a,b)
```

Multiple assignments:

Multiple values can be assigned to multiple variables using tuple assignment.

```
>>>(a,b,c)=(1,2,3)
>>>print(a)
1
>>>print(b)
2
>>>print(c)
3
```

Tuple as return value:

- A Tuple is a comma separated sequence of items.
- It is created with or without ( ).
- A function can return one value. if you want to return more than one value from a function. we can use tuple as return value.

Example1:

```
def div(a,b):
r=a%b
q=a//b
return(r,q)
a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
r,q=div(a,b)
print("reminder:",r)
print("quotient:",q)
```

Output:

```
enter a value:4
enter b value:3
reminder: 1
quotient: 1
```

Tuple as argument:

The parameter name that begins with * gathers argument into a tuple.

Example:

```
def printall(*args):
print(args)
printall(2,3,'a')
```

Output:

```
(2, 3, 'a')
```