TUPLE

23.01.2023

INTRODUCTION

- 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.

BENEFITS 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

- Indexing
- Slicing
- Concatenation
- Repetitions
- Membership
- 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.

Operations	Examples	Description

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.

Operations	Examples	Description

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.

Operations	Examples	Description
a.index(tuple)	>>> a=(1,2,3,4,5) >>> a.index(5) Ans: 4	Returns the index of the first matched item.
a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) Ans: 1	Returns the count of the given element.
len(tuple)	>>> len(a) Ans: 5	return the length of the tuple

CONTD…

Operations	Examples	Description
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.

Swapping using temporary variable	Swapping using tuple assignment

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

Example:

```
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
```

Example:	Output

`def min_max(a):`
`    small=min(a)`
`    big=max(a)`
`    return(small,big)`
`a=[1,2,3,4,6]`
`small,big=min_max(a)`
`print("smallest:",small)`
`print("biggest:",big)`
	`smallest: 1`
	`biggest: 6`

TUPLE AS ARGUMENT

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

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

DICTIONARIES

INTRODUCTION

- Dictionary is an unordered collection of elements. An element in dictionary has a key: value pair.
- All elements in dictionary are placed inside the curly braces i.e. { }
- Elements in Dictionaries are accessed via keys and not by their position.
- The values of a dictionary can be any data type.
- Keys must be immutable data type (numbers, strings, tuple)

OPERATIONS ON DICTIONARY

- Accessing an element
- Update
- Add element
- Membership

Operations	Examples	Description

Creating a dictionary	>>> a={1:"one",2:"two"}
	>>> print(a)
	{1: 'one', 2: 'two'}
		Creating the dictionary with elements of different data types

accessing an element	>>> a[1]
	'one'
	>>> a[0]
	KeyError: 0
		Accessing the elements by using keys.

Update	>>> a[1]="ONE"
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Assigning a new value to key. It replaces the old value by new value.

Operations	Examples	Description

add element	>>> a[3]="three"
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
		Add new element in to the dictionary with key.

membership	a={1: 'ONE', 2: 'two', 3: 'three'}
	>>> 1 in a
	True
	>>> 3 not in a
	False
		Returns True if the key is present in dictionary. Otherwise returns false.

METHODS IN DICTIONARY

Method	Examples	Description

a.copy( )	a={1: 'ONE', 2: 'two', 3: 'three’}
	>>> b=a.copy()
	>>> print(b)
	{1: 'ONE', 2: 'two', 3: 'three'}
		It returns copy of the dictionary. here copy of dictionary ’a’ get stored in to dictionary ‘b’

a.items()	>>> a.items() dict_items([(1, 'ONE'), (2, 'two'), (3, 'three')])	Return a new view of the dictionary's items. It displays a list of dictionary’s (key, value) tuple pairs.

Method	Examples	Description
a.keys()	>>> a.keys() dict_keys([1, 2, 3])	It displays list of keys in a dictionary
a.values()	>>> a.values() dict_values(['ONE', 'two', 'three'])	It displays list of values in dictionary

a.pop(key)	>>> a.pop(3)
	'three’
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Remove the element with key and return its value from the dictionary.

Method	Examples	Description

setdefault(key,value)	>>> a.setdefault(3,"three")
	'three’
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three’}
	>>> a.setdefault(2)
	'two'
		If key is in the dictionary, return its value. If key is not present, insert key with a value of dictionary and return dictionary.

a.update(dictionary)	>>> b={4:"four"}
	>>> a.update(b)
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three', 4: 'four'}
		It will add the dictionary with the existing dictionary

fromkeys()	>>> key={"apple","ball"}
	>>> value="for kids"
	>>> d=dict.fromkeys(key,value)
	>>> print(d)
	{'apple': 'for kids', 'ball': 'for kids'}
		It creates a dictionary from key and values.

Method	Examples	Description

len(a)	a={1: 'ONE', 2: 'two', 3: 'three’}
	>>>len(a)
	3
		It returns the length of the list.

clear()	a={1: 'ONE', 2: 'two', 3: 'three’}
	>>>a.clear()
	>>>print(a)
	>>>{ }
		Remove all elements form the dictionary.

del(a)	a={1: 'ONE', 2: 'two', 3: 'three’} >>> del(a)	It will delete the entire dictionary.

DIFFERENCE BETWEEN LIST, TUPLES AND DICTIONARY

List	Tuple	Dictionary
A list is mutable	A tuple is immutable	A dictionary is mutable
Lists are dynamic	Tuples are fixed size in nature	In values can be of any data type and can repeat, keys must be of immutable type
List are enclosed in brackets[ ] and their elements and size can be changed	Tuples are enclosed in parenthesis ( ) and cannot be updated	Dictionary are enclosed in curly braces { } and consist of key:value
Homogenous	Heterogeneous	Homogenous

List	Tuple	Dictionary

Example:
List = [10, 12, 15]
	Example:
	Words = ("spam", "egss")
	Or
	Words = "spam", "eggs"
		Example:
		Dict = {"ram": 26, "abi": 24}

Access: print(list[0])	Access: print(words[0])	Access: print(dict["ram"])
Can contain duplicate elements	Can contain duplicate elements. Faster compared to lists	Cant contain duplicate keys, but can contain duplicate values
Slicing can be done	Slicing can be done	Slicing can't be done

List	Tuple	Dictionary
Usage: o List is used if a collection of data that doesn't need random access. o List is used when data can be modified frequently	Usage: o Tuple can be used when data cannot be changed. o A tuple is used in combination with a dictionary i.e.a tuple might represent a key.	Usage: o Dictionary is used when a logical association between key:value pair. o When in need of fast lookup for data, based on a custom key. o Dictionary is used when data is being constantly modified.

Array Techniques

11.01.2023

Unit V

Introduction – Array Order Reversal – Array Counting – Finding the Maximum number in a set – Removal of Duplicates from an ordered array – Partitioning an array – Finding the Kth smallest element

Python Lists: list operations – Tuples – Sets Operations – Dictionaries – Time Tradeoff.

List

- List is a versatile data type available in python.
- Like a string, a list is a sequence of values.
- In a string, the values are characters; in a list, they can be any type. The values in a list are called elements or sometimes items.
- There are several ways to create a new list; the simplest is to enclose the elements in square brackets ([and]):
- It can have any number of items and they may be of different types (integer, float, string etc.).

Syntax

List_Variable = [val1, val2, ...]

- [ ]   # empty list
- [1, 2, 3]   # list of integers
- ['physics', 'chemistry',‘computer‘] # list of strings
- [1, "Hello", 3.4]   # list with mixed datatypes

A list can even have another list as an item.
This is called nested list.

○ my_list = ["mouse", [8, 4, 6], ['a’]]   # nested list

Practice using Python

```
>>>subject= ['physics', 'chemistry','computer’]
>>>mark=[98,87,94]
>>>empty=[ ]
>>>print(Subject,mark,empty)
['physics', 'chemistry', 'computer'], [98, 87, 94], []
```

Lists are Mutable

- The syntax for accessing the elements of a list is the same as for accessing the characters of a string—the bracket operator.
- The expression inside the brackets specifies the index.
- The indices start at 0:

```
>>>subject[0]
'physics'
```

Practice using Python

Unlike strings, lists are mutable

```
>>>mark=[98,87,94]
>>>mark[2]=100
>>>mark
[98, 87, 100]
```

State diagrams

subject

$$\begin{tikzcd}
0 \arrow[r] & \text{'physics'} \\
1 \arrow[r] & \text{'chemistry'} \\
2 \arrow[r] & \text{'computer'}
\end{tikzcd}$$

Mark

$$\begin{tikzcd}
0 \arrow[r] & 98 \\
1 \arrow[r] & 87 \\
2 \arrow[r, dashed] \arrow[dr] & \cancel{94} \\
& 100
\end{tikzcd}$$

Empty $\boxed{}$

Access Values in list

Syntax

```
Seq= List [start:stop:step]
```

List indices work the same way as string indices

- Any integer expression can be used as an index.
- If you try to read or write an element that does not exist, you get an IndexError.
- If an index has a negative value, it counts backward from the end of the list.

Example

- Seq = List[: : 2]   # get every other element, starting with index 0
- Seq = List[1 : : 2]   # get every other element, starting with index 1

```
num_list = [1,2,3,4,5,6,7,8,9,10]
print("num_list is:", num_list)
num_list is: [1,2,3,4,5,6,7,8,9,10]
print ("First element in the list is ", num_list[0])
First element in the list is 1
```

Contd…

```
print ("num_list [2:5] = " , num_list [2:5] )
num_list [2:5] = [3, 4, 5]
print ("num_list [: : 2] = " , num_list [: : 2] )
num_list [: : 2] = [1, 3, 5, 7, 9]
print ("num_list [1 : : 3] = " , num_list [1 : : 3] )
num_list [1 : : 3] = [2, 5, 8]
```

The in operator also works on lists.

```
>>>subject= ['physics', 'chemistry', 'computer’]
>>> 'chemistry' in subject
True
>>> 'english' in subject
False
```

Traversing a List, List Loop

- The most common way to traverse the elements of a list is with a for loop.

Syntax

```
for VARIABLE in LIST:
        STATEMENT(S) USING VARIABLE
```

- Elements in the List are stores in VARIABLE one by one for each iteration.
- Use the element stored in VARIABLE for further processing inside the loop body

Example

```
>>>for s in subject:
            print(s)
physics
chemistry
computer
```

Contd…

- A common way to do that is to combine the built-in functions range and len:
- **For ITERATION_VARIABLE in range(len(LIST_NAME)):**

  STATEMENT(S) Using LIST_NAME[ITERATION_VARIABLE]

  ITERATION_VARIABLE holds value from o to length of LIST_NAME one by one. Using that ITERATION_VARIABLE. we can access individual element of LIST_NAME for reading and writing

Example

```
>>>for i in range(len(mark)):
mark[i] = mark[i] * 2
>>>mark
[196, 174, 200]
```

Contd...

- This loop traverses the list and updates each element. len returns the number of elements in the list.
- range returns a list of indices from 0 to n-1, where n is the length of the list. Each time through the loop i gets the index of the next element.
- The assignment statement in the body uses i to read the old value of the element and to assign the new value.

Contd…

- A for loop over an empty list never runs the body:

```
for x in [ ]:
print('This never happens.’)
```

- Although a list can contain another list, the nested list still counts as a single element.
- The length of the following list is 3:

Example

```
>>>my_list = ["mouse", [8, 4, 6], ['a’]]
>>>len(my_list)
3
```

List Operations

The **+** operator concatenates lists:

```
>>>first=[100,200,300]
>>>second=[55,65]
>>>third=first + second
>>>third
[100, 200, 300, 55, 65]
```

List Operations

The * operator repeats a list a given number of times:

```
>>> [5]*3
[5, 5, 5]
>>> [55,65]*3
[55, 65, 55, 65, 55, 65]
```

List Slices

```
>>> w=['w','e','l','c','o','m','e’]
>>>w[2:5]
['l', 'c', 'o’]
>>>w[:3 ]
['w', 'e', 'l’]
>>>w[5:]
['m', 'e’]
>>>w[:]
['w', 'e', 'l', 'c', 'o', 'm', 'e']
```

- If you omit the first index, the slice starts at the beginning.
- If you omit the second, the slice goes to the end.
- So if you omit both, the slice is a copy of the whole list.

List Methods

- **list.append(obj) :** Appends object obj to list
- **list.count(obj) :** Returns count of how many times obj occurs in list
- **list.extend(seq):** Appends the contents of sequence to list
- **list.index(obj):** Returns the lowest index in list that obj appears
- **list.insert(index, obj):** Inserts object obj into list at offset index
- **list.pop(obj=list[-1]):** Removes and returns last object or obj from list

Contd…

- **list.remove(obj):** Removes object obj from list
- **list.reverse():** Reverses objects of list in place
- **list.sort([func]):** Sorts objects of list, use compare func if given
- **list.pop(obj):** pop modifies the list and returns the element that was removed.
- **list.remove(obj):** If you know the element you want to remove (but not the index), you can use remove:
- **list.append(obj):** The append method add the object at the end of a list.

list.append(obj)

```
>>> t = ['a', 'b', 'c’]
 >>>t.append('d’)
>>> t
['a', 'b', 'c', 'd']
```

list.count(obj)

```
>>>aList = [123, 'xyz', 'zara', 'abc', 123]
>>>aList.count(123)
2
>>>aList.count('xyz’)
1
```

list.extend(seq)

```
>>> t1 = ['a', 'b', 'c’]
>>> t2 = ['d', 'e’]
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

list.index(obj):

```
>>>aList = [123, 'xyz', 'zara', 'abc', 123]
>>>aList.index(123)
0
>>>aList.index('xyz’)
1
```

list.remove(obj):

```
>>>aList = [123, 'xyz', 'zara', 'abc’]
>>>aList.remove('xyz’)
>>>aList
[123, 'zara', 'abc']
```

list.pop(obj=list[-1]):

```
>>>aList = [123, 'xyz', 'zara', 'abc']
>>>aList.pop()
'abc’
>>>aList.pop(2)
'zara’
>>>aList
[123, 'xyz']
```

list.reverse():

```
>>>aList = [123, 'xyz', 'zara', 'abc', 'xyz']
>>>aList.reverse()
>>>aList
['xyz', 'abc', 'zara', 'xyz', 123]
```

list.sort([func]):

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

list.pop(obj):

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>>t
['a', 'c']
 >>>x
'b'
```

list.remove(obj)

```
>>> t = ['a', 'b', 'c']
>>>t.remove('b')
>>>t
['a', 'c']
```

list.append(obj):

```
>>> t1 = [10, 20]
>>> t2 = t1.append(30)
>>> t1
[10, 20, 30]
>>> t2
None
```

list.insert(index, obj):

```
>>>aList.insert( 3, 2009)
>>>aList [123, 'xyz', 'zara', 2009, 'abc', 123]
```

Deleting Elements

- There are several ways to delete elements from a list.
- If you know the index of the element you want to delete, you can use **pop:**

```
>>> t = ['a', 'b', 'c’]
>>> x = t.pop(1)
>>>t
['a', 'c’]
>>>x
'b'
```

Contd…

- pop modifies the list and returns the element that was removed.
- If you don‘t provide an index, it deletes and returns the last element.
- If you don‘t need the removed value, you can use the del operator:

```
>>> t = ['a', 'b', 'c’]
>>>del t[1]
>>>t
['a', 'c']
```

Contd…

- If you know the element you want to remove (but not the index), you can use remove:

```
>>> t = ['a', 'b', 'c’]
>>>t.remove('b’)
 >>>t
['a', 'c’]
```

Contd…

- To remove more than one element, you can use del with a slice index:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f’]
>>>del t[1:5]
>>>t
['a', 'f']
```

Aliasing

- If one is a refers to an object and you assign two = one, then both variables refer to the same object:

```
>>>one = [10, 20, 30]
>>>two = one
>>>two is one
True
>>>two[1]=40
>>>one
[10, 40, 30]
```

Cloning Lists

- If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference.
- This process is sometimes called cloning, to avoid the ambiguity of the word copy.
- The easiest way to clone a list is to use the slice operator.
- Taking any slice of a creates a new list. In this case the slice happens to consist of the whole list.

Example

```
a = [81, 82, 83] b = a[:] # make a clone using slice
print(a == b)
print(a is b)
b[0] = 5
print(a)
print(b)
```

- Output will be

```
True
False
[81, 82, 83]
[5, 82, 83]
```

But the problem is if the list is a nested one this method won‘t work

Example

```
a = [81, 82, [83,84]]
b = a[:]            # make a clone using slice
print(a == b)
print(a is b)
b[2][0] = 5
print(a)
print(b)
```

- Output will be

```
True
False
[81, 82, [5, 84]]
[81, 82, [5, 84]]
```

Change in list b affect list a. In order to overcome this issue python provide a module called copy.

Tutorial - 3

11.01.2023

Problems

- Write a program that creates a list of numbers from 1 to 20 that are either divisible by 2 or divisible by 4 without using the filter function.
- Write a program that defines a list of countries that are a member of BRICS(Brazil, Russia, India, China, Srilanka). Check whether a country is a member of BRICS or not.

Problems

- Write a program that creates a list of 10 random integers. Then create two lists – odd list and even list that has all odd and even values in the list respectively.

(Hint: random , append)

- Write a program to add two matrices (using nested lists).