Tuples

- Tuples are ordered sequence that are written as a comma-separated elements within round brackets..
- Tuples once created can’t be modified that makes them faster as compared to list.
- A tuple is a collection which is **ordered and unchangeable**.

- Example:

```
#Here we are declaring a tuple named score:
Score = (20,30,55,68,12,32)

#All the different types in Python - String, Integer, Float can all be contained in tuple.
Tuple1 = (‘person’, 20, 12.5)
```

Tuples

Each element of tuple can be accessed via index similar to lists.

```
>>>tup=('rakesh',23,6.2,False)
>>>print(tup[0])
>>>print(tup[-1])
```

Tuple has immutable nature, i.e., tuple can’t be changed or modified after its creation.

```
>>>tup_num = (1,2,3,4)
>>>tup_num[2] = 5
```

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

Tuples allow duplicate values:

- thistuple = ("apple", "banana", "cherry", "apple", "cherry")
  print(thistuple)

('apple', 'banana', 'cherry', 'apple', 'cherry')

Print the number of items in the tuple:

- thistuple = ("apple", "banana", "cherry")
  print(len(thistuple))

3

Create Tuple With One Item

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) **Output**1 2 3 4 5
Length	It is used to get the length of the tuple.	len(T1) = 5

Tuple Methods: count() and index()

count() method returns the number of times a specified value appears in the tuple

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

Output: 2

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(7)
print(x)
```

Output:2

Questions

- Python Tuple inbuilt functions
- When to use tuple and when to use list
- List differences between List and tuple
- What is packing ad unpacking in tuples. Give example

Set

- Set items are unordered, unchangeable, and do not allow duplicate values.

- * **Note:** Set *items* are unchangeable, but you can remove items and add new items.

Set

Creating a set

You can create a set with curly braces and within them you may put the elements.

Example

```
Set1 = { 1, 2, 1, 3, 4, 5 }
print(Set1)
```

You can see that we have kept a duplicate element inside set while declaring but when the actual set is created this element will be removed and you will get only unique elements.

Output

```
{1, 2, 3, 4, 5}
```

Accessing elements of Sets

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the **in** keyword.

Set

Sets may contain homogeneous or heterogeneous elements

Example

```
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
```

Sets can be made from lists

```
my_set = set([1, 2, 3, 2])
print(my_set)
```

Output:

{1, 2, 3}

Get the Length of a Set

- thisset = {"apple", "banana", "cherry"}
  print(len(thisset))

type()

- myset = {"apple", "banana", "cherry"}
  print(type(myset))

set() Constructor

- It is also possible to use the set() constructor to make a set

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)

How to create an empty set in python?

You can create an empty set in python by using the set() function. One can think an empty set can be created using the curly braces {}, but notice that python interprets empty curly braces as a dictionary. Let us look at this scenario using an example:

Ex1:

```
 a ={}
 type(a)
```

O/p: <class 'dict'>

```
a= set()
 type(a)
```

O/p: <class 'set'>

Access Items

You cannot access items in a set by referring to an index or a key.

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)

print("banana" in thisset)
```

Note: Once a set is created, you cannot change its items, but you can add new items.

- Add Items

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Add Sets

To add items from another set into the current set, use the update() method

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

```
{'banana', 'cherry', 'papaya', 'apple', 'pineapple', 'mango'}
```

Remove Item

Remove "banana" by using the **remove**() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

Note: If the item to remove does not exist, remove() will raise an error.

Remove "banana" by using the **discard**() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

Note: If the item to remove does not exist, discard() will NOT raise an error.

The del keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

Join Two Sets

- You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another
- The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Keep ONLY the Duplicates

- The intersection_update() method will keep only the items that are present in both sets.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are very good at data retrieval you just need to know the key.
- Dictionaries are written with curly brackets, and have keys and values
- Each key and value pairs are separated by colon. on the left side is the key and right side is the value.

```
>>> dict1 = {}  #Empty Dictionary declaration

>>> dict2 = {"k":"value","two":2}
>>> type(dict2)
      <class 'dict'>
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:Duplicate values will overwrite existing values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)                 #output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
print(len(thisdict))            #3
print(type(thisdict))           #dict
```

Dictionary

Accessing elements of Dictionary

- You can access the items of a dictionary by referring to its key name, inside square brackets.
- There is also a method called **get()** that will also help in accessing the element from a dictionary.

Example

```
d = {1:“hhh","2":"Two"}
print("d[1]=",d[1])  #Accessing with key
print("d[2]=",d.get("2"))  #Accessing using get() function
```

Output

```
('d[1]=', hhh')
('d[2]=', 'Two')
```

Dictionary

Manipulate or add elements in dictionary

```
        my_info = {'name':'Rahul', 'age': 25}
         # update value
        my_info['age'] = 26
       print(my_info)

      #Adding Items
thisdict =         {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)  #{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Removing Items

The pop() method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)  #{'brand': 'Ford', 'year': 1964}
```

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)

#{'brand': 'Ford', 'model': 'Mustang'}
```

The del keyword can also delete the dictionary completely:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

The clear() method empties the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)   #{}
```

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

```
thisdict =	{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

Print all *values* in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

Loop through both keys and values, by using the items() method:

```
thisdict =      {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Output:

```
brand Ford
model Mustang
year 1964
```

Copy a Dictionary

```
t = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
m = t.copy()
print(m)
```

Another way to make a copy is to use the built-in function dict().

Make a copy of a dictionary with the dict() function:

```
m = dict(t)
print(m)
```

- Homework
- Practice more python Dictionaries methods:

Built-in Dictionary Functions & Methods

- Can refer https://www.tutorialspoint.com/python/python_dictionary.htm

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

output_list = [output_exp for var in input_list if (var satisfies this condition)]

Note that list comprehension may or may not contain an if condition. List comprehensions can contain multiple for (nested list comprehensions).

Example 1: Suppose we want to create an output list which contains only the even numbers which are present in the input list. Let's see how to do this using *for loops* and *list comprehension* and decide which method suits better.

```
# Constructing output list WITHOUT using List comprehensions
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_list = []
# Using loop for constructing output list
for var in input_list:
    if var % 2 == 0:
        output_list.append(var)

print("Output List using for loop:", output_list)
```

Output:

Output List using for loop: [2, 4, 4, 6]

```
# Using List comprehensions for constructing output list

input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

list_using_comp = [var for var in input_list if var % 2 == 0]

print("Output List using list comprehensions:", list_using_comp)
```

Output:

Output List using list comprehensions: [2, 4, 4, 6]

Advantages of List Comprehension

- More time efficient and space efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.

Example 1 : Suppose we want to create an output set which contains only the even numbers that are present in the input list. Note that set will discard all the duplicate values. Let's see how we can do this using for loops and set comprehension.

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
output_set = set()
# Using loop for constructing output set
for var in input_list:
    if var % 2 == 0:
        output_set.add(var)

print("Output Set using for loop:", output_set)
```

Output:

Output Set using for loop: {2, 4, 6}

 # Using Set comprehensions for constructing output set

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:", set_using_comp)
```

Output:

Output Set using set comprehensions: {2, 4, 6}

Dictionary Comprehensions:

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Example 1: Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values. Let's see how to do this using for loops and dictionary comprehension.

```
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}
# Using loop for constructing output dictionary
for var in input_list:
    if var % 2 != 0:
        output_dict[var] = var**3

print("Output Dictionary using for loop:", output_dict )
```

Output:

Output Dictionary using for loop: {1: 1, 3: 27, 5: 125, 7: 343}

```
# Using Dictionary comprehensions for constructing output dictionary

input_list = [1,2,3,4,5,6,7]

dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}

print("Output Dictionary using dictionary comprehensions:", dict_using_comp)
```

Output:

Output Dictionary using dictionary comprehensions: {1: 1, 3: 27, 5: 125, 7: 343}

Summary

Python has flexible and powerful data structures with in-built functions for all basic manipulation

Accessing, retrieving and doing any computation is so easy with Python.

Python Collections (Arrays)

- There are four collection data types in the Python programming language:
- List is a collection which is ordered and changeable. Allows duplicate members. []
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members. ()
- Set is a collection which is unordered and unindexed. No duplicate members. {}
- Dictionary is a collection which is unordered and changeable. No duplicate members. {}