Comprehension

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:

- **List Comprehensions**
- **Dictionary Comprehensions**
- **Set Comprehensions**
- **Generator Comprehensions**

List Comprehensions:

List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:

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

For e.g
1.If we want to create a list which contains square of elements of old list.

```
#Traditional approach
li = [1,2,3,4,5]
sqr = []
for i in li:
    sqr.append(i**2)

print(li)
print(sqr)
```

OUTPUT:
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]

```
#By comprehension:
sqr = [i**2 for i in li]
print(sqr)
```

OUTPUT:
[1, 4, 9, 16, 25]

2. If we want to create an output list which contains only the even numbers which are present in the input list.

#Traditional approach:

```
li = [1,2,3,4,5]
even = []
for i in li:
    if i%2 == 0:
        even.append(i)

print(even) #[2,4]
```

#List comprehension:

```
even = [i for i in li if i%2 == 0]
print(even) #[2,4]
```

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)}

For e.g.,

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.

#Traditional approach

```
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}

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}

#Dictionary comprehension

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}

2.Swap key value pair of given dictionary
d = {1:'a',2:'b',3:'c',4:'d'}
swap = {v:k for k,v in d.items()}
print(swap) # **{'a': 1, 'b': 2, 'c': 3, 'd': 4}**

Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets {}. The following is the basic structure of a set comprehension:

output_set ={output_exp for var in input_set if (var satisfies this condition)}

For e.g.
If we want to create a set which contains only the odd numbers which are present in the input list.
#Traditional approach:

```
li = [1,2,3,4,5]
odd = set()
for i in li:
    if i%2 != 0:
        odd.add(i)

print(odd) #{1,3,5}
```

#Set comprehension:

```
odd = {i for i in li if i%2 != 0}
print(odd) #{1,3,5}
```

Generator Comprehensions:

Generator Comprehensions are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don't allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient.

For e.g. Display only even elements from given list using generator comprehension.

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_gen = (var for var in input_list if var % 2 == 0)

print("Output values using generator comprehensions:", end = ' ')

for var in output_gen:
    print(var, end = ' ')
```

OUTPUT:

Output values using generator comprehensions: 2 4 4 6