

# **Python List and Dictionary Comprehension**

# List Comprehension

- ❑ **Python's List Comprehension is a quick and compact syntax to generate a list from a string or any other list.**
- ❑ **Creating a new list by performing an operation on each item in the existing list is a very straightforward way .**
- ❑ **Comprehension of the list is considerably faster than using the for loop to process a list.**

**A list comprehension generally consist of these parts :**

**Output expression ,input sequence,a variable representing member of input sequence and an optional predicate part.**

**For example :**

**list = [x \*\* 2 for x in range (1, 11) if x % 2 == 1]**

**here, x \*\* 2 is output expression,**

**range (1, 11) is input sequence,**

**x is variable and if x % 2 == 1 is predicate part.**

# Example: Separate Letters from String

```
chars=[]  
for ch in 'GKTCS INNOVATIONS':  
    chars.append(ch)  
print(chars)
```

```
>>>  
>>>  
>>>  
>>> chars=[]  
>>> for ch in 'GKTCS INNOVATIONS':  
...     chars.append(ch)  
... [ch for ch in 'GKTCS INNOVATIONS']  
File "<stdin>", line 3  
    [ch for ch in 'GKTCS INNOVATIONS']  
    ^  
SyntaxError: invalid syntax  
>>> print(chars)  
[]  
>>> [ch for ch in 'GKTCS INNOVATIONS']  
['G', 'K', 'T', 'C', 'S', ' ', 'I', 'N', 'N', 'O', 'V', 'A', 'T', 'I', 'O', 'N', 'S']  
>>> █
```

Online co

**Example to uses a list comprehension to build a list of squares of the numbers between 1 and 10.**

```
>>>  
>>>  
>>>  
>>> squares = [x*x for x in range(11)]  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
>>> █
```

# Nested loops

- ❑ Nested loops can be used in a comprehension expression of the list.
- ❑ In the example below, any combination of items from two lists in the form of a tuple is added to a third list object.

```
>>>
>>>
>>> list1=[1,2,3]
>>> list2=[4,5,6]
>>> CombLst=[(x,y) for x in list1 for y in list2]
>>> CombLst
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
>>> 
```

# Applications

One of the **applications** of list comprehension is to flatten a list comprising of multiple lists into a single list

```
>>>  
>>>  
>>> matrix=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
>>> flatList=[num for row in matrix for num in row]  
>>> flatList  
[1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> █
```

# Nested if conditions with list comprehension.

We can use nested if conditions with list comprehension.

```
>>>  
>>>  
>>> [x for x in range(21) if x%2==0 if x%5==0]  
[0, 10, 20]  
>>>  
>>> █
```



# If..else loop with list comprehension.

```
>>>
>>>
>>>
>>> obj=["Yes" if i%2==0 else "No" for i in range(10)]
>>> obj
['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No']
>>> obj=[str(i)+" = Yes" if i%2==0 else str(i)+" = No" for i in range(10)]
>>> obj
['0 = Yes', '1 = No', '2 = Yes', '3 = No', '4 = Yes', '5 = No', '6 = Yes', '7 = No', '8 = Yes', '9 = No']
>>>
>>>
>>>
```

# What is Dictionary Comprehension in Python?

- ❑ Dictionary comprehension is an elegant and concise way to create dictionaries.
- ❑ Syntax for dictionary comprehension

```
dictionary = {key: value for vars in iterable}
```

# Dictionary comprehensions

- ❑ Like List Comprehension, Python allows dictionary comprehensions.
- ❑ We can create dictionaries using simple expressions.
- ❑ A dictionary comprehension takes the form **{key: value for (key, value) in iterable}**

# Dictionary from a list using comprehension

```
>>>  
>>>  
>>> myDict = {x: x**2 for x in [1,2,3,4,5]}  
>>> print (myDict)  
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}  
>>> █
```

**We can use Dictionary comprehensions with if and else statements and with other expressions too.**

```
>>>  
>>> newdict = {x: x**3 for x in range(10) if x**3 % 4 == 0}  
>>> print(newdict)  
{0: 0, 2: 8, 4: 64, 6: 216, 8: 512}  
>>> □
```

# How to use Dictionary Comprehension..?

```
old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
dollar_to_pound = 0.76
new_price = {item: value*dollar_to_pound for (item, value) in
old_price.items()}
print(new_price)
```

## Output :

```
{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

# Conditionals in Dictionary Comprehension

## If Conditional Dictionary Comprehension

```
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}  
even_dict = {k: v for (k, v) in original_dict.items() if v % 2 == 0}  
print(even_dict)
```

### Output :

```
{'jack': 38, 'michael': 48}
```

# Multiple if Conditional Dictionary Comprehension

```
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}  
new_dict = {k: v for (k, v) in original_dict.items() if v % 2 != 0 if v < 40}  
print(new_dict)
```

**Output :**

```
{'john': 33}
```



# If-else Conditional Dictionary Comprehension

```
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}  
new_dict_1 = {k: ('old' if v > 40 else 'young') for (k, v) in  
               original_dict.items()}  
print(new_dict_1)
```

## Output :

```
{'jack': 'young', 'michael': 'old', 'guido': 'old', 'john': 'young'}
```

# Advantages of Using Dictionary Comprehension

- ❑ **As we can see, dictionary comprehension shortens the process of dictionary initialization by a lot. It makes the code more pythonic.**
- ❑ **Using dictionary comprehension in our code can shorten the lines of code while keeping the logic intact.**

# Warnings on Using Dictionary Comprehension

**Even though dictionary comprehensions are great for writing elegant code that is easy to read, they are not always the right choice.**

**We must be careful while using them as :**

- ☐ **They can sometimes make the code run slower and consume more memory.**
- ☐ **They can also decrease the readability of the code.**