List comprehension:

List:

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

For example, assume we want to create a list of squares, like:

```
>>> list1=[]

>>> for x in range(10):

        list1.append(x**2)

>>> list1

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

(or)

This is also equivalent to

```
>>> list1=list(map(lambda x:x**2, range(10)))

>>> list1

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

(or)

Which is more concise and redable.

```
>>> list1=[x**2 for x in range(10)]

>>> list1

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Similarily some examples:

```
>>> x=[m for m in range(8)]
>>> print(x)
[0, 1, 2, 3, 4, 5, 6, 7]

>>> x=[z**2 for z in range(10) if z>4]
>>> print(x)
[25, 36, 49, 64, 81]

>>> x=[x ** 2  for x in range (1, 11)   if x % 2 == 1]
>>> print(x)
[1, 9, 25, 49, 81]

>>> a=5
>>> table = [[a, b, a * b] for b in range(1, 11)]
>>> for i in table:
        print(i)


[5, 1, 5]
[5, 2, 10]
[5, 3, 15]
[5, 4, 20]
[5, 5, 25]
[5, 6, 30]
[5, 7, 35]
[5, 8, 40]
[5, 9, 45]
[5, 10, 50]
```

Generator Expression:

Generator expressions are similar to list comprehensions, but they return an iterator instead of a list. The syntax is the same, but without the square brackets:

```
>>> (expression for variable in iterable if condition)
```

For example,

```
>>>numbers = [1, 2, 3, 4, 5]
```

```
>>> squared_numbers_gen = (x**2 for x in numbers)
>>> print(next(squared_numbers_gen))  # 1
>>> print(next(squared_numbers_gen))  # 4
>>> print(next(squared_numbers_gen))  # 9
```

Use list comprehensions when:

- You need to create a new list from an existing iterable.
- You need to perform a simple transformation on each element.
- You need to filter out unwanted elements.

Use generator expressions when:

- You need to create an iterator over a large dataset.
- You need to perform a complex transformation on each element.
- You need to create an infinite sequence.