List Comprehensions

List comprehensions provide a concise way to create lists. Common applications include creating new lists where each element is the result of some operations applied to each member of another sequence or iterable.

Syntax:

```
[expression for item in iterable if condition]
```

Examples:

- Creating a list of squares :

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

even_squares = [x**2 for x in range(10) if x % 2 == 0]
```

Generator Expressions

Generator expressions are similar to list comprehensions, but they return a generator instead of a list. Generators are more memory-efficient because they yield items one by one using yield and do not store the entire list in memory.

Basic Syntax:

```
(expression for item in iterable if condition)
```

Examples:

- Creating a generator for squares:

```
squares_gen = (x**2 for x in range(10))
```

- Iterating through the generator:

```
for square in squares_gen:
    print(square)
```

Decorators

Decorators are a powerful and useful tool in Python since it allows programmers to modify the behavior of function or class methods. Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.