The syntax for the for loop is as follows:

```
for item in iterable:
    statement
```

Note that in the for loop syntax is not the same as the membership logical operator earlier discussed.

Here is a program example:

```
a = [2, 4, 6, 8, 10]
for elem in a:
    print(elem**2)
```

```
'Output': 4
    16
    36
    64
    100
```

To loop for a specific number of time, use the range() function.

```
for idx in range(5):
    print('The index is', idx)
```

```
'Output': The index is 0
     The index is 1
     The index is 2
     The index is 3
     The index is 4
```

List Comprehensions

Using list comprehension, we can succinctly rewrite a for loop that iteratively builds a new list using an elegant syntax. Assuming we want to build a new list using a for loop, we will write it as

```
new_list = []
for item in iterable:
    new_list.append(expression)
```

We can rewrite this as

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

Let's have some program examples.

```
squares = []
for elem in range(0,5):
    squares.append((elem+1)**2)

squares
'Output': [1, 4, 9, 16, 25]
```

The preceding code can be concisely written as

```
[(elem+1)**2 for elem in range(0,5)]
'Output': [1, 4, 9, 16, 25]
```

This is even more elegant in the presence of nested control structures.

```
evens = []
for elem in range(0,20):
    if elem % 2 == 0 and elem != 0:
        evens.append(elem)

evens
'Output': [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

With list comprehension, we can code this as

```
[elem for elem in range(0,20) if elem % 2 == 0 and elem != 0]
'Output': [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

The break and continue Statements

The break statement terminates the execution of the nearest enclosing loop (for, while loops) in which it appears.

```
for val in range(0,10):
    print("The variable val is:", val)
    if val > 5:
```