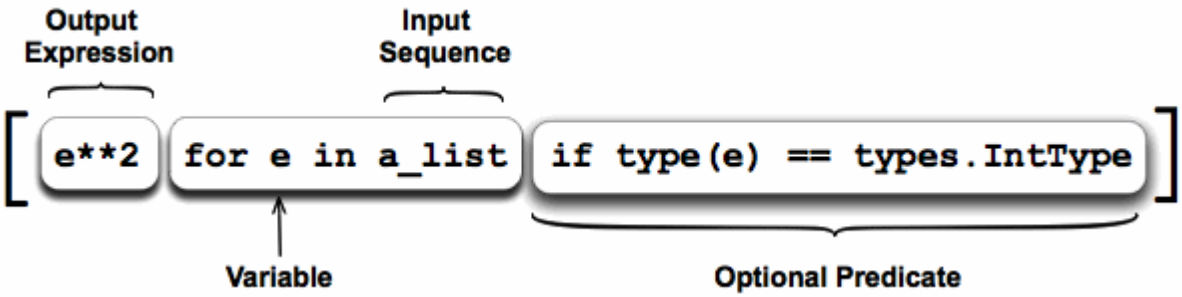
List Comprehensions

Comprehensions provide an easy way to build sequences from other sequences.

A comprehension construct consists of following parts:

- An Input Sequence
- A Variable representing members of the input sequence
- An Output Expression producing elements of the output list from members of the Input Sequence
- An Optional Predicate expression which filters the input sequence



1. Basic Comprehension

Try Code:

Use list comprehension to generate a list `list1 = x * 2` , where x is between 0 and 9.

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

In [21]:
```
[x*2 for x in range(10)]
```

Out[21]: `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]`

Use `while-loop` to implement equivalent code.

In [20]:
```
x = 0
result = []
while x < 10:
    result.append(x*2)
    x = x + 1

print(result)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Exercise:

Use list comprehension to generate a list of numbers wihch are `x * 2` if `x` is even or `x * 3` if `x` is odd, where x is between 0 and 9.

In [23]:
```
[x*2 if x%2==0 else x*3 for x in range(10)]
```

Out[23]: `[0, 3, 4, 9, 8, 15, 12, 21, 16, 27]`

Use `for-loop` to implement equivalent code.

In [26]:
```
result = []
for x in range(10):
    result.append(x*2 if x % 2 ==0 else x*3)

print(result)
```

```
[0, 3, 4, 9, 8, 15, 12, 21, 16, 27]
```

2. Comprehension with Filtering

You can filter list items in the comprehension using `if-statement` . Items will be included when the `if-statement` is evaluated to True.

- Note the difference between if-statements before for-loop and if-statements after for-loop

Exercise:

Use list comprehension to generate a list of numbers wihch are `x^2` if `x` is divisible by both 2 and 3, where `x` is between 0 and 19

In [28]:
```
[x*2 for x in range(20) if x%2==0 and x%3==0]
```

Out[28]: `[0, 12, 24, 36]`

Use `for-loop` to implement equivalent code.

In [27]:
```
result = []
for x in range(20):
    if x%2 == 0 and x%3 == 0:
        result.append(x*2)

print(result)
```

```
[0, 12, 24, 36]
```

3. Comprehension with Nested Loops (Optional)

Comprehension also allow you to generate list with nested loops. Subsecquent `for-loop` is nested in previous `for-loop` .

Exercise:

Implement a function `nested()` which generates and returns following nested list using duble for-loops.

```
[[0, 100], [0, 101], [0, 102], [1, 100], [1, 101], [1, 102], [2, 100], [2, 101], [2, 102]]
```

In [3]:

```
def nested():
    result = []
    for x in range(3):
        for y in range(100, 103):
            result.append([x,y])
    return result

r = nested()
# import pprint
# pprint.pprint(r)
print(r)
```

```
[[0, 100], [0, 101], [0, 102], [1, 100], [1, 101], [1, 102], [2, 100], [2, 101], [2, 102]]
```

Exercise:

Use list comprehension to generate re-write above funtion.

In [4]:

```
def nested():
    return [[x,y] for x in range(3) for y in range(100,103)]

r = nested()
print(r)
```

```
[[0, 100], [0, 101], [0, 102], [1, 100], [1, 101], [1, 102], [2, 100], [2, 101], [2, 102]]
```