Collection Comprehensions

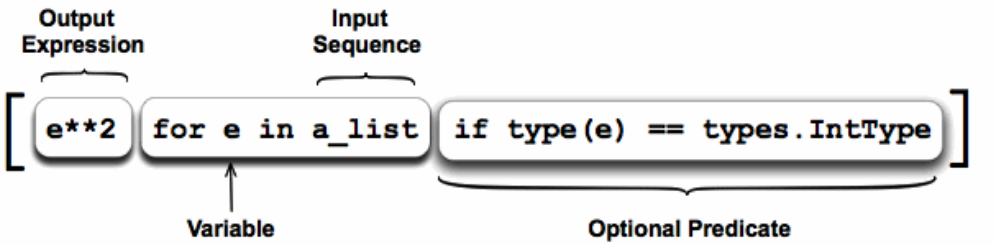
Comprehensions provide an easy way to build sequences from other sequences.

- List Comprehensions
- Set Comprehensions
- Dictionary Comprehensions

1. List Comprehensions

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



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]
```

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]
```

Comprehension with Nested Loops

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]]
```

2. Set Comprehensions

Similiar to List Comprehensions, we can construct sets using Set Comprehensions.

- Instead of using `[ ]` , it uses `{ }` instead.

Excercise:

- Ask user to enter a sentence
- Split the sentence into words
- Create a set of words, which appears more than once in the sentence

Sample Output:

```
Enter a sentence: get busy living or get busy dying
{'busy', 'get'}
```

In [9]:
```
s = input('Enter a sentence: ')
words = s.split()
result = {w for w in words if words.count(w) >= 2}
print(result)
```

```
Enter a sentence: get busy living or get busy dying
{'get', 'busy'}
```

3. Dictionary Comprehensions

Dictionary Comprehension uses `{ }` and its output expression must be **key-value** pair.

Exercise:

The `ord()` function converts a character into ASCII code; `chr()` function converts an ASCII code into corresponding character. ASCII code of 'A' = 65, 'B' = 66 ...

- Create a dictionary which gives character to ASCII mapping for character 'A' to 'F'

In [11]:
```
{ chr(x):x for x in range(65,70)}
```

Out[11]:
```
{'A': 65, 'B': 66, 'C': 67, 'D': 68, 'E': 69}
```

4. Zip() to Handle Multiple Sequences

To generate sequence from multiple input sequences, you can use `zip()` function. Function `zip()` maps similar index of multiple sequences so that they can be used just using as single entity.

- If iterables have different number of elements, zip() will only process smallest length of elements.

```
zip(*iterables)
```

Try Code:

```
z = zip(range(3), range(6), range(9))
list(z)
```

In [29]:
```
z = zip(range(3), range(6), range(9))
list(z)
```

Out[29]:
```
[(0, 0, 0), (1, 1, 1), (2, 2, 2)]
```

Exercise:

- Generate a list of odd numbers `odds` between 0 and 9
- Generate a list of even numbers `evens` between 0 and 9
- Generate another sequence `result` by adding odd number with corresponding even number of same index

Optional 1: Using `for-loop`

In [27]:
```
odds = [x for x in range(10) if x%2==1]
evens = [x for x in range(10) if x%2==0]
result = []
for i in range(len(odds)):
    result.append(odds[i] + evens[i])
print(result)
```

```
[1, 5, 9, 13, 17]
```

Option 2: Using `zip()`

In [24]:
```
odds = [x for x in range(10) if x%2==1]
evens = [x for x in range(10) if x%2==0]
result = [x + y for x,y in zip(odds, evens)]
print(result)
```

```
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8]
[1, 5, 9, 13, 17]
```

Zip again to Unzip

How do you unzip the zipped sequence `zipped` back to `odds` and `evens` ? **Zip to unzip**

Try Code:

```
x = list(range(0,9,2))
y = list(range(1,10,2))
z = list(zip(x, y))
print(z)

a, b = zip(*z)
print(a)
print(b)
```

In [19]:
```
x = list(range(0,9,2))
y = list(range(1,10,2))
z = list(zip(x, y))
print(z)

a, b = zip(*z)
print(a)
print(b)
```

```
[(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]
(0, 2, 4, 6, 8)
(1, 3, 5, 7, 9)
```