14.Comprehensions

January 20, 2019

1 Comprehension

1.1 Comprehensions

It is a brief way to create lists, sets and some other structures, i.e. shorthand for the loops. Here is some examples

```
In [1]: ten_numbers = [x for x in range(10)]
        ten_numbers
```

```
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

That code is analogical for this

```
In [2]: ten_numbers = []
        for i in range(10):
            ten_numbers.append(i)
        ten_numbers
```

```
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Or this

```
In [3]: ten_numbers = list(range(10))
        ten_numbers
```

```
Out[3]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

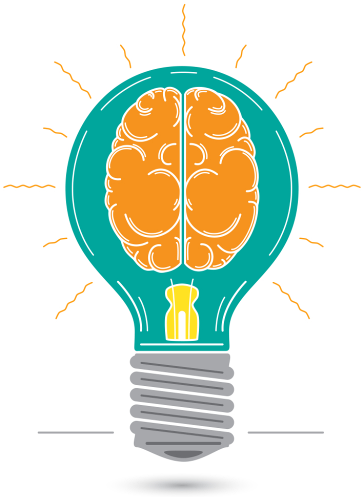
1.2 Morphology

Thus we have this basic structure

```
[iteration_variable for iteration_variable in collection]
```

It will simply copy collection to a list
But that's not all

```
In [7]: fractions = [0.1 * x for x in range(1, 50, 3)]
        fractions
```



```
Out[7]: [0.1,
         0.4,
         0.7000000000000001,
         1.0,
         1.3,
         1.6,
         1.9000000000000001,
         2.2,
         2.5,
         2.8000000000000003,
         3.1,
         3.4000000000000004,
         3.7,
         4.0,
         4.3,
         4.6000000000000005,
         4.9]
```

```
In [8]: # Similar
        fractions = []
        for i in range(1, 50, 3):
            fractions.append(0.1 * i)
        fractions
```

```
Out[8]: [0.1,
         0.4,
         0.7000000000000001,
         1.0,
         1.3,
         1.6,
         1.9000000000000001,
         2.2,
         2.5,
         2.8000000000000003,
         3.1,
         3.4000000000000004,
         3.7,
         4.0,
         4.3,
         4.6000000000000005,
         4.9]
```

```
In [6]: modules = [abs(x) for x in range(-10, 11)]
```

```
Out[6]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [9]: # Similar
        modules = []
        for i in range(-10, 11):
            modules.append(abs(i))
        modules
```

```
Out[9]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Thus structure can be more complex

```
[action(iteration_variable) for iteration_variable in collection]
```

where action is some operation with iteration_variable (element from collection)
That's not all either (this is infinite)

```
In [23]: evens = [i for i in range(20) if i % 2 == 0]
         evens
```

```
Out[23]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
In [24]: # Similar
         for i in range(20):
             if i % 2 == 0:
                 evens.append(i)
         evens
```

```
Out[24]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

And else of course
Note this strange rearrangement of parts when we use `if` and `else`

```
In [28]: odds_fenixes = [i if i % 2 == 0 else 'Phoenix' for i in range(10)]
         odds_fenixes
```

```
Out[28]: [0, 'Phoenix', 2, 'Phoenix', 4, 'Phoenix', 6, 'Phoenix', 8, 'Phoenix']
```

```
[action(iteration_variable) if predicate else another_variant for iteration_variable in collecti
```

1.3 Set comprehension

Everything stays the same, just substitute the brackets with braces

```
In [29]: evens = {i for i in range(20) if i % 2 == 0}
         evens
```

```
Out[29]: {0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
```

1.4 Dictionary comprehension

Almost the same

```
In [30]: # Simple copy
         original_dict = {1: 10, 2: 20, 3:30}
         imba_dict = {k: v for k, v in original_dict.items()}
         imba_dict
```

```
Out[30]: {1: 10, 2: 20, 3: 30}
```

```
In [32]: # Similar
         imba_dict = {}
         for k, v in original_dict.items():
             imba_dict[k] = v
         imba_dict
```

```
Out[32]: {1: 10, 2: 20, 3: 30}
```

You need 2 iterables to pass through or only 1 dependency from data in dict case

```
In [34]: fruits = ['mango', 'apple', 'pineapple', 'grape', 'lemon']
         volumes = [100, 150, 110, 200, 100]
         imba_dict = {fruit: volume for fruit, volume in zip(fruits, volumes)}
         imba_dict
```

```
Out[34]: {'apple': 150, 'grape': 200, 'lemon': 100, 'mango': 100, 'pineapple': 110}
```

```
In [33]: imba_dict = {number: 'int' for number in range(10)}
         imba_dict
```

```
Out[33]: {0: 'int',
          1: 'int',
          2: 'int',
          3: 'int',
          4: 'int',
          5: 'int',
          6: 'int',
          7: 'int',
          8: 'int',
          9: 'int'}
```

1.5 Nested comprehensions

You can simulate nested cycles

```
In [36]: combinations = [x + y for x in 'AB' for y in 'CD']
         combinations

Out[36]: ['AC', 'AD', 'BC', 'BD']

In [40]: matrix = [[3 * j + i for i in range(3)] for j in range(3)]
         matrix

Out[40]: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Parenthesis comprehension gives you a generator, we will talk about them later