Python Programming

by Narendra Allam

Chapter 8

Comprehensions, Lambdas and Functional Programming

Topics Covering

- List Comprehension
  - Creating a list using for loop
  - Comprehension to create a list
- Tuple Comprehension and generators
- Set Comprehension
- Dictionary Comprehension
- Zip and unzip
  - Creating List of tuples
  - List of tuples to list of tuple-sequences
- Enumerate
  - Adding index to a sequence
  - Starting custom index
- Lambdas
- Funcional Programming
  - map()
  - filter()
  - reduce()

Comprehension

List Comprehension

Comprehension is a short-hand technique to create data structures in-place dynamically. Comprehensions are faster than their other syntactical counterparts.

Creating a list using loop:

```
In[]  l = []
      for x in xrange(1, 11):
          l.append(x)
      print l
```

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

Comprehension to create a list:

```
In[]  l = [i for i in xrange(1, 11)]
      print l
```

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

Applying a function in list comprehension:

```
In[]  from math import sin
      l = [sin(i) for i in xrange(1, 6)]
      print l
```

```
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672, -0.75
68024953079282, -0.9589242746631385]
```

Filtering values from an exisiting list:

```
In[]  print l
      l1 = [x  for x in l if x > 0]
      print l1
```

```
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672, -0.75
68024953079282, -0.9589242746631385]
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672]
```

Using multiple for loops: Cartesian Product

```
In[]  cartesian = [(x, y) for x in ['a', 'b'] for y in ['p', 'q']]
      cartesian
```

```
Output: [('a', 'p'), ('a', 'q'), ('b', 'p'), ('b', 'q')]
```

Example: Converting teforenheit to celsius using list comprehension

```
In[]  temps = [45, 67, 89, 73, 45, 89, 113]
      cels = [(f-32.0)/(9.0/5.0) for f in temps]
      cels
```

```
Output: [7.222222222222222,
         19.444444444444443,
         31.666666666666664,
         22.77777777777778,
         7.222222222222222,
         31.666666666666664,
         45.0]
```

Tuple comprehension

We know that tuples are immutable, then how a tuple is being constructued dynamically. Python creates a generator instead of creating a tuple.

Note: Tuple comprehension is a generator

```
In[]  gen = (i for i in xrange(1, 6))
      print gen
```

```
      <generator object <genexpr> at 0x10d81edc0>
```

next() function is used to get the next item in the sequence.

```
In[]  next(gen)
```

```
Output: 1
```

```
In[]  next(gen)
```

```
Output: 2
```

and soon..

Set Comprehension

```
In[]  nums = {n**2 for n in range(10)}
```

```
In[]  nums
```

```
Output: {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}
```

Zip and Enumerate

Creating list of tuples from more than one sequence

zip() function packs items from multiple sequences into a list of tuples, and we know how to iterates list of tuples. zip() takes len() of the sequence with smallest size and only makes those many iterations.

```
In[]    l1 = [3, 4, 5, 7, 1]
        l2 = ["Q", "P", "A", "Z", "T", 'K']
        l3 = [True, False, True, True, False, True]

        zip(l1, l2, l3)
```

```
Output: [(3, 'Q', True),
         (4, 'P', False),
         (5, 'A', True),
         (7, 'Z', True),
         (1, 'T', False)]
```

In the above example zip produces only 5 tuples as l1 is the sequence with smalles t length.

Iterating more than one iterable using zip()

```
In[]    l1 = [3, 4, 5, 7, 1]
        l2 = ["Q", "P", "A", "Z", "T", 'K']
        for x, y in zip(l1, l2):
            print x, y
```

```
3 Q
4 P
5 A
7 Z
1 T
```

Working with multiple types for sequences

```
In [ ]: l = [ 3, 4, 2, 1, 9, 6]
        a = 'Apple'
        s = {4.5, 6.7, 3.4, 9.8}
        zip(l, a, s)
```

Unzipping into multiple sequences(tuples)

```
In[]  lt = [(3, 'Q'), (4, 'P'), (5, 'A'), (7, 'Z'), (1,  'T')]
```

```
In[]  p, q = zip(*lt)
```

```
In[]  p
```

```
Output: (3, 4, 5, 7, 1)
```

```
In[]  q
```

```
Output: ('Q', 'P', 'A', 'Z', 'T')
```

Creating a dict using zip

```
In[]  keys = [3, 4, 5, 7, 1]
      values = ["Q", "P", "A", "Z", "T"]
      dict(zip(keys, values))
```

```
Output: {1: 'T', 3: 'Q', 4: 'P', 5: 'A', 7: 'Z'}
```

enumerate

Associating sequences with positional values, index starting from zero

```
In[]  l = ["Q", "P", "A", "Z",  "T"]

      for idx, val in  enumerate(l):
          print idx, "->", val
```

```
0 -> Q
1 -> P
2 -> A
3 -> Z
4 -> T
```

Custom 'start' value

```
In[]  l = ["Q", "P", "A", "Z", "T"]
      for idx, val in enumerate(l, start=1):
          print idx, "->", val
```

```
1 -> Q
2 -> P
```

```
3 -> A
4 -> Z
5 -> T
```

Dict Comprehension

Creating a dict using two lists

```
In[] keys = [x for x in range(1, 6)]
     values = ['one', 'Two', 'Three', 'Four', 'Five']
     d = {k: v for k, v in zip(keys, values)}
     print d
```

```
{1: 'one', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
```

setting default value 0 for all keys

```
In[] keys = ['Orange', 'Apple', 'Peach', 'Banana', 'Grape']
     d = {k: 0 for k in keys}
     print d
```

```
{'Orange': 0, 'Grape': 0, 'Apple': 0, 'Peach': 0, 'Banana': 0}
```

Functional Programming

- map()
- filter()
- reduce()

For loop based implementation

```
In[] temps_farenheit = [45, 67, 89, 73, 45, 89, 113]

     def farenheit_to_celcius(f):
         c = (f-32.0)/(9.0/5.0)
         return c

     temps_celicius = []

     for t in temps_farenheit:
         temps_celicius.append(farenheit_to_celcius(t))

     print temps_celicius
```

```
[7.222222222222222, 19.444444444444443, 31.666666666666664, 22.777
77777777778, 7.222222222222222, 31.666666666666664, 45.0]
```

List Comprehension

```
In[]  temps_farenheit = [45, 67, 89, 73, 45, 89, 113]

      def farenheit_to_celcius(f):
          c = (f-32.0)/(9.0/5.0)
          return c

      temps_celcius = [farenheit_to_celcius(t) for t in  temps_farenheit]
      print temps_celcius
```

```
[7.222222222222222, 19.444444444444443, 31.666666666666664, 22.777
77777777778, 7.222222222222222, 31.666666666666664, 45.0]
```

Using map()

```
In[]  temps_farenheit = [45, 67, 89, 73, 45, 89, 113]

      def farenheit_to_celcius(f):
          c = (f-32.0)/(9.0/5.0)
          return round(c, 2)

      temps_celcius = map(farenheit_to_celcius, temps_farenheit)
      print temps_celcius
```

```
[7.22, 19.44, 31.67, 22.78, 7.22, 31.67, 45.0]
```

```
In[]  y = 20
      x = 20 if y > 10 else 30
```

Using filter()

```
In[] temps_farenheit = [45, 67, 89, 73, 45, 89, 113]

     def farenheit_to_celcius(f):
         c = (f-32.0)/(9.0/5.0)
         return c

     temps_celcius = map(farenheit_to_celcius, temps_farenheit)
     room_temp = 27

     def more_than_room_temp(t):
         return True if t > room_temp else False

     filter(more_than_room_temp, temps_celcius)
```

```
Output: [31.666666666666664, 31.666666666666664, 45.0]
```

Using reduce()

```
In[] def add(x, y):
         return x + y

     reduce(add, [5, 6, 7, 8, 9, 1, 9])
```

```
Output: 45
```

Note: We should pass a callable object or function to reduce() function, which must take 2 parameters and return one value

```
In[] def add(x, y, z):
         return x + y + z

     reduce(add, [5, 6, 7, 8, 9, 1, 9])
```

```
-----------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-1-988168ed6472> in <module>()
      2     return x + y + z
      3
----> 4 reduce(add, [5, 6, 7, 8, 9, 1, 9])

TypeError: add() takes exactly 3 arguments (2 given)
```

we can use variable arguments function in reduce(), but that deosn't help any, as reduce() passes exactly two values to the callable object. We cannot control this.

```
In[]  def add(*args):
          print len(args)
          return  sum(args)

      reduce(add, [5, 6, 7, 8, 9, 1, 9])
```

```
      2
      2
      2
      2
      2
      2

Output: 45
```

Using lambdas

- lambda is anonymous function
- lambda is inline function
- lambda is single line function

When ever we need use-and-throw functions(only one-time usage), lamdas are preferable.

Syntax:

```
lambda params: expression
```

```
In[]  f = lambda x: x*x
      f(4)
```

```
Output: 16
```

```
In[]  f = lambda x, y: x*y
      f(4,5)
```

```
Output: 20
```

In python, **lambda**s are used along with functional tools, **map()**, **reduce()** and **filter()**.

Above code can be re written using lambdas as below,

```
In[]  temps_farenheit = [45, 67, 89, 73, 45, 89, 113]
      room_temp = 27

      temps_celcius = map(lambda t: round((t-32.0)/(9.0/5.0), 2), temps_f
      arenheit)
      print 'Temps in celcius:', temps_celcius

      vals = filter(lambda t: True if t > room_temp else False, temps_cel
      cius)
      print 'Temps > room temperature:', vals

      cum_sum = reduce(lambda x, y: x+y, [5, 6, 7, 8, 9, 1])
      print 'Aggregate value: ', cum_sum
```

```
Temps in celcius: [7.22, 19.44, 31.67, 22.78, 7.22, 31.67, 45.0]
Temps > room temperature: [31.67, 31.67, 45.0]
Aggregate value:  36
```

Interview Questions

1. What is lambda?
2. What is map(), reduce and filter()
3. list comprehension vs tuple comprehension
4. What zip() function does?
5. What is unzipping()
6. list comprehension vs map() vs for loop which is faster?

```
In [ ]:
```