Python Programming

by Narendra Allam

Copyright 2018

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 [1]:

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

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

Comprehension to create a list:

In [2]:

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

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

Applying a function in list comprehension:

In [3]:

```
from math import sin
l = [sin(i) for i in range(1, 11)]
print(l)
```

```
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672, -0.756802
4953079282, -0.9589242746631385, -0.27941549819892586, 0.6569865987187
891, 0.9893582466233818, 0.4121184852417566, -0.5440211108893699]
```

round(): function

In [4]:

```
from math import sin
l = [round(sin(i), 2) for i in range(1, 11)]
print(l)
```

```
[0.84, 0.91, 0.14, -0.76, -0.96, -0.28, 0.66, 0.99, 0.41, -0.54]
```

Filtering values from an exisiting list:

In [5]:

```
print ("List:")
print (l)
l1 = [x for x in l if x > 0]
print ('Filtered List:')
print (l1)
```

```
List:
[0.84, 0.91, 0.14, -0.76, -0.96, -0.28, 0.66, 0.99, 0.41, -0.54]
Filtered List:
[0.84, 0.91, 0.14, 0.66, 0.99, 0.41]
```

Using multiple for loops: Cartesian Product

In [6]:

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

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

Above is equivalent of the below for loop:

In [7]:

```
l = []
for x in ['a', 'b']:
    for y in ['p', 'q']:
        l.append((x, y))
print (l)
```

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

Example: Converting forenheit to celsius using list comprehension

In [8]:

```
temps = [45, 67, 89, 73, 45, 89, 113]
cels = [round((f-32.0)/(9.0/5.0), 2) for f in temps]
print(cels)
```

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

Exercise: List of temps less than 30 degrees celcius

In [9]:

```
[t for t in cels if t < 30]
```

Out[9]:

```
[7.22, 19.44, 22.78, 7.22]
```

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 [10]:

```
gen = (i for i in range(1, 6))
print(gen)
```

```
<generator object <genexpr> at 0x10ca16e60>
```

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

In [11]:

```
next(gen)
```

Out[11]:

```
1
```

In [12]:

```
next(gen)
```

Out[12]:

```
2
```

and soon..

Set Comprehension

In [13]:

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

In [14]:

```
nums
```

Out[14]:

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

Zip

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 [17]:

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

for t in zip(l1, l2, l3):
    print(t)
```

```
(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 [18]:

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

for x, y in zip(l1, l2):
    print (x, y)
```

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

In [19]:

```
l1 = [3, 4, 5, 7, 1]
l2 = ["Q", "P", "A", "Z", "T", 'K', 'B']
l3 = [True, False, True, True, False, True]
l4 = [99, 44, 55, 66, 77, 11, 88]
it1 = iter(l1)
it3 = iter(l3)

for t in zip(l2, l4):
    print((next(it1, 0), next(it3, 0)) + t)
```

```
(3, True, 'Q', 99)
(4, False, 'P', 44)
(5, True, 'A', 55)
(7, True, 'Z', 66)
(1, False, 'T', 77)
(0, True, 'K', 11)
(0, 0, 'B', 88)
```

Working with multiple types for sequences

In [20]:

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

```
(3, 'A', 9.8)
(4, 'p', 3.4)
(2, 'p', 4.5)
(1, 'l', 6.7)
```

Unzipping into multiple sequences(tuples)

In [21]:

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

In [22]:

```
for x in zip(*lt):
    print(x)
```

```
(3, 4, 5, 7, 1)
('Q', 'P', 'A', 'Z', 'T')
```

Creating a dict using zip

In [23]:

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

Out[23]:

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

enumerate

Associating sequences with positional values, index starting from zero

In [24]:

```
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 [27]:

```
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 [28]:

```
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 [29]:

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

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

Functional Programming

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

For loop based implementation

In [30]:

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

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

temps_celicius = []

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

print(temps_celicius)
```

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

List Comprehension

In [31]:

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

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

Using map()

In [32]:

```
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)

for x in temps_celcius:
    print(x)
```

```
7.22
19.44
31.67
22.78
7.22
31.67
45.0
```

Using filter()

In [33]:

```
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)

room_temp = 27

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

print('\nTemps more than room temp:')
for x in filter(more_than_room_temp, temps_celcius):
    print(x)
```

```
Temps more than room temp:
31.67
31.67
45.0
```

Using reduce()

In [40]:

```
from functools import reduce

def add(x, y):
    return x + y

reduce(add, [1, 3, 4, 5])
```

Out[40]:

```
13
```

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

In [39]:

```
import functools

def add(x, y, z):
    return x + y + z

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

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-39-321da66c57b3> in <module>()
      4     return x + y + z
      5
----> 6 functools.reduce(add, [5, 6, 7, 8, 9, 1, 9])

TypeError: add() missing 1 required positional argument: 'z'
```

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 [41]:

```
import functools
def add(*args):
    print (len(args))
    return sum(args)

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

```
2
2
2
2
2
2
```

Out[41]:

```
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), lambdas are preferable.

Syntax:

```
lambda params: expression
```

In [42]:

```
f = lambda x: x*x
f(4)
```

Out[42]:

```
16
```

In [43]:

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

Out[43]:

```
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 [44]:

```
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_farenheit)
print ('Temps in celcius:', temps_celcius)

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

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

```
Temps in celcius: <map object at 0x10cae5b38>
Temps > room temperature: <filter object at 0x10ca6f7b8>
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?