Python-Reminders

November 14, 2021

1 Python Reminders

I assume that you all know Python. A brief introduction to Python Basics can be found in this notebook from last year (ipynb, html). Here we will only review some useful concepts

1.0.1 List Comprehension

Recall the mathematical notation:

$$L_1 = \left\{x^2 : x \in \{0 \dots 9\}\right\}$$

$$L_2 = \left(1, 2, 4, 8, \dots, 2^{12}\right)$$

[1]:
```python
L1 = [x**2 for x in range(10)] # range(n): returns an iterator over the numbers 0,...,n-1
L2 = [2**i for i in range(13)]
print (L1)
print (L2)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
```

[2]:
```python
L12 = []
for x in range(10):
    L12.append(x**2)
L12
```

[2]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List comprehension with conditions

$$M = \{x \mid x \in L_1 \text{ if } x \text{ is even}\}$$

[3]:
```python
L3 = [x for x in L1 if x % 2 == 0]
print (L3)
```

```
[0, 4, 16, 36, 64]
```

Nested use of link comprehension

```
[4]: [x for x in [x**2 for x in range(10)] if x % 2 == 0]
```

```
[4]: [0, 4, 16, 36, 64]
```

Use of list comprehension for string processing

```
[4]: words = 'The quick brown fox jumps over the lazy dog'.split()
     print(words)
     upper = [w.upper() for w in words]
     print(upper)
     upper_lower = [[w.upper(), w.lower()] for w in words]
     print(upper_lower)
     long_words = [x for x in words if len(x) > 3]
     print(long_words)
```

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
['THE', 'QUICK', 'BROWN', 'FOX', 'JUMPS', 'OVER', 'THE', 'LAZY', 'DOG']
[['THE', 'the'], ['QUICK', 'quick'], ['BROWN', 'brown'], ['FOX', 'fox'],
['JUMPS', 'jumps'], ['OVER', 'over'], ['THE', 'the'], ['LAZY', 'lazy'], ['DOG',
'dog']]
['quick', 'brown', 'jumps', 'over', 'lazy']
```

Use list comprehension for obtaining input

```
[7]: s = input('Give numbers separated by comma: ')
     x = [int(n) for n in s.split(',')]
     print(x)
```

```
Give numbers separated by comma: 1,5,3
[1, 5, 3]
```

Creating vectors and matrices

Create a vector of 10 zeros

```
[8]: z = [0 for i in range(10)]
     print(z)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Create a 10x10 matrix with all 0s

```
[9]: M = [[0 for i in range(10)] for j in range(10)]
     M
```

```
[9]: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Set the diagonal to 1

[10]:

```
for i in range(10): M[i][i] = 1
M
```

[10]:

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

Create a list of random integers in [0,99]

[10]:

```
import random
R = [random.choice(range(100)) for i in range(10)]
print(R)
```

```
[79, 60, 3, 47, 23, 41, 66, 15, 91, 55]
```

Removing elements from a list

Removing elements from a list while you iterate it can lead to problems

[5]:

```
L = [1,2,4,5,6,8]
for x in L:
    if x%2 == 0:
        L.remove(x)
print(L)
```

```
[1, 4, 5, 8]
```

Another way to do this using list comprehension:

[13]:

```
L = [1,2,4,5,6,8]
L = [x for x in L if x%2 == 1] #creates a new list
print(L)
```

```
[1, 5]
```

```
[14]: L = [1,2,4,5,6,8]
      R =[y for y in L if y%2 == 0]
      for x in R: L.remove(x)
      print(L)
```

```
[1, 5]
```

```
[15]: L = [1,2,4,5,6,8]
      R =[y for y in L if y%2 == 0]
      L = [x for x in L if x not in R]
      print(L)
```

```
[1, 5]
```

Using a dictionary in the list comprehension

```
[16]: D = {'A':1,'B':5,'C':4,'D':2}
      print([x for x in D if D[x]>2])
```

```
['B', 'C']
```

1.0.2 Dicitonary Comprehension

We can create dictionaries in a similar way as with list comprehension

```
[17]: {str(i):i for i in [1,2,3,4,5]}
```

```
[17]: {'1': 1, '2': 2, '3': 3, '4': 4, '5': 5}
```

```
 [7]: fruits = ['apple', 'mango', 'banana','cherry']
      fl = {f:len(f) for f in fruits}
      fl
```

```
 [7]: {'apple': 5, 'mango': 5, 'banana': 6, 'cherry': 6}
```

```
[19]: f_dict = {f.capitalize():i for i,f in enumerate(fruits)}
      print(f_dict)
```

```
{'Apple': 0, 'Mango': 1, 'Banana': 2, 'Cherry': 3}
```

```
[20]: {v:k for k,v in f_dict.items()}
```

```
[20]: {0: 'Apple', 1: 'Mango', 2: 'Banana', 3: 'Cherry'}
```

1.0.3 Using the right data structure

Using the right data structure makes a big difference when handling large data. Dictionaries and sets have expected constant time for finding an element, while lists have linear complexity. Even logarithmic time is significantly faster than linear.

Example

Looking for 10K integers in a collection of 10K integers

[13]:

```
L = [random.choice(range(10000000)) for i in range(100000)]
S = set(L)
Q = [random.choice(range(10000000)) for i in range(100000)]
import time
start = time.time()
[x for x in Q if x in L]
end = time.time()
print(end - start)
```

```
131.8199474811554
```

[14]:

```
start = time.time()
[x for x in Q if x in S]
end = time.time()
print(end-start)
```

```
0.014998912811279297
```

Example

You are given a graph in the form of a collection of edges, that is, pairs of vertices

How do you store it in order to be able to quickly answer if there is an edge (x,y) in the graph, and also to get all the neighbors of node?

Create a dictionary with nodes as the keys, and sets with neighboring nodes as values

[25]:

```
E = [(1,2),(2,3),(2,5),(2,6),(2,7),(3,4),(3,5),(5,6),(5,7),(7,8),(8,9),(8,10)]
G = {}
for (x,y) in E:
    if x not in G:
        G[x] = set()
    if y not in G:
        G[y] = set()
    G[x].add(y)
    G[y].add(x)
G
```

[25]:

```
{1: {2},
 2: {1, 3, 5, 6, 7},
 3: {2, 4, 5},
 5: {2, 3, 6, 7},
 6: {2, 5},
 7: {2, 5, 8},
 4: {3},
 8: {7, 9, 10},
 9: {8},
 10: {8}}
```

1.1 The Random library

We will often need to work with randomness. A library for this that is part of the main Python distribution is the random library.

Useful functions: 1. seed: Allows you to repeat the same random choices in different experiments 2. random: produces a random real number between 0 and 1 3. randint: select int for a range 4. choice: select an element from a list 5. choices: select k elements form a list with replacement. It is possible to use weights 6. shuffle: suffle a list of elements 7. sample: sample k elements from a list

1.1.1 Example

How do I implement the following?

With probability 0.7 I print 'A', with probability 0.2 I print 'B', and with probability 0.1 I do nothing.

[5]:

```python
import random
p = random.random()
if p <0.7: print('A')
elif p < 0.9: print('B')
```

1.1.2 Example

From a list I want to sample k elements where k is a parameter. I want my samples for the different k's to have the property that smaller samples are subsets of bigger samples. That is, the sample of size k+1 will contain one more random element to the sample of size k.

[16]:

```python
L = [i for i in range(20)]
random.shuffle(L)
sample4 = L[:4]
sample5 = L[:5]
```

[17]:

```python
print(sample4)
print(sample5)
```

```
[13, 6, 2, 18]
[13, 6, 2, 18, 17]
```