Branch: master ▾   Find file   Copy path

ma591spring2020 / material / datastructures.ipynb

asaibab Add files via upload

`8d20bfd`   4 days ago

1 contributor

`<>` 📄   Raw   Blame   History   🖥️ ✏️ 🗑️

1188 lines (1187 sloc)   22 KB

Built-in data structures

1. Lists

```
a = [1,2,3]
```

2. Tuples

```
a = (1,2,3)
```

3. Dictionaries

```
a = {'one':1, 'two': 2, 'three': 3}
```

Lists

Lists

- A list is a collection of (unlabelled) items which can be any data type.
- The elements items are comma separated
- Can change the size and the elements of a list
- A list can be used as an array, stack, queue.

Constructors

```
In [38]: lst = list()    # Empty list
         lst = []
         lst = [1,2,3]
         print(lst)

         [1, 2, 3]
```

```
In [2]: # Construct a list by enumeration
        num = [i**2 for i in range(10)]
        print(num)

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

Properties and methods

```
In [3]: #Length of a list
        l = [1,2,3]
        len(l)

Out[3]: 3
```

```
In [4]: #Reverse elements in place
```

```
l.reverse()
print("Reversed list is", l)

#Sort a list
l.sort()
print("Sorted list is ",l)
```

```
Reversed list is [3, 2, 1]
Sorted list is  [1, 2, 3]
```

reverse and sort are functions associated with a list (class). We will call such functions as methods.

Adding elements to a list

In [5]:
```
#Add to a list
l.append(4)
print(l)
```

```
[1, 2, 3, 4]
```

In [39]:
```
# Insert an element
l.insert(5,'6') # 1st argument: position, #2nd argument: val
ue
print(l)
```

```
[1, 'a', [1, 2, 3], '6']
```

In [40]:
```
## Add two different lists
l += [12,13]
print(l)
```

```
[1, 'a', [1, 2, 3], '6', 12, 13]
```

Data types in list

Lists can be constructed out of different data types, or even a mix of types.

In [8]:
```
l = ['a','b','c']
print(l)
```

```
['a', 'b', 'c']
```

In [9]:
```
l = [1, 'a', [1,2,3]]
print(l)
```

```
[1, 'a', [1, 2, 3]]
```

Indexing

Recall that counting in Python begins from 0. However, there are multiple ways to index an element. Consider a list of size 5.

```
In [10]: L = ['a','b','c','d','e']
```

We can access the elements of the list in multiple ways:

'a'	'b'	'c'	'd'	'e'	
0	1	2	3	4	
-5	-4	-3	-2	-1	

```
In [11]: print(L[0], L[-5])
         print(L[1], L[-4])
         print(L[2], L[-3] )
```

```
a a
b b
c c
```

Out of range exception

When we access an element that is out of range, this results in an error.

```
In [12]: L = ['a','b','c','d','e']

         try:
             print(L[5])
         except IndexError:
             print('IndexError: list index out of range')
```

```
IndexError: list index out of range
```

Slicing lists

Slicing the list has the general syntax

```
listname[start:end:stride]
```

where

- start determines starting point
- end is one index higher than the end point
- stride is the spacing between selected elements

```
In [13]: L = ['a','b','c','d','e']
         print(L[0:2:1])
         print(L[0:5:2])
         print(L[-5:-1:1])
```

```
['a', 'b']
['a', 'c', 'e']
['a', 'b', 'c', 'd']
```

Slicing lists

One or more quantities can be skipped while slicing. Let's look at some examples

```
In [14]:  print(L[:3:1])       #skip the beginning (context dependent)
          print(L[2::1])       #skip the end (context dependent)
          print(L[:3])         #skip the stride, default stride is 1
          print(L[::])         #skip everything, gives the whole list
```

```
['a', 'b', 'c']
['c', 'd', 'e']
['a', 'b', 'c']
['a', 'b', 'c', 'd', 'e']
```

```
In [15]:  print(L[3:0:-1])    # Negative stride
          print(L[3::-1])     # The end point is now the start of the a
          rray
          print(L[-1:-6:-1])  # reverses the array
          print(L[::-1])      # easier way to reverse array
```

```
['d', 'c', 'b']
['d', 'c', 'b', 'a']
['e', 'd', 'c', 'b', 'a']
['e', 'd', 'c', 'b', 'a']
```

Assignment/Modification using slicing and indexing

We can not only access elements using slicing and indexing, but we can also assign/modify the elements.

```
In [43]:  L = [1,2,3,4,5]

          L[0] = 17.1
          print(L)
          L[1:3] = ['a','b','c']
          print(L)
          L[-1] += 5.
          print(L)
```

```
[17.1, 2, 3, 4, 5]
[17.1, 'a', 'b', 'c', 4, 5]
[17.1, 'a', 'b', 'c', 4, 10.0]
```

Shallow vs deep copy

```
In [17]:  a = [1,2,3]
          b = a
          b.append(4)
          print(a, b)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

What happened here? b is not a copy of a, but a **pointer to** a. This means, when b is changed, a is also changed. This kind of copying is called a *shallow copy*.There is also an option for

deep copy.

```
In [18]: b = a.copy()
         b.append(5)
         print(a,b)
```

```
[1, 2, 3, 4] [1, 2, 3, 4, 5]
```

Other useful commands

```
In [19]: L = [1,2,3,3,4]
         print(min(L))              #Minimum entry
         print(max(L))              #Maximum entry
         print(L.count(3))          #Count number of entries that have
         the given value
         print(type(L))             #Prints the datatype of L
         print(isinstance(L,list)) #Checks if L is a list
```

```
1
4
2
<class 'list'>
True
```

Tuples

Tuples

Tuples are like lists except they are immutable (can't add or change entries).

```
In [20]: t = (1,2,[1,2,3])
         print(t)
```

```
(1, 2, [1, 2, 3])
```

```
In [21]: print("Length of tuple is ", len(t))
         print("First entry of the tuple is ", t[0]) #First entry of
          the tuple
```

```
Length of tuple is  3
First entry of the tuple is  1
```

Most of the commands used in the context of lists are also appropriate here.

Tuples cannot be manipulated

```
In [22]: try:
             t[1] = 4
             t.append(4)
         except:
             print('An error occured because a tuple cannot be manipu
```

```
lated.')
```

An error occured because a tuple cannot be manipulated.

Tuples into lists and back

```
In [23]: T = (1,2,'3')
         print(T)
         L = list(T)
         L.append(4.0)
         print(L)
```

```
(1, 2, '3')
[1, 2, '3', 4.0]
```

```
In [24]: M = tuple(L)
         print(M)
```

```
(1, 2, '3', 4.0)
```

Dictionaries

Dictionary

Is a collection of key value pairs. As motivation, consider two different lists:

```
In [25]: names = ['Newton', 'Einstein', 'VonNeumann']
         year  = [1643, 1879, 1903]
```

The year corresponds to the names; but maintaining two different lists can be cumbersome.

```
In [26]: scientists = {'Newton': 1643, 'Einstein': 1879, 'VonNeumann'
         :1903}
         #or
         scientists = dict(Newton = 1643, Einstein = 1879, VonNeumann
         = 1903)
         print(scientists)
```

```
{'Newton': 1643, 'Einstein': 1879, 'VonNeumann': 1903}
```

Details of dictionaries

Dictionary maintains two lists of keys and values.

```
In [27]: #List of keys
         print(scientists.keys())
```

```
dict_keys(['Newton', 'Einstein', 'VonNeumann'])
```

```
In [28]: #List of values
         print(scientists.values())
```

```
                  dict_values([1643, 1879, 1903])
```

```
In [29]:  scientists['Raman'] = 1888   #Adds Raman to the list
          print(scientists)
```

```
          {'Newton': 1643, 'Einstein': 1879, 'VonNeumann': 1903, 'Rama
          n': 1888}
```

More details of dictionaries

Accessing values through keys

```
In [30]:  print("The birth year of Einstein is ", scientists['Einstei
          n'])
```

```
          The birth year of Einstein is   1879
```

What if a key is not present in a dictionary?

```
In [31]:  try:
              print(scientists['Madonna'])
          except KeyError:
              print(' The key "Madonna" is not in the dictionary')
```

```
           The key "Madonna" is not in the dictionary
```

```
In [32]:  #Checks if a key is in a given dictionary
          print('Einstein' in scientists)
          print('Madonna' in scientists)
```

```
          True
          False
```

Keys need not be strings!

Any immutable object such as int, float, complex, str, tuple can be used as a key. Consider as
an example a quadratic polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2$$

We can store the coefficients of the polynomial as a dictionary.

```
In [33]:  quad = {0:-1, 1: 1, 2:3}
          print(quad[0] + quad[1]*5 + quad[2]*(5**2)) #Evaluates p(5)
```

```
          79
```

Be careful of integers vs floating point numbers.

```
In [34]:  D = {1:5, 1.:2.0}
          print(D)
          print(D[1.])
```

```
{1: 2.0}
2.0
```

Summary of Useful commands

```
In [35]: a = {}                    #Empty dictionary
         #Dictionaries can hold different data types
         a = {'list': [0,1], 'tuple': (0,1), 'dict': {}}
```