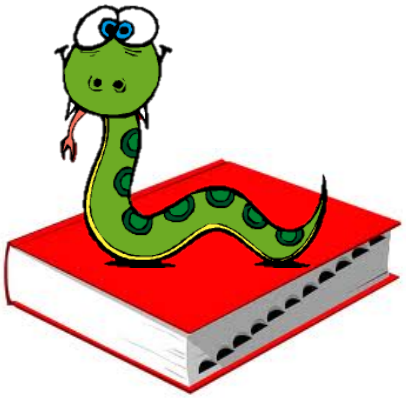
IDCE 302: Chapter 10

Dictionaries



Outline

- What is a dictionary?
- How to create a dictionary?
- Dictionary operations
- Represent a sparse matrix using a dictionary
- One more thing: Global variable

Compound Types Learned So Far

- `strings` `"Hello World"`
- `lists` `["Hello", "World"]`
- `tuples` `("Hello", "World")`

Now, we are going to explore **`Dictionaries`**

What is a dictionary?

- A **dictionary** is a list of key : value pairs

```
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

- The order of the key-value pairs does not stay the same.
- In fact, the order of items in a dictionary is unpredictable.

```
>>> print eng2sp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Create a Dictionary

Method 1: create a dictionary by providing a list of key-value pairs

```
>>> eng2sp = {"one": "uno", "two": "dos", "three": "tres"}


>>> print eng2sp
{"one": "uno", "three": "tres", "two": "dos"}


>>> print eng2sp["two"]
"dos"

>>> dic = {1:"one", 2:"two", 1:"uno"}
# How does Python deal with it?
```

Method 2: Create an empty dictionary and then add elements

- When a key is absent from the dictionary, the key and its value are inserted.
- When the key already exists, its associated value is replaced.

```
>>> eng2sp      #The empty dictionary is denoted a pair of braces "{}":
= {}

>>> eng2sp["one"] = "uno"
>>> eng2sp["two"] = "dos"

>>> print eng2sp
{"one": "uno", "two": "dos"}
```

Quick Questions

- Do all the keys have to be in the same type?
- Do all the values have to be in the same type?
- Do a key and its associated value have to be in the same type?
- Does the order matter?

Dictionary Operations

- Create a dictionary called inventory

```
inventory = {'apples':430, 'bananas': 312, 'oranges': 525, 'pears': 217}
```

- Print its contents

```
>>> print inventory
{'pears': 217, 'apples': 430, 'oranges': 525, 'bananas': 312}
```

- Del the 'pears' element

```
>>> del inventory['pears']
```

- Print the updated dictionary

```
>>> print inventory
{'apples': 430, 'oranges': 525, 'bananas': 312}
```

- Check the length

```
# the len() works for string, list, tuples, and dictionaries
>>> len(inventory)
3
```

Dictionary Functions

All return a List

keys()

```
>>> inventory.keys()
['apples', 'oranges', 'bananas']
```

values()

```
>>> inventory.values()
[430,  525,  312]
```

items()

```
>>> inventory.items()
[('apples', 430), ('oranges', 525), ('bananas', 312)]
```

has_key()

```
>>> inventory.has_key('oranges')
True
```

Aliasing & Copying

- Aliasing is a shallow copy.
- Two variables refer to the same object

```
>>> sizes = {'S': 28, 'M': 32, 'L': 36, 'XL': 40}
>>> sizesNew = sizes   #similar to a list aliasing

>>> del sizesNew['XL']

>>> sizes
{'S': 28, 'M': 32, 'L': 36}
```

- copy() performs deep copy.
- A separate new copy is created.

```
>>> sizes = {'S': 28, 'M': 32, 'L': 36, 'XL': 40}

>>> sizeNew = sizes.copy()

>>> del sizeNew['XL']

>>> sizeNew # modified
{'S': 28, 'M': 32, 'L': 36}

>>> sizes # it is not affected
{'S': 28, 'M': 32, 'L': 36, 'XL': 40}
```

Sparse Matrices

- A matrix can be represented using a list

```
matrix = [ [0,0,0,1,0],
           [0,0,0,0,0],
           [0,2,0,0,0],
           [0,0,0,0,0],
           [0,0,0,3,0] ]
```

- A matrix can also be represented using a dictionary.

```
matrix = {(0,3): 1, (2, 1): 2, (4, 3): 3}
```

- Especially for sparse matrices

- Use get() to get matrix elements from the dictionary

Function **get(key, default_val)**

Arguments:

1st: key

2nd: the return value if key does not exist

```
>>>matrix.get((0,3),0)   #the key can also be  a  tuple
1
>>>matrix.get((1,3),0)
0
```

Sample Code: *Use* ***`dictionary`*** *as a lookup table*

```
"""
input:
   n: an integer value
note:
   This is a recursive function
   to evaluate a Fabinacci number.
   The input must be an integer
"""

lookuptable={0: 1, 1: 1, 2: 2, 3: 3, 4: 5, 5: 8, 6: 13, 7: 21}

def fibonacci(n):
   if lookuptable.has_key(n):
      return lookuptable[n]
   else:
      return fibonacci(n-1) + fibonacci(n-2)

n=50

print 'Fibonacci (', n, ') is : ', fibonacci(n)
```

Output:

Fibonacci (50) is : 20365011074

***Sample Code:* Count letters in string**

```
def countLetters(string):
    # Initialize an empty dictionary
    letterCounts = {}
    for letter in string:
        # When looping over each character, add it as a key if it is
        # not in the dictionary yet and put 1 as its associated value.
        # If the character is already in the dic, update its value by
        # adding 1 to the current value.
        letterCounts[letter] = letterCounts.get(letter, 0) + 1
    return letterCounts

# Test case
mystr = "Mississippi"
print countLetters(mystr)

mystr = "Massachusets"
print countLetters(mystr)
```

Here are the outputs:

```
{'i': 4, 'p': 2, 's': 4, 'M': 1}
{'a': 2, 'c': 1, 'e': 1, 'h': 1, 'M': 1, 's': 4, 'u': 1, 't': 1}
```

***Sample Code:* Sort the dictionary**

```
mystr = "Massachusets"
mydic = countLetters(mystr)

print mydic
letterItems = mydic.items()   #items() returns a list of (keys and values) tuples

letterItems.sort()
print letterItems
```

Output:

```
{'a':2,'c':1, 'e':1,'h':1,'M':1,'s': 4,'u': 1,'t':1}
[('M',1),('a',2),('c',1),('e',1),('h',1),('s',4),('t',1),('u',1)]
```

- Items in a dictionary are not ordered in any way.
- Sorting a dictionary creates a list of tuples.

Traversing a Dictionary

- Loop through all the items (`keys` and `values`)

```
def printDictionary(dicInput):
    for (key, value) in dicInput.items():
      print key, value

legend = {0: "no value", 1: "deciduous", 2: "conifers", \
            3: "industrial", 4: "residential", \
            5: "water bodies", 6: "agricultural"}
printDictionary(legend)
```

Output:

```
0 no value
1 deciduous
2 conifers
3 industrial
4 residential
5 water bodies
6 agricultural
```

- Loop through all the keys

```
def printKeys(dicInput):
    for key in dicInput.keys():
      print key

legend = {0: "no value", 1: "deciduous", 2: "conifers", \
            3: "industrial", 4: "residential", \
            5: "water bodies", 6: "agricultural"}
printKeys(legend)
```

Output:

```
0
1
2
3
4
5
6
```

- Loop through all the values

```
def printValues(dicInput):
    for value in dicInput.values():
      print value

legend = {0: "no value", 1: "deciduous", 2: "conifers", \
            3: "industrial", 4: "residential", \
            5: "water bodies", 6: "agricultural"}
printValues(legend)
```

Output:

```
no value
deciduous
conifers
industrial
residential
water bodies
agricultural
```

Global Variable

- Variables used outside of functions
- Can be accessed by any functions in the script file.

```
verbose = True

def exampleFunction():
        if verbose:
            print 'Running exampleFunction'
```

- <u>Be cautious when reassigning a global variable</u>

```
count = 0

def changeIt():
        count = 2
>>> changeIt()
>>> count
0
```

```
count = 0

def plusOne():
        count = count + 1
```

Error: local variable 'count' referenced before assignment

- Python assumes the variable is local when it is reused in a function.
- To let Python know, you have to declare it global.

```
Count = 0
def plusOne():
      global count
      count += 1

# count will increase by 1 every time plusOne() is called
and ran.
```

Summary

- Dictionary is a list of key:value pairs.
- Dictionary items are NOT in order.
- Dictionary items can be added or deleted or updated.
- Use get() to access the value to a key of interest.
- Sparse matrices are better represented by dictionaries.
- Declare the variable global if you reassign it in a function.