Ch09-1-Dictionaries

October 30, 2020

1 Dictionaries

http://openbookproject.net/thinkcs/python/english3e/dictionaries.html

1.1 Topics

- dictionary data types
- create and use dictionary
- dictionary methods and operations
- dictionary applications and problems

1.2 Dictionary

- another compound type/container like lists and tuples
- very useful data structure/container that can store data as lookup table
- Python's mapping type similar to `map` container, or associative arrays in C++ and other languages
- dictionaries maps keys (immutable type) to values of any type (heterogeneous)
- Python uses complex hash algorithm to index key for fast access
- starting from verion 3.6, python dict remembers the orders of the elements inserted

1.3 Creating dictionary objects

[1]:
```
eng2sp = {} # or
eng2sp1 = dict()
```

[2]:
```
print(eng2sp, eng2sp1)
```

```
{} {}
```

[3]:
```
eng2sp["One"] = "uno"
eng2sp["two"] = "dos"
eng2sp["three"] = "tres"
eng2sp[4] = "quatro"
eng2sp["five"] = "sinco"
```

[4]:
```
eng2sp
```

[4]:
```
{'One': 'uno', 'two': 'dos', 'three': 'tres', 4: 'quatro', 'five': 'sinco'}
```

[5]:
```
key = 'Five'
eng2sp[key] = 'Sinco'
```

[6]:
```
print(eng2sp)
```

```
{'One': 'uno', 'two': 'dos', 'three': 'tres', 4: 'quatro', 'five': 'sinco',
'Five': 'Sinco'}
```

[7]:
```
print(eng2sp['One'])
```

```
uno
```

[8]:
```
symbolNames = {'*':'asterick', '+':"plus", '-': 'minus'}
```

[9]:
```
print(eng2sp, symbolNames)
```

```
{'One': 'uno', 'two': 'dos', 'three': 'tres', 4: 'quatro', 'five': 'sinco',
'Five': 'Sinco'} {'*': 'asterick', '+': 'plus', '-': 'minus'}
```

[10]:
```
dict1 ={'one': 'uno', 'two': 'dos', 'three': 'tres', '4': 'quatro', 'five':␣
↪'sinco'}
```

[11]:
```
dict1
```

[11]:
```
{'one': 'uno', 'two': 'dos', 'three': 'tres', '4': 'quatro', 'five': 'sinco'}
```

1.4 Accessing values

- use index operator [‘key’]
- dict object is mutable

[12]:
```
one = 'One'
```

[13]:
```
eng2sp[one]
```

[13]:
```
'uno'
```

[14]:
```
eng2sp
```

[14]:
```
{'One': 'uno',
 'two': 'dos',
 'three': 'tres',
 4: 'quatro',
 'five': 'sinco',
 'Five': 'Sinco'}
```

[15]:
```
key = 'ten'
value = 'diez'
eng2sp[key] = value
print(eng2sp['ten'])
```

```
diez
```

[16]:
```
eng2sp['One'] = 'Uno'
```

[17]:
```
eng2sp
```

[17]:
```
{'One': 'Uno',
 'two': 'dos',
 'three': 'tres',
 4: 'quatro',
 'five': 'sinco',
 'Five': 'Sinco',
 'ten': 'diez'}
```

[16]:
```
eng2sp['One'] = ['uno']
```

[17]:
```
eng2sp['One'].append('Uno')
```

[18]:
```
eng2sp['One'].insert(0, 'UNO')
```

[19]:
```
print(eng2sp)
```

```
{'One': ['UNO', 'uno', 'Uno'], 'two': 'dos', 'three': 'tres', 4: 'quatro',
'five': 'sinco', 'Five': 'Sinco', 'ten': 'diez'}
```

[20]:
```
adict = {1: ['uno', 'one'], 2:('two', 'dos'), 3:{'three':'tres'}}
```

[21]:
```
print(adict[2][1])
```

```
dos
```

[22]:
```
# How do you access tres in adict?
print(adict[3]['three'])
```

```
tres
```

1.5 Dictionary methods

[36]:
```
help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
```

```
|  dict() -> new empty dictionary
|  dict(mapping) -> new dictionary initialized from a mapping object's
|      (key, value) pairs
|  dict(iterable) -> new dictionary initialized as if via:
|      d = {}
|      for k, v in iterable:
|          d[k] = v
|  dict(**kwargs) -> new dictionary initialized with the name=value pairs
|      in the keyword argument list.  For example:  dict(one=1, two=2)
|
|  Methods defined here:
|
|  __contains__(self, key, /)
|      True if the dictionary has the specified key, else False.
|
|  __delitem__(self, key, /)
|      Delete self[key].
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(…)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
```

```
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(…)
 |      D.__sizeof__() -> size of D in memory, in bytes
 |
 |  clear(…)
 |      D.clear() -> None.  Remove all items from D.
 |
 |  copy(…)
 |      D.copy() -> a shallow copy of D
 |
 |  get(self, key, default=None, /)
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  items(…)
 |      D.items() -> a set-like object providing a view on D's items
 |
 |  keys(…)
 |      D.keys() -> a set-like object providing a view on D's keys
 |
 |  pop(…)
 |      D.pop(k[,d]) -> v, remove specified key and return the corresponding
value.
 |      If key is not found, d is returned if given, otherwise KeyError is
raised
 |
 |  popitem(…)
 |      D.popitem() -> (k, v), remove and return some (key, value) pair as a
 |      2-tuple; but raise KeyError if D is empty.
 |
 |  setdefault(self, key, default=None, /)
 |      Insert key with a value of default if key is not in the dictionary.
 |
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  update(…)
 |      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
 |      If E is present and has a .keys() method, then does:  for k in E: D[k] =
E[k]
 |      If E is present and lacks a .keys() method, then does:  for k, v in E:
D[k] = v
```

```
|      In either case, this is followed by: for k in F:  D[k] = F[k]
|
|  values(…)
|      D.values() -> an object providing a view on D's values
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  fromkeys(iterable, value=None, /) from builtins.type
|      Create a new dictionary with keys from iterable and values set to value.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

[37]:
```
for k in eng2sp.keys(): # the keys are ordered based on their insertion order
    print("key {} maps to value {}".format(k, eng2sp[k]))
```

```
key One maps to value ['UNO', 'uno', 0, 'Uno']
key two maps to value dos
key three maps to value tres
key 4 maps to value quatro
key five maps to value sinco
key Five maps to value Sinco
key ten maps to value diez
```

[23]:
```
print(list(eng2sp.keys()))
```

```
['One', 'two', 'three', 4, 'five', 'Five', 'ten']
```

[24]:
```
print(list(eng2sp.values()))
```

```
[['UNO', 'uno', 'Uno'], 'dos', 'tres', 'quatro', 'sinco', 'Sinco', 'diez']
```

[25]:
```
# iterate over keys
for k in eng2sp:
    print('key = {} value = {}'.format(k, eng2sp.get(k)))
```

```
key = One value = ['UNO', 'uno', 'Uno']
key = two value = dos
```

```
key = three value = tres
key = 4 value = quatro
key = five value = sinco
key = Five value = Sinco
key = ten value = diez
```

[26]:
```
# get method returns None if the key is not found
print(eng2sp.get('asdfsf'))
```

```
None
```

[28]:
```
# can also return default value if key doesn't exist
print(eng2sp.get("Ondfe", '?'))
```

```
?
```

[29]:
```
# iterate over values
for val in eng2sp.values():
    print("value = ", val)
```

```
value =  ['UNO', 'uno', 'Uno']
value =  dos
value =  tres
value =  quatro
value =  sinco
value =  Sinco
value =  diez
```

[30]:
```
values = list(eng2sp.values())
```

[31]:
```
values
```

[31]:
```
[['UNO', 'uno', 'Uno'], 'dos', 'tres', 'quatro', 'sinco', 'Sinco', 'diez']
```

[32]:
```
items = list(eng2sp.items())
```

[33]:
```
print(items)
```

```
[('One', ['UNO', 'uno', 'Uno']), ('two', 'dos'), ('three', 'tres'), (4,
'quatro'), ('five', 'sinco'), ('Five', 'Sinco'), ('ten', 'diez')]
```

[34]:
```
dict2 = dict(items)
print(dict2)
```

```
{'One': ['UNO', 'uno', 'Uno'], 'two': 'dos', 'three': 'tres', 4: 'quatro',
'five': 'sinco', 'Five': 'Sinco', 'ten': 'diez'}
```

[35]:
```
for k, v in eng2sp.items():
    print('{} -> {}'.format(k, v))
```

```
One -> ['UNO', 'uno', 'Uno']
two -> dos
three -> tres
4 -> quatro
five -> sinco
Five -> Sinco
ten -> diez
```

[36]:
```
print(eng2sp.popitem())
print(eng2sp)
```

```
('ten', 'diez')
{'One': ['UNO', 'uno', 'Uno'], 'two': 'dos', 'three': 'tres', 4: 'quatro',
'five': 'sinco', 'Five': 'Sinco'}
```

1.6 Checking keys

- in and not in operators can be used to check if some keys exist in a given dictionary
- knowing if key exists helps automatically create dictionaries and access corresponding values

[37]:
```
"One" in eng2sp
```

[37]: True

[38]:
```
"Ten" in eng2sp
```

[38]: False

[39]:
```
"twenty" not in eng2sp
```

[39]: True

1.7 Copying dictionary objects

- shallow copy vs deep copy

[ ]:
```
import copy
digits = {1: 'one', 2: 'two', 3: ['three', 'Three', 'THREE']}
digits1 = digits # creates an alias
digits2 = digits.copy() # shallow copy
digits3 = copy.deepcopy(digits) # deep copy
```

1.7.1 visualize in pythontutor.com

[ ]:
```
from IPython.display import IFrame
src = '''
http://pythontutor.com/iframe-embed.
 ↪html#code=import%20copy%0Adigits%20%3D%20%7B1%3A%20'one',%202%3A%20'two',%203%3A%20%5B'thre
 ↪copy%28%29%20%23%20shallow%20copy%0Adigits3%20%3D%20copy.
 ↪deepcopy%28digits%29%20%23%20deep%20copy&codeDivHeight=400&codeDivWidth=350&cumulative=fals
 ↪js&py=3&rawInputLstJSON=%5B%5D&textReferences=false
'''
IFrame(src, width=900, height=600)
```

1.8 Passing dictionaries to functions

- dict is a mutable type
- therefore, dict objects are passed by reference

[40]:
```
# find the histogram (frequency of each unique character) in a word
def histogram(word, hist):
    for c in word:
        c = c.lower()
        if c in hist:
            hist[c] += 1
        else:
            hist[c] = 1
```

[41]:
```
h = {}
histogram('Mississippim', h)
for k, v in h.items():
    print(k, v)
```

```
m 2
i 4
s 4
p 2
```

1.9 Returning dict from functions

- dict objects can be returned from functions

[42]:
```
def getHist(word):# = "Mississippi"
    h = {}
    for c in word:
        if c in h:
            h[c] += 1
        else:
            h[c] = 1
    return h
```

[43]:
```
hist = getHist('Mississippi')
print(hist)
if 'M' in hist:
    print('M is in histogram')
```

```
{'M': 1, 'i': 4, 's': 4, 'p': 2}
M is in histogram
```

1.10 Exercises

1. Count and print letter frequency in a given word. Hint: use get method
2. Write a program that reads some text data and prints a frequency table of the letters in alphabetical order. Case should be ignored. A sample output of the program when the user enters the data "ThiS is String with Upper and lower case Letters", would look this:

1.11 Kattis problems that can be solved using dict

1. I've Been Everywhere, Man - https://open.kattis.com/problems/everywhere
2. Seven Wonders - https://open.kattis.com/problems/sevenwonders
3. ACM Contest Scoring - https://open.kattis.com/problems/acm
4. Stacking Cups - https://open.kattis.com/problems/cups
5. A New Alphabet - https://open.kattis.com/problems/anewalphabet
6. Words for Numbers - https://open.kattis.com/problems/wordsfornumbers
7. Babelfish - https://open.kattis.com/problems/babelfish
8. Popular Vote - https://open.kattis.com/problems/vote
9. Adding Words - https://open.kattis.com/problems/addingwords
10. Grandpa Bernie - https://open.kattis.com/problems/grandpabernie
11. Judging Troubles - https://open.kattis.com/problems/judging
12. Not Amused - https://open.kattis.com/problems/notamused
13. Engineering English - https://open.kattis.com/problems/engineeringenglish
14. Hardwood Species - https://open.kattis.com/problems/hardwoodspecies
15. Conformity - https://open.kattis.com/problems/conformity
16. Galactic Collegiate Programming Contest - https://open.kattis.com/problems/gcpc
17. Simplicity - https://open.kattis.com/problems/simplicity

[ ]:
```

```