Python核心数据类型-字典

一.字典概述

相对于前面学的有序集合列表, 字典是无序的集合, 差别在于字典的值是通过键的索引的形式来获取的, 而不是通过偏移来获取. 字典的数据格式为key:value

字典的主要特点:

1. 通过键而不是通过偏移来读取: 字典通过键将一系列的值联系起来, 这样就可以使用键从字典中取出其值, 而不是像列表那样使用相对偏移.
2. 任意对象的无序集合: 与前面讲的列表不同, 存储在字典中的项没有明确的顺序, 键提供了字典中元素的象征性位置(key)而不是物理位置(偏移)
3. 无序, 序列运算无效: 与列表不同, 保存在字典中的项没特定的顺序, 字典是映射机制而不是序列, 字典元素之间没有顺序的概念, 所以类似 + 切片这些操作在字典中是无法操作的
4. 变长 异构 任意嵌套: 与列表类似, 字典可以在原处增长或缩短, 可以包含任意类型的对象, 支持任意深度的嵌套
5. 字典属于可变类型映射

In [1]:
```
#示例代码1. 通过索引key来获其value
x = {"a": "apple"}
print x["a"]
```

```
apple
```

In [2]:
```
#示例代码2. 变长 异构 任意嵌套
x = {'a': [1, 2, 3, 4], "b": {'x': 1, "y": {"x": 1, "y": 2}}}
print x
```

```
{'a': [1, 2, 3, 4], 'b': {'y': {'y': 2, 'x': 1}, 'x': 1}}
```

In [3]:
```
#示例代码3. 字典属于可变类型
x = {"x": 1, "y": 2}
print x
x["x"] = "apple"
print x
```

```
{'y': 2, 'x': 1}
{'y': 2, 'x': 'apple'}
```

二.字典常用方法

In [1]:
```
x = {"a": 1, "b": 2, "c": 3, "d": 4}
print(dir(x))
```

```
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__h
ash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '_
_setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

In [13]:
```
#1.keys 返回字典的所有key组成的列表
help(x.keys)
print x.keys()
```

```
Help on built-in function keys:

keys(...)
    D.keys() -> list of D's keys

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

In [14]:
```
#2.values返回字典所有的value组成的列表
help(x.values)
print(x.values())
```

```
Help on built-in function values:

values(...)
    D.values() -> list of D's values

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

In [15]:
```
#3.items返回字典中的键值对形成的元组组成的列表
help(x.items)
print x.items()
```

```
Help on built-in function items:

items(...)
    D.items() -> list of D's (key, value) pairs, as 2-tuples

[('a', 1), ('c', 3), ('b', 2), ('d', 4)]
```

In [16]:
```
#4.iterkeys返回字典的所有key组成的迭代器
help(x.iterkeys)
print x.iterkeys()
```

```
Help on built-in function iterkeys:

iterkeys(...)
    D.iterkeys() -> an iterator over the keys of D

<dictionary-keyiterator object at 0x7ffaecdfe260>
```

In [17]:
```
#5.itervalues返回字典的所有value组成的迭代器
help(x.itervalues)
print x.itervalues()
```

```
Help on built-in function itervalues:

itervalues(...)
    D.itervalues() -> an iterator over the values of D

<dictionary-valueiterator object at 0x7ffaecdfe368>
```

In [18]:
```
#6.iteritems()返回字典中的键值对形成的元组组成的迭代器
help(x.iteritems)
print x.iteritems()
```

```
Help on built-in function iteritems:

iteritems(...)
    D.iteritems() -> an iterator over the (key, value) items of D

<dictionary-itemiterator object at 0x7ffaecdfe3c0>
```

In [22]:
```
#7.pop从字典中删除指定的key, 返回该key的值
help(x.pop)
x.pop('a')
```

```
Help on built-in function pop:

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


1
```

In [24]:
```
#8.popitem 从字典中随机删除一对键值对, 返回其所组成的元组
help(x.popitem)
x.popitem()
```

```
Help on built-in function popitem:

popitem(...)
    D.popitem() -> (k, v), remove and return some (key, value) pair as a
    2-tuple; but raise KeyError if D is empty.


('c', 3)
```

In [28]:
```
#9.viewkeys 返回一个view对象, 优点是如果字典key发生了变化, view也会同步发生变化, 而普通的keys是不会发生变化的
x = {"a": 1, "b": 2, "c": 3, "d": 4}
print x
x1 = x.keys()
print x1, 'this is x1'
x2 = x.viewkeys()
print x2, 'this is x2'
x.pop('b')
print x1, 'after pop'
print x2, 'after pop'
```

```
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
['a', 'c', 'b', 'd'] this is x1
dict_keys(['a', 'c', 'b', 'd']) this is x2
['a', 'c', 'b', 'd'] after pop
dict_keys(['a', 'c', 'd']) after pop
```

In [29]:
```
#10.viewvalues返回一个view对象, 优点是如果字典value发生了变化, view也会同步发生变化, 而普通的values是不会发生变化的
x = {"a": 1, "b": 2, "c": 3, "d": 4}
print x
x1 = x.values()
print x1, 'this is x1'
x2 = x.viewvalues()
print x2, 'this is x2'
x.pop('b')
print x1, 'after pop'
print x2, 'after pop'
```

```
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
[1, 3, 2, 4] this is x1
dict_values([1, 3, 2, 4]) this is x2
[1, 3, 2, 4] after pop
dict_values([1, 3, 4]) after pop
```

In [30]:
```
#11.viewitems返回一个view对象, 优点是如果字典items发生了变化, view也会同步发生变化, 而普通的items是不会发生变化的
x = {"a": 1, "b": 2, "c": 3, "d": 4}
print x
x1 = x.items()
print x1, 'this is x1'
x2 = x.viewitems()
print x2, 'this is x2'
x.pop('b')
print x1, 'after pop'
print x2, 'after pop'
```

```
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
[('a', 1), ('c', 3), ('b', 2), ('d', 4)] this is x1
dict_items([('a', 1), ('c', 3), ('b', 2), ('d', 4)]) this is x2
[('a', 1), ('c', 3), ('b', 2), ('d', 4)] after pop
dict_items([('a', 1), ('c', 3), ('d', 4)]) after pop
```

In [54]:
```
#12.get获取字典key对应的value, 如果key不存在则返回你所指定的value默认值, 但不修改原字典
x = {"a": 1}
help(x.get)
print x.get("a", 0)
print x.get("b", 10)
print x
```

```
Help on built-in function get:

get(...)
    D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.

1
10
{'a': 1}
```

In [55]:
```
#13.setdefault 获取字典key对应的value, 如果key不存在则返回你所指定的value默认值, 但会修改原字典
x = {"a": 1}
help(x.get)
print x.setdefault("a", 0)
print x.setdefault("b", 10)
print x
```

```
Help on built-in function get:

get(...)
    D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.

1
10
{'a': 1, 'b': 10}
```

In [48]:
```
#14.has_key判断字典是否包含指定的key
help(x.has_key)
print x
print x.has_key('b')
print x.has_key('e')
```

```
Help on built-in function has_key:

has_key(...)
    D.has_key(k) -> True if D has a key k, else False

{'a': 1, 'c': 3, 'b': 2, 'd': 4}
True
False
```

In [43]:
```
#15.copy对字典进行一次浅拷贝
x = {"a": 1, "b": 2, "c": 3, "d": 4}
y = x.copy()
print x, id(x)
print y, id(y)
```

```
{'a': 1, 'c': 3, 'b': 2, 'd': 4} 140715692638288
{'a': 1, 'c': 3, 'b': 2, 'd': 4} 140715692654672
```

In [50]:
```
#16.fromkeys当字典的所有key拥有相同的value时, 可以用来快速创建字典
help(x.fromkeys)
y = {}.fromkeys(['x', 'y', 'z'], 10)
print y
```

```
Help on built-in function fromkeys:

fromkeys(...)
    dict.fromkeys(S[,v]) -> New dict with keys from S and values equal to v.
    v defaults to None.

{'y': 10, 'x': 10, 'z': 10}
```

In [39]:
```
#17.clear清空字典中的所有键值对
x = {"a": 1, "b": 2, "c": 3, "d": 4}
help(x.clear)
x.clear()
print x
```

```
Help on built-in function clear:

clear(...)
    D.clear() -> None.  Remove all items from D.

{}
```

In [52]:
```
#18.update将一个字典中的键值对,添加到另一个字典中
help(x.update)
x = {'x': 1, 'y': 2}
y = {"a": 10, "b": 20}
x.update(y)
print x
```

```
Help on built-in function update:

update(...)
    D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
    If E present and has a .keys() method, does:     for k in E: D[k] = E[k]
    If E present and lacks .keys() method, does:     for (k, v) in E: D[k] = v
    In either case, this is followed by: for k in F: D[k] = F[k]

{'y': 2, 'x': 1, 'b': 20, 'a': 10}
```

三.字典用法注意事项

1. 序列运算无效: 字典是映射机制 不是序列 字典的元素之间没有顺序的概念
2. 新索引赋值会添加新的项目
3. key不一定是字符串, 任何不可变的类型其实都可以 数字 字符串 元组

In [56]:
```
#示例代码.
x = {}
x[1] = 'x'
x['x'] = 1
x[(1, 2, 3)] = 10
print x
```

```
{1: 'x', (1, 2, 3): 10, 'x': 1}
```

四.创建字典的方法

In [57]:
```
#1.知道键值对直接创建字典
{"x": 1, "y": 2, "z": 3}

#2.动态创建
x = {}
x["a"] = 1
x["b"] = 2
print x
```

```
{'a': 1, 'b': 2}
```

In [58]:
```
#3.使用dict内置函数
dict(x = 1, y= 2, z=3)
```

Out[58]:
```
{'x': 1, 'y': 2, 'z': 3}
```

In [59]:
```
#4.dict函数接收[("x", 1),("y", 2),("z", 3)]结构的数据构成字典
l = ["a", "b", "c"]
w = [1, 2, 3]
dict(zip(l, w))
```

Out[59]:
```
{'a': 1, 'b': 2, 'c': 3}
```

五. 如何避免key-error错误

尝试获取一个字典中不存在的key的值时将报 key-error错误

In [60]:
```
#示例代码1.
x = {'age': 45, 'name': 'liu'}
x["job"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-60-742cd200743a> in <module>()
      1 #示例代码1.
      2 x = {'age': 45, 'name': 'liu'}
----> 3 x["job"]

KeyError: 'job'
```

In [62]:
```
#列举三种方法防止key-error错误
#方法1.
if 'job' in x:
    print x["job"]
else:
    print 'not has key job'
```

```
not has key job
```

In [63]:
```
#方法2.
try:
    print x['job']
except:
    print 'not has key job'
```

```
not has key job
```

In [64]:
```
#方法3.
print x.get("job", 1)
print x.setdefault('a', 10)
```

```
1
10
```

In [ ]:
```

```