量化金融 python 基础 4-Dictionary

1 Data Structure 数据结构 - Dictionary 字典

字典是一种可变的数据结构，用于存储键值对 Key-Value。
结构类似于这种 Dict {Key: Value}
其中，键 Key 是唯一的，必须是不可变的类型，例如字符串、数字或元组等,
而值 Value 可以是任何类型的数据, 包括列表、元组、字典等等。

[1]:
```
# example for dictionary
# 每个大宗商品的期货合约都有对应的合约交易单位，即一手合约对应多少量的标的物，可用字典来存储期货合约的对应交易单位。
# 大商所
contract_unit1 = {
    "c2309": 10, # 玉米 23 年 9 月到期的期货合约。
    "m2309": 10, # 豆粕
    "lh2309": 16, # 生猪
    "pp2309": 5, # 聚丙烯
    "i2309": 100, # 铁矿石
}
# 郑商所
contract_unit2 = {
    "AP310": 10, # 苹果 23 年 9 月到期的期货合约
    "CF309": 5, # 棉花
    "SA309": 20, # 纯碱
}

contract_unit1
```

[1]:
```
{'c2309': 10, 'm2309': 10, 'lh2309': 16, 'pp2309': 5, 'i2309': 100}
```

[2]:
```
# 访问字典内某键值
contract_unit1['c2309']
```

[2]: 10

[3]:
```
# 如果访问不存在的键 key, 会报错
contract_unit1['c2310']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[3], line 2
      1 # 如果访问不存在的键 key, 会报错
----> 2 contract_unit1['c2310']

KeyError: 'c2310'
```

[4]:
```
# 也可以通过 dict.get() 访问某键值,
# 这样做的好处是 即使字典内不存在相应的键 key, 也不会报错
# 而是返回第二个给定的参数, 也叫 default value.

print(contract_unit1.get("c2309"))
print(contract_unit1.get("c2305", "no such key"))
```

```
10
no such key
```

[5]:
```
# 使用 del 语句删除字典内某键值
del contract_unit1['c2309']
contract_unit1
```

[5]: {'m2309': 10, 'lh2309': 16, 'pp2309': 5, 'i2309': 100}

[6]:
```
# dict.pop() 方法删除某键值并返回该内容
m2309_trade_unit = contract_unit1.pop("m2309")
m2309_trade_unit
```

[6]: 10

[7]:
```
# 使用 pop 方法之后
contract_unit1
```

[7]:
```
{'lh2309': 16, 'pp2309': 5, 'i2309': 100}
```

[8]:
```
# 合并 contract_unit1 and 2
contract_unit_combined = contract_unit1.copy()
contract_unit_combined.update(contract_unit2)
contract_unit_combined
# 注意： 对于相同的 key，合并后 value 会被覆盖。
```

[8]:
```
{'lh2309': 16, 'pp2309': 5, 'i2309': 100, 'AP310': 10, 'CF309': 5, 'SA309': 20}
```

[9]:
```
# 通过 dict.keys() 遍历所有 key
for key in contract_unit1.keys():
    print(key)
```

```
lh2309
pp2309
i2309
```

[10]:
```
# 通过 dict.values() 遍历所有 value
for value in contract_unit1.values():
    print(value)
```

```
16
5
100
```

[11]:
```
# 通过 dict.items() 遍历所有的 key-value
for key, value in contract_unit1.items():
    print(key, value)
```

```
lh2309 16
pp2309 5
i2309 100
```

[12]:
```
# 将字典转换成 tuple


tuple(contract_unit1.items())
```

[12]: (('lh2309', 16), ('pp2309', 5), ('i2309', 100))

2 字典进阶应用

[13]:
```
# sorted with dict.values 根据字典 values 排序

# 进行这项任务之前先了解一下什么是 python lambda
"""
lambda 是 python 中用于创建匿名函数。

lambda arguments: expression
- arguments 可以是一个或者多个函数参数，
- expression 表示一个返回值的表达式

"""
# Example for lambda
# 返回一个平方数
square = lambda x: x*x

print(square(5))
```

25

[14]:
```
# continue sorted with dict.values 根据字典值 values 排序
"""
sorted 是一个排序函数，key 为一个函数，用于接收参数并返回用于排序的值

在这里 sorted key 接收了一个 lambda 函数，这个 lambda 函数用于返回对象的第二个值

字典在这里被当作一个元组 tuple，tuple 的第一个数为字典的键 key， 第二个值为字典的值 value

所以对象 contract_unit.items() 被传入到 sorted 函数内后，再从 lambda 返回字典的值 value 用于排序，

再整体输出排序完成的整个字典
"""
```

```
# 按字典 value 大小排序输出
print(sorted(contract_unit1.items(), key = lambda d: d[1]))

# 重新以字典形式输出
print(dict(sorted(contract_unit1.items(), key = lambda d: d[1])))
```

```
[('pp2309', 5), ('lh2309', 16), ('i2309', 100)]
{'pp2309': 5, 'lh2309': 16, 'i2309': 100}
```

[15]:
```
# 修改上面的 d[1] 成 d[0], 可以根据字典键 keys 进行排序
# 这里按字母大小顺序排序

dict(sorted(contract_unit1.items(), key = lambda d: d[0]))
```

[15]:
```
{'i2309': 100, 'lh2309': 16, 'pp2309': 5}
```

[16]:
```
# 从两个列表, 利用列表解析式创建字典 dict

contract_unit1_keys = list(contract_unit1.keys())
contract_unit1_values = list(contract_unit1.values())
print(contract_unit1_keys)
print(contract_unit1_values)
# 创建 dict
contract_unit1_duplicate = {v:k for v,k in zip(contract_unit1_keys,␣
 ↪contract_unit1_values)}
print(contract_unit1_duplicate)
```

```
['lh2309', 'pp2309', 'i2309']
[16, 5, 100]
{'lh2309': 16, 'pp2309': 5, 'i2309': 100}
```

[17]:
```
# example for dict.setdefault()
"""
假设某交易员当天交易记录如 trade_record 所示
trade_record 保存每次交易的品种名称和买卖手数, 正数为多头, 负数为空头
holding_position 字典汇总交易结束后的持仓
假设开盘前交易员持仓为零
```

```
创建一个名为 holding_position 的空字典，通过 for 循环遍历每笔交易，
如果对应品种名不在 holding_position 中，创建一个默认值为 0 的新键 key，
后续再进行加减对应交易量

如果不使用 dict.setdefault()， 由于 holding_position 一开始是个空字典，
默认情况下访问一个不存在的键 key 时，程序就会报错，所以这里使用 setdefault 使初始的
默认持仓为零。
"""

trade_record = (
    ("c2309", 2),
    ("m2309", 3),
    ("pp2309", 1),
    ("c2309", -4),
    ("pp2309", -1)
)

holding_position = {}
for i in trade_record:
    holding_position.setdefault(i[0], 0)
    holding_position[i[0]] += i[1]

holding_position
```

[17]:
```
{'c2309': -2, 'm2309': 3, 'pp2309': 0}
```

[18]:
```
# 除了 dict.setdefault 方法外，还有 defaultdict，
# 它是一个内置的子类，位于 collections module，它为不存在的键提供默认值。
# defaultdict(int) 默认初始值为 0，
# 如果要从 1 开始，则使用 defaultdict(lambda 1)

from collections import defaultdict

holding_position2 = defaultdict(int)
for i in trade_record:
    holding_position2[i[0]] += i[1]
```

```
holding_position2
```

[18]:
```
defaultdict(int, {'c2309': -2, 'm2309': 3, 'pp2309': 0})
```

最后还要提醒的是字典的键 key 必须是不可变的，

数字，布尔值 bool，字符 string，元组 tuple 是不可变的

列表 list，字典 dict，集合 set 是可变的

hash( ) 函数可以确定是否为不可变对象。

[19]:
```
dict1 = {
    (2, 4): 2,
    "hello": "world"
}
dict1
```

[19]:
```
{(2, 4): 2, 'hello': 'world'}
```

[20]:
```
# 可变的列表当 key 就会报错
dict2 = {
    [2, 4]: 2
}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 dict2 = {
      2     [2, 4]: 2
      3 }

TypeError: unhashable type: 'list'
```

[21]:
```
hash(33)
```

[21]:
```
33
```

[22]:
```
hash("hello")
```

[22]: 1673680036326662119

[23]:
```
hash([2, 4])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 hash([2, 4])

TypeError: unhashable type: 'list'
```

3 Summary

del dict[element]

dict_name.pop()

dict.update

dict.get()

dict.keys()

dict.values()

sorted(dict.values)

dict.items()

列表解析创建 dict - zip(key_list, value_list)

setdefault

defaultdict

字典的 key 不可变对象

hash()