Python Nested Dictionary

A dictionary can contain another dictionary, which in turn can contain dictionaries themselves, and so on to arbitrary depth. This is known as nested dictionary.

Nested dictionaries are one of many ways to represent structured information (similar to 'records' or 'structs' in other languages).

Create a Nested Dictionary

A nested dictionary is created the same way a normal dictionary is created. The only difference is that each value is another dictionary.

Let's build a dictionary that stores employee record.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}
```

The dict() Constructor

There are several ways to create a nested dictionary using a type constructor called dict().

To create a nested dictionary, simply pass dictionary key:value pair as keyword arguments to `dict()` Constructor.

```
D = dict(emp1 = {'name': 'Bob', 'job': 'Mgr'},
         emp2 = {'name': 'Kim', 'job': 'Dev'},
         emp3 = {'name': 'Sam', 'job': 'Dev'})


print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'},
#         'emp3': {'name': 'Sam', 'job': 'Dev'}}
```

You can use dict() function along with the zip() function, to combine separate lists of keys and values obtained dynamically at runtime.

```
IDs = ['emp1','emp2','emp3']


EmpInfo = [{'name': 'Bob', 'job': 'Mgr'},
           {'name': 'Kim', 'job': 'Dev'},
           {'name': 'Sam', 'job': 'Dev'}]


D = dict(zip(IDs, EmpInfo))


print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'},
#         'emp3': {'name': 'Sam', 'job': 'Dev'}}
```

You'll often want to create a dictionary with default values for each key. The fromkeys() method offers a way to do this.

```
IDs = ['emp1','emp2','emp3']
Defaults = {'name': '', 'job': ''}


D = dict.fromkeys(IDs, Defaults)


print(D)
# Prints {'emp1': {'name': '', 'job': ''},
#         'emp2': {'name': '', 'job': ''},
#         'emp3': {'name': '', 'job': ''}}
```

Access Nested Dictionary Items

You can access individual items in a nested dictionary by specifying key in multiple square brackets.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}


print(D['emp1']['name'])
# Prints Bob


print(D['emp2']['job'])
# Prints Dev
```

If you refer to a key that is not in the nested dictionary, an exception is raised.

```
print(D['emp1']['salary'])
# Triggers KeyError: 'salary'
```

To avoid such exception, you can use the special dictionary get() method. This method returns the value for key if key is in the dictionary, else `None`, so that this method never raises a `KeyError`.

```
# key present
print(D['emp1'].get('name'))
# Prints Bob


# key absent
print(D['emp1'].get('salary'))
# PrintsNone
```

Change Nested Dictionary Items

To change the value of a specific item in a nested dictionary, refer to its key.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}
```

```
D['emp3']['name'] = 'Max'
D['emp3']['job'] = 'Janitor'


print(D['emp3'])
# Prints {'name': 'Max', 'job': 'Janitor'}
```

Add or Update Nested Dictionary Items

Adding or updating nested dictionary items is easy. Just refer to the item by its key and assign a value. If the key is already present in the dictionary, its value is replaced by the new one.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}


D['emp3'] = {'name': 'Max', 'job': 'Janitor'}


print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'},
#         'emp3': {'name': 'Max', 'job': 'Janitor'}}
```

If the key is new, it is added to the dictionary with its value.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}


D['emp4'] = {'name': 'Max', 'job': 'Janitor'}


print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
```

```
#         'emp2': {'name': 'Kim', 'job': 'Dev'},
#         'emp3': {'name': 'Sam', 'job': 'Dev'},
#         'emp4': {'name': 'Max', 'job': 'Janitor'}}
```

Merge Two Nested Dictionaries

Use the built-in update() method to merge the keys and values of one nested dictionary into another. Note that this method blindly overwrites values of the same key if there's a clash.

```
D1 = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
      'emp2': {'name': 'Kim', 'job': 'Dev'}}


D2 = {'emp2': {'name': 'Sam', 'job': 'Dev'},
      'emp3': {'name': 'Max', 'job': 'Janitor'}}


D1.update(D2)


print(D1)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Sam', 'job': 'Dev'},
#         'emp3': {'name': 'Max', 'job': 'Janitor'}}
```

Here the 'emp2' record is updated while 'emp3' is added to the dictionary.

Remove Nested Dictionary Items

There are several ways to remove items from a nested dictionary.

Remove an Item by Key

If you know the key of the item you want, you can use pop() method. It removes the key and returns its value.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
```

```
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}

x = D.pop('emp3')

print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'}}

# get removed value
print(x)
# Prints {'name': 'Sam', 'job': 'Dev'}
```

If you don't need the removed value, use the `del` statement.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}

del D['emp3']

print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'}}
```

Remove Last Inserted Item

The popitem() method removes and returns the last inserted item as a tuple.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}

x = D.popitem()
```

```
print(D)
# Prints {'emp1': {'name': 'Bob', 'job': 'Mgr'},
#         'emp2': {'name': 'Kim', 'job': 'Dev'}}

# get removed pair
print(x)
# Prints ('emp3', {'name': 'Sam', 'job': 'Dev'})
```

In versions before 3.7, `popitem()` would remove a random item.

Iterate Through a Nested Dictionary

You can iterate over all values in a nested dictionary using nested for loop.

```
D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},
     'emp2': {'name': 'Kim', 'job': 'Dev'},
     'emp3': {'name': 'Sam', 'job': 'Dev'}}

for id, info in D.items():
    print("\nEmployee ID:", id)
    for key in info:
        print(key + ':', info[key])

# Prints Employee ID: emp1
#        name: Bob
#        job: Mgr

#        Employee ID: emp2
#        name: Kim
#        job: Dev
```

```
#         Employee ID: emp3
#         name: Sam
#         job: Dev
```