Python Dictionary Tutorial with Examples

Dictionary represents the **key-value** pair in Python, enclosed in curly braces. Keys are unique and a colon separates it from value, whereas comma separates the items. The keys are immutable (strings, numbers, etc.), whereas the value can be of any Python object. Apart from this, Keys in Dictionary are case-sensitive.

Note: You can easily relate Python Dictionaries with the real-world Dictionary, such as a word with its meaning as **key-value** pair.

How to create Python Dictionary

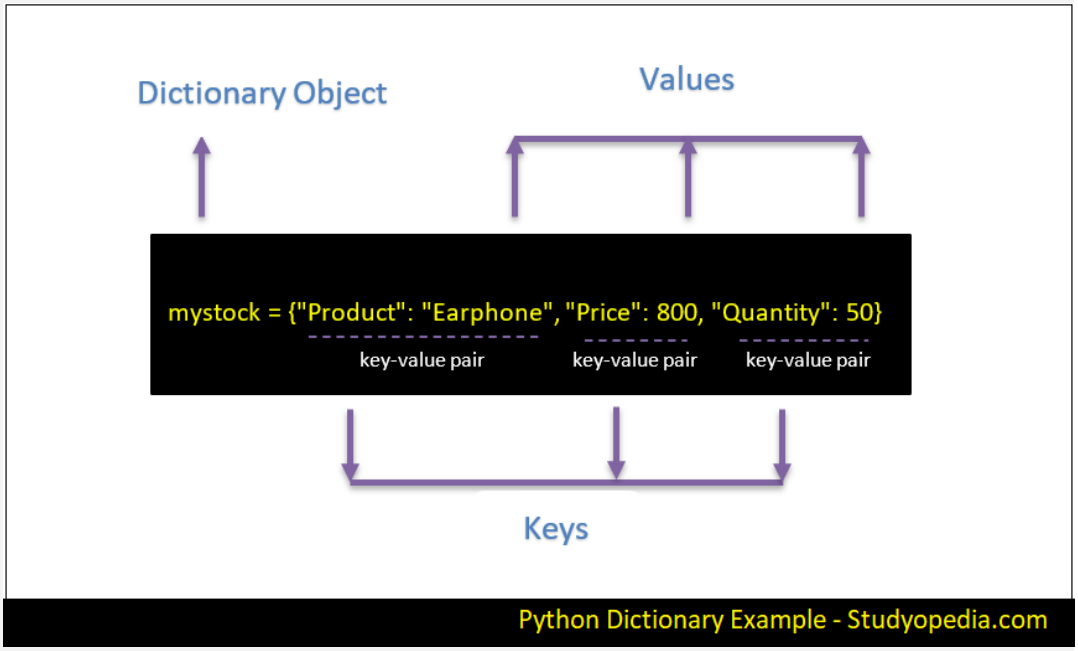
To create a Dictionary, set items separated by comma. By items, we meant the key-value pair, with a key and its value, for example:

```
mystock = {“Product”: “Earphone”, “Price”: 800, “Quantity”: 50}
```

Above, we have created a Dictionary object **mystock.** Moving further, on the left side before colon (:) is the key, whereas on the right, the corresponding value. Key can be a number, tuple or string. We have string as our Key. Furthermore, we have 3 key-value pairs inside the curly braces separated by comma. Let us now see the representation before moving further with a live example.

The **mystock** is our Dictionary object. The ”**Product**”, “**Price**” and “**Quantity**” are the string, which is an immutable object. Remember, the key cannot be repeated. The **“Product”:**

"Earphone" is one of the 3 key-value pairs in the above example:



Note: In the below examples, we have taken a Dictionary object **mystock**, and set Product details as key-value pairs.

Let us now see an example to create a Dictionary in Python with string keys:

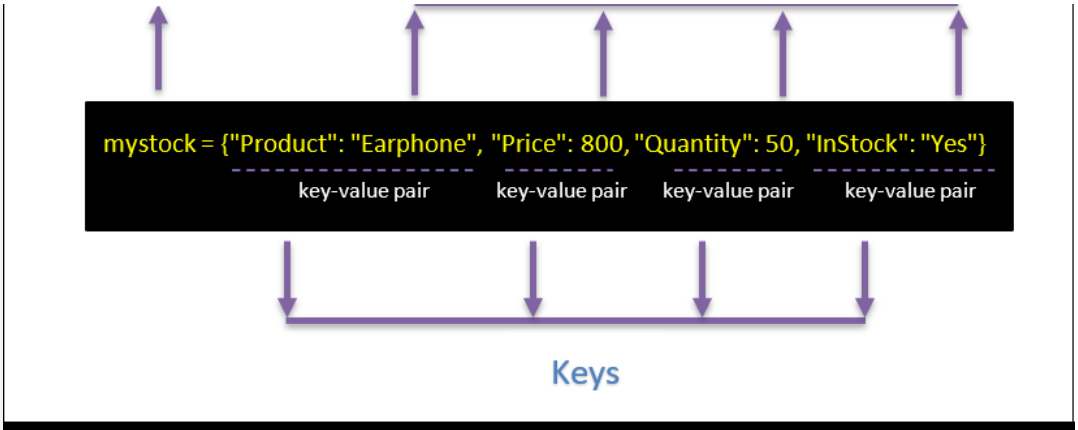
```
#Creating a Dictionary
mystock = {
  "Product": "Earphone",
  "Price": 800,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print(mystock)
```

The output is as follows:

```
{'Product': 'Earphone', 'Price': 800, 'Quantity': 50, 'InStock'
```

Following is the representation of the above example:



Python Dictionary with 4 key value pairs - Studyopedia.com

Let us now see another example wherein we will create Dictionary using the dict() method:

```
#Creating a Dictionary using dict()
mystock = dict ({
  "Product": "Earphone",
  "Price": 800,
  "Quantity": 50,
  "InStock" : "Yes"
})

#Printing the Dictionary
print(mystock)
```

The output is as follows:

```
{'Product': 'Earphone', 'Price': 800, 'Quantity': 50, 'InStock'
```

Let us now see how to create an empty Dictionary in Python:

```
#Creating a Dictionary using dict()
mystock = dict ({
  "Product": "Earphone",
  "Price": 800,
  "Quantity": 50,
  "InStock" : "Yes"
})

#Printing the Dictionary
print(mystock)

#Empty Dictionary
newstock = {}

#Printing empty Dictionary
print(newstock)
```

The output is as follows:

```
{'Product': 'Earphone', 'Price': 800, 'Quantity': 50, 'InStock'
{}
```

How to access values in Python Dictionary

To access values in Dictionary, use the key name. Therefore, refer the key under square brackets and fetch the corresponding value, for example:

```
mystock["Product"]
```

Above, will fetch the corresponding value of the key **"Product"**.

Let us now see an example and fetch specific values from keys:

```
#Creating a Dictionary
mystock = {
  "Product": "Earphone",
  "Price": 800,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Fetching value with key Product
print("Corresponding value for key Product = ",mystock["Product

#Fetching value with key Quantity
print("Corresponding value for key Quantity = ",mystock["Quanti
```

The output is as follows:

```
Corresponding value for key Product =  Earphone
Corresponding value for key Quantity =  50
Dictionary =  {'Product': 'Earphone', 'Price': 800, 'Quantity':
```

In the above code we displayed the corresponding values for key "**Product**" and "**Quantity**". You can achieve the same result with the **get()** method as in the below example:

```
mystock.get("Product")
```

As shown above, we are fetching the corresponding value for key "**Product**" using the get() method. Let us now see an

example to fetch corresponding value using get():

```
#Creating a Dictionary
mystock = {
  "Product": "Earphone",
  "Price": 800,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Fetching value with key Price
print("Corresponding value for key Product = ",mystock.get("Pri

#Fetching value with key InStock
print("Corresponding value for key Quantity = ",mystock.get("In
```

The output is as follows:

```
Dictionary =  {'Product': 'Earphone', 'Price': 800, 'Quantity':
Corresponding value for key Product =  800
Corresponding value for key Quantity =  Yes
```

Print all the keys of a Dictionary

To print the keys, use the keys() method of the Python Dictionary. Let us see an example to display only the keys of a Dictionary in Python:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
```

Print all the values of a Dictionary

To print the values, use the **values()** method of the Python Dictionary. Let us see an example to display only the values of a Dictionary in Python:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
```

Update values in a Dictionary

In Python, using the key, we can update the values in Dictionary. Refer the key name for which you want to update the value as in the below example:

```
mystock["Price"] = 4000
```

In the above code snippet, we have updated the corresponding value with key **“Price”** to **4000**. Previously, it was **3000**. Let us see an example wherein we have updated two values based on its keys:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
```

```
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Updating a specific value with key "Price"
mystock["Price"] = 4000;

#Updating a specific value with key "Quantity"
mystock["Quantity"] = 70;

print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Updated Values =  dict_values(['SSD', 4000, 70, 'Yes'])
Updated Dictionary =  {'Product': 'SSD', 'Price': 4000, 'Quanti
```

Adding items to a Dictionary

To add items to an already created Dictionary, create a new index key and assign value to it, for example:

```
mystock["Rating"] = 5
```

Let us now see an example and add items to an already created Dictionary:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())
```

```
print("Values = ",mystock.values())

#Calculate length
print("Length = ",len(mystock))

mystock["Rating"] = 5

#Updated keys
print("Updated Keys = ",mystock.keys())

#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)

#Calculate length
print("Updated Length = ",len(mystock))
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Length =  4
Updated Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InS
Updated Values =  dict_values(['SSD', 3000, 100, 'Yes', 5])
Updated Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quanti
Updated Length =  5
```

How to delete an element in a Dictionary with a specific key

To delete specific elements in a Dictionary, use the **del** keyword and in the square bracket, set the key of specific element you want to delete. Let's say you want to delete the element with key **Price**, then:

```
del dict['Price']
```

The above deletes the entry with key **Price.**

Let us now see an example and delete an element with a specific key:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
```

```
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Deleting the entry with key "Price"
del mystock["Price"];

#Updated keys
print("Updated Keys = ",mystock.keys())

#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Updated Keys =  dict_keys(['Product', 'Quantity', 'InStock'])
Updated Values =  dict_values(['SSD', 100, 'Yes'])
Updated Dictionary =  {'Product': 'SSD', 'Quantity': 100, 'InSt
```

Delete the Dictionary completely

To delete the Dictionary completely, use the **del** keyword in Python as in the below example:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Deleting the entry with key "Price"
del mystock["Price"];

#Updated keys
print("Updated Keys = ",mystock.keys())
```

```
#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)

#deleting complete Dictionary
del mystock

#error since we have deleted the Dictionary above
print(mystock);
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Updated Keys =  dict_keys(['Product', 'Quantity', 'InStock'])
Updated Values =  dict_values(['SSD', 100, 'Yes'])
Updated Dictionary =  {'Product': 'SSD', 'Quantity': 100, 'InSt
Traceback (most recent call last):
  File "./prog.py", line 34, in <module>
NameError: name 'mystock' is not defined
```

Empty the Dictionary

To empty the Dictionary in Python, the **clear()** method is provided by Python. Let us see an example:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Deleting the entry with key "Price"
del mystock["Price"];

#Updated keys
print("Updated Keys = ",mystock.keys())

#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)

#emptying the Dictionary
mystock.clear()
```

```
#Dictionay is empty now
print(mystock);
```

The code is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Updated Keys =  dict_keys(['Product', 'Quantity', 'InStock'])
Updated Values =  dict_values(['SSD', 100, 'Yes'])
Updated Dictionary =  {'Product': 'SSD', 'Quantity': 100, 'InSt
{}
```

Above, using the **clear()**, empties the Dictionary. On printing the empty Dictionary, only two curly brackets are visible.

Delete keys from the Dictionary

To delete a key, use the **del** keyword in Python Dictionary as in the below example:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Deleting the entry with key "InStock"
del mystock["InStock"];

#Updated keys
print("Updated Keys = ",mystock.keys())

#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
```

```
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Updated Keys =  dict_keys(['Product', 'Price', 'Quantity'])
Updated Values =  dict_values(['SSD', 3000, 100])
Updated Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quanti
```

Delete a key and return the corresponding value

To delete a key and return the corresponding value, use the pop() method. Let us see an example wherein we will be deleting a key and return the corresponding value:

```
#Creating a Dictionary
mystock = {
  "Product": "Headphone",
  "Price": 2500,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Deleting key
result = mystock.pop("Quantity")
print("Corresponding Value with popped key Quantity = ",str(resu

#Printing the updated Dictionary
print("Updated Dictionary (after deletion) = ",mystock)
```

The output is as follows:

```
Dictionary =  {'Product': 'Headphone', 'Price': 2500, 'Quantity
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['Headphone', 2500, 50, 'Yes'])
Corresponding Value with popped key Quantity =  50
Updated Dictionary (after deletion) =  {'Product': 'Headphone',
```

Iterate through a Dictionary

Use the for loop in Python to iterate through a Dictionary as in the below example:

```
#Creating a Dictionary
mystock = {
  "Product": "Headphone"
```

```
  "Price": 2500,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Iterating through the Dictionary
for p in mystock:
   print(mystock[p])
```

The output is as follows:

```
Dictionary =  {'Product': 'Headphone', 'Price': 2500, 'Quantity
Headphone
2500
50
Yes
```

Display a printable string representation of a Dictionary

To display a printable string representation of a Dictionary on Python, the **str()** method is used. Let us see an example:

```
#Creating a Dictionary
mystock = {
  "Product": "Headphone",
  "Price": 2500,
  "Quantity": 50,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Iterating through the Dictionary
for p in mystock:
    print(mystock[p])

#Equivalent String
print ("String = %s" % str (mystock))
```

The output is as follows:

```
Dictionary =  {'Product': 'Headphone', 'Price': 2500, 'Quantity
Headphone
2500
50
Yes
String = {'Product': 'Headphone', 'Price': 2500, 'Quantity': 50
```

Get the length of the Dictionary

To get the total length of the Dictionary, the **len()** method is used as in the below example:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
}

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Calculate length
print("Length = ",len(mystock))

#Deleting the entry with key "InStock"
del mystock["InStock"];

#Updated keys
print("Updated Keys = ",mystock.keys())

#Updated values
print("Updated Values = ",mystock.values())

#Printing the updated Dictionary
print("Updated Dictionary = ",mystock)

#Calculate length
print("Updated Length = ",len(mystock))
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Length =  4
Updated Keys =  dict_keys(['Product', 'Price', 'Quantity'])
Updated Values =  dict_values(['SSD', 3000, 100])
Updated Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quanti
Updated Length =  3
```

Make a copy of a Dictionary

Use the **copy()** method in Python Dictionary to make a copy, for example, a copy of **mystock** Dictionary will get created like this:

```
mystock2 = mystock.copy()
```

Let us now see an example to make a copy of a Dictionary:

```
#Creating a Dictionary
mystock = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
  }

#Printing the Dictionary
print("Dictionary = ",mystock)

#Display keys
print("Keys = ",mystock.keys())

#Display values
print("Values = ",mystock.values())

#Calculate length
print("Length = ",len(mystock))

#make a copy of Dictionary1 (mystock)
mystock2 = mystock.copy()
print(mystock2)

#Dictionary2 (mystock2) keys
print("Dictionary2 Keys = ",mystock2.keys())

#Dictionary2 (mystock2) values
print("Dictionary2 Values = ",mystock2.values())

#Printing Dictionary2 (mystock2)
print("Dictionary2 = ",mystock2)

#Calculate length of Dictionary2 (mystock2)
print("Dictionary2 Length = ",len(mystock2))
```

The output is as follows:

```
Dictionary =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 100
Keys =  dict_keys(['Product', 'Price', 'Quantity', 'InStock'])
Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Length =  4
{'Product': 'SSD', 'Price': 3000, 'Quantity': 100, 'InStock': '
Dictionary2 Keys =  dict_keys(['Product', 'Price', 'Quantity',
Dictionary2 Values =  dict_values(['SSD', 3000, 100, 'Yes'])
Dictionary2 =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 10
Dictionary2 Length =  4
```

Create a Multi-dimensional Dictionary

To create a multi-dimensional Dictionary, at first create Dictionary objects.

```
#Creating a Dictionary object
mystock1 = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
  }

#Creating another Dictionary object
mystock2 = {
  "Product": "HDD",
  "Price": 2500,
  "Quantity": 50,
  "InStock" : "Yes"
  }
```

After that, create a multi-dimensional Dictionary and set the dictionaries as their value:

```
#Creating multi-dimensional Dictionary
products={1:mystock1,2:mystock2}
```

Let us now see an example:

```
#Creating a Dictionary
mystock1 = {
  "Product": "SSD",
  "Price": 3000,
  "Quantity": 100,
  "InStock" : "Yes"
  }

#Printing the Dictionary1
print("Dictionary2 = ",mystock1)

#Creating Dictionary2
mystock2 = {
  "Product": "HDD",
  "Price": 2500,
  "Quantity": 50,
  "InStock" : "Yes"
  }

#Printing the Dictionary2
print("Dictionary1 = ",mystock2)

#Creating multi-dimensional Dictionary
products={1:mystock1,2:mystock2}

#Printing the Multi-Dimensional Dictionary
print("Multi-dimensional Dictionary = ",products)
```

The output is as follows:

```
Dictionary1 =  {'Product': 'SSD', 'Price': 3000, 'Quantity': 10
Dictionary2 =  {'Product': 'HDD', 'Price': 2500, 'Quantity': 50
Multi-dimensional Dictionary =  {1: {'Product': 'SSD', 'Price':
```

Properties of Dictionary keys

Following defines what Dictionary keys rules are:

- Keys are unique and cannot be repeated.
- Dictionary Keys are case-sensitive.
- Keys are immutable.
- Key can be a number, tuple or string.

Create Nested Dictionaries

A nested Dictionary is a Dictionary that has many Dictionaries. Let us see an example, wherein we will create a Dictionary that has 2 Dictionaries:

```
#Creating Nested Dictionary
products = {
  "mystock1" : {
   "Product": "SSD",
   "Price": 3000,
   "Quantity": 100,
   "InStock" : "Yes"
  },
  "mystock2" : {
   "Product": "HDD",
   "Price": 2500,
   "Quantity": 50,
   "InStock" : "Yes"
  }
}

#Printing the Nested Dictionary
print("Nested Dictionary = ",products)
```

The output is as follows:

```
Nested Dictionary =  {'mystock1': {'Product': 'SSD', 'Price': 3
```

Let us now see an example of a Dictionary with 4 Dictionaries i.e. Nested Dictionary:

```
#Creating Nested Dictionary
products = {
  "mystock1" : {
   "Product": "SSD",
   "Price": 3000,
   "Quantity": 100,
   "InStock" : "Yes"
  },
  "mystock2" : {
```

```
    "Product": "HDD",
    "Price": 2500,
    "Quantity": 50,
    "InStock" : "Yes"
  },
  "mystock3" : {
    "Product": "Headphone",
    "Price": 2800,
    "Quantity": 40,
    "InStock" : "Yes"
  },
  "mystock4" : {
    "Product": "Earphone",
    "Price": 1500,
    "Quantity": 10,
    "InStock" : "No"
  }
}

#Printing the Nested Dictionary
print("Nested Dictionary = ",products)
```

The output is as follows;

```
Nested Dictionary =  {'mystock1': {'Product': 'SSD', 'Price': 3
```

Access elements from a Nested Dictionary

To access elements from a Nested Dictionary, use the square brackets:

```
print(products["mystock2"]["Product"])
```

Above code snippet fetches the value of key Product from **Dictionary2** i.e. **mystock2.** Let us now see an example with 4 Dictionaries i.e. Nested Dictionary:

```
#Creating Nested Dictionary
products = {
  "mystock1" : {
    "Product": "SSD",
    "Price": 3000,
    "Quantity": 100,
    "InStock" : "Yes"
  },
  "mystock2" : {
    "Product": "HDD",
    "Price": 2500,
    "Quantity": 50,
    "InStock" : "Yes"
  },
  "mystock3" : {
    "Product": "Headphone",
    "Price": 2800,
    "Quantity": 40
```

```
    "InStock" : "Yes"
  },
  "mystock4" : {
    "Product": "Earphone",
    "Price": 1500,
    "Quantity": 10,
    "InStock" : "No"
  }
}

#Printing the Nested Dictionary
print("Nested Dictionary = ",products)

#Fetching the value of key Product from Dictionary mystock2
print(products["mystock2"]["Product"])

#Fetching the value of key Quantity from Dictionary mystock4
print(products["mystock4"]["Quantity"])
```

The output is as follows:

```
Nested Dictionary =  {'mystock1': {'Product': 'SSD', 'Price': 3
HDD
10
```

In this tutorial, we learned about Python Dictionaries. Moreover, we saw how to create a Dictionary and perform operations on it with live running examples. Additionally, some illustration were also used to explain the concept thoroughly.