Python Dictionary Operations

Activities

Let's take forward our discussion on Python dictionaries. In this class, we will learn to retrieve items from Python dictionaries. We will first learn to find the number of items stored in a dictionary, then how to retrieve only keys, then only values and then how to retrieve both of them together as a pair.

Let's quickly create the `my_dict` dictionary again and continue this class from the **Activity 1: Dictionary Length** section.

A python dictionary is a collection of key-value pairs.

Key	Value
Front Camera	16 MP
Battery	3300 mAh
Processor	Qualcomm Snapdragon 845
Display	6.28 inches
RAM	6 GB
Rear Camera	16 MP + 20 MP
Price (INR)	28990
Fast Charge	True

Collectively, both a key and its corresponding value constitute an item stored in a dictionary.

A dictionary is created using curly brackets (`{}`). A key and its corresponding value are mapped using the colon (`:`) sign and each key-value pair is separated by a comma.

```
# Create a dictionary as a collection of key-value pairs cont
d1={ 'Front Camera' : '16 MP' , 'Battery' : 3300 ,
     'Display' : 6.28 ,

  }
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
my_dict = {'Front Camera' : '16 MP', # First key-value pair
           'Battery' : '3300 mAh', # Second key-value pair
           'Processor' : 'Qualcomm Snapdragon 845', # Third k
           'Display' : '6.28 inches', # Fourth key-value pair
           'RAM' : '6 GB', # Fifth key-value pair
           'Rear Camera' : '16 MP + 20 MP', # Sixth key-value
           'Price' : 28990, # Seventh key-value pair
           'Fast Charge' : True} # Eighth key-value pair

my_dict
```

```
{'Front Camera': '16 MP',
 'Battery': '3300 mAh',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP',
 'Price': 28990,
 'Fast Charge': True}
```

```
print(my_dict)
```

```
{'Front Camera': '16 MP', 'Battery': '3300 mAh', 'Processor': 'Qualcomm Snapdrag
```

Activity 1: Dictionary Length

The length of a dictionary or dictionary length is the number of key-value pairs contained in the dictionary. The `my_dict` has eight key-value pairs. So, the length of `my_dict` is 8.

To calculate the length of a dictionary (or number of key-value pairs contained in a dictionary), you can use the `len()` function.

```
#  Calculate the length of the 'my_dict' dictionary.
len(my_dict)
```

```
8
```

As you can see, the `len()` function when applied on `my_dict` returns a numeric value `8` signifying that the dictionary stored in the `my_dict` variable has `8` key-value pairs.

Activity 2: The `keys()` Function

To get a list of all the keys in a dictionary, you can use the `keys()` function using the `dictionary_name.keys()` method.

Refer to the code shown below.

```
#  Get the list of all the keys contained in 'my_dict' dictio
l=list(my_dict.keys())
```

```
l
```

```
['Front Camera',
 'Battery',
 'Processor',
 'Display',
 'RAM',
 'Rear Camera',
 'Price',
 'Fast Charge']
```

```
type(l)
```

```
list
```

```
type(my_dict)
```

```
dict
```

```
my_dict.keys()
```

```
dict_keys(['Front Camera', 'Battery', 'Processor', 'Display', 'RAM', 'Rear
Camera', 'Price', 'Fast Charge'])
```

```
Start coding or generate with AI.
```

```
my_dict.keys()
```

```
dict_keys(['Front Camera', 'Battery', 'Processor', 'Display', 'RAM', 'Rear Camera', 'Price', 'Fast Charge'])
```

```
my_dict['Battery']
```

```
'3300 mAh'
```

```
my_dict['Front Camera']
```

```
'16 MP'
```

```
a=list(my_dict.keys())
```

```
a
```

```
a[0]
```

As you can see, the `keys()` function has returned a list of all the keys contained in the `my_dict` dictionary.

Note that the list returned by the `keys()` function is not the same as a Python list. It behaves differently as compared to a conventional Python list.

You can also use a `for` loop to print all the keys of a dictionary one after the other.

```
# Get the list of all the keys contained in 'my_dict' diction
for i in my_dict:
  print(i)
```

```
Front Camera
Battery
Processor
Display
RAM
Rear Camera
Price
Fast Charge
```

```
for i in my_dict.keys():
  print(i)
```

```
Front Camera
Battery
Processor
```

```
Display
RAM
Rear Camera
Price
Fast Charge
```

```
my_dict['Front Camera']
```

Actually, even if you don't use the `keys()` function, you can retrieve all the keys from a Python dictionary using a `for` loop. Let's retrieve all the keys from the `my_dict` dictionary without using the `keys()` function.

```
# Retrieve all the keys present in 'my_dict' dictionary witho
```

Activity 3: Getting A Value For A Key^

Now let's learn how to retrieve a value for a key from a Python dictionary. There are two methods to do this. They are:

1. The square brackets method
2. The `get()` function

Method I: The Square Brackets

For a key, in a dictionary, you can retrieve its corresponding value by using the `dictionary_name[key]` method where `dictionary_name` is the variable storing a Python dictionary and `key` is the key stored in the dictionary.

```
# Print the corresponding values to each key contained in the
# Value for the first key.
print(my_dict['Front Camera'])
print(my_dict.get("Front Camera"))

# Value for the second key.
print(my_dict['Battery'])
print(my_dict.get("Battery"))

# Value for the third key.
```

```
# Value for the fourth key.


# Value for the fifth key.


# Value for the sixth key.


# Value for the seventh key.


# Value for the eighth key.
```

```
print(my_dict['Front Camera'])
print(my_dict.get("Front Camera"))
```

```
16 MP
16 MP
```

```
print(my_dict['Front Camera'])
print(my_dict.get("Front Camera"))
```

```
for i in my_dict.keys():
  print(my_dict[i])
```

```
16 MP
3300 mAh
Qualcomm Snapdragon 845
6.28 inches
6 GB
16 MP + 20 MP
28990
True
```

```
for i in my_dict.keys():
  print(my_dict.get(i))
```

```
16 MP
3300 mAh
Qualcomm Snapdragon 845
6.28 inches
6 GB
```

```
16 MP + 20 MP
28990
True
```

```
for i in my_dict.keys():
  print(i)
```

```
Front Camera
Battery
Processor
Display
RAM
Rear Camera
Price
Fast Charge
```

```
for i in my_dict.keys():
  print(my_dict.get(i))
```

Instead of retrieving all the values one-by-one, you can get them using a `for` loop along with keys enclosed in square brackets.

```
# Print all the values contained in the 'my_dict' dictionary
for i in my_dict.keys():
  print(my_dict.get(i))
```

```
for i in my_dict:
  print(my_dict.get(i))
```

```
for i in my_dict.keys():
  print(my_dict[i])
```

```
for i in my_dict:
  print(my_dict[i])
```

The `my_dict.keys()` code returns a list of all the keys in a dictionary. In this case the list of keys is

`dict_keys(['Front Camera', 'Battery', 'Processor', 'Display', 'RAM', 'Rear Camera', 'Price', 'Fast Charge'])`

- Next, `for` loop iterates through each key in the above list and stores them in the `key` variable one-by-one.

- Then, `my_dict[key]` returns a value for the key stored in the `key` variable.
- Finally, the `print()` function prints the value for each `key`, as you can see in the output.

You don't necessarily have to use the `keys()` function to retrieve all the values from a dictionary.

```
# Print all the values stored in the 'my_dict' dictionary usi
```

Note: In a way, the keys act like indices for a Python dictionary. Unlike a Python list, a Python dictionary does not have indices of their own because a dictionary is an **unordered** collection of items. An only ordered collection of items have indices. Also, since a Python dictionary does not have indices, it is best to use `for` loop over `while` loop to iterate through each item that is stored in a dictionary.

Method II: The `get()` Function

You can also use the `get()` function to get a value of a key by using the `dictionary_name.get(key)` method.

```
# Print the corresponding values to each key contained in the
# Value for the first key.


# Value for the second key.


# Value for the third key.


# Value for the fourth key.


# Value for the fifth key.


# Value for the sixth key.
```

```
# Value for the seventh key.


# Value for the eighth key.
```

The `my_dict.keys()` code returns a list of all the keys in a dictionary. In this case the list of keys is

`dict_keys(['Front Camera', 'Battery', 'Processor', 'Display', 'RAM', 'Rear Camera', 'Price', 'Fast Charge'])`

- Next, `for` loop iterates through each key in the above list and stores them in the `key` variable one-by-one.
- Then, `my_dict[key]` returns a value for the key stored in the `key` variable.
- Finally, the `print()` function prints the value for each `key`, as you can see in the output.

Similarly, you can also use the `get()` function with `for` loop to get the value for a key.

```
#  Print all the values contained in the 'my_dict' dictionary
```

In the code above, `for` loop iterates through each key in the list generated by the `keys()` function. Then, the `get()` function returns the value for a key. Finally, the `print()` function prints the corresponding value for each key.

Activity 4: The `values()` Function^^

Let's learn how to get the collection of all the values contained in a Python dictionary. You can do this by using the `values()` function.

Syntax: `dictionary_name.values()`

```
# Retrieve a collection of all the values stored in the 'my_d
my_dict.values()
```

```
dict_values(['16 MP', '3300 mAh', 'Qualcomm Snapdragon 845', '6.28 inches', '6 GB', '16 MP + 20 MP', 28990, True])
```

As you can see, the `values()` function has returned a list of all the values contained in the `my_dict` dictionary.

Note that the list returned by the `values()` function is not the same as a Python list. This list behaves differently as compared to a conventional Python list.

You can also use a `for` loop to print each item contained in the list returned by the `values()` function.

```
# Using 'for' loop, print each item contained in the list ret
for i in my_dict.values():
  print(i)
```

As you can see, in the `for` loop, the `value` variable iterates through every value contained in the list,

`dict_values(['16 MP', '3300 mAh', 'Qualcomm Snapdragon 845', '6.28 inches', '6 GB', '16 MP + 20 MP', 28990, True])`

Then, the `print()` function prints the value stored in the `value` variable.

Activity 5: The `items()` Function^^^

You can also retrieve all the key-value pairs from a dictionary as a collection of tuples by using the `items()` function.

Syntax: `dictionary_name.items()` method.

```
print(my_dict.items())
```

```
dict_items([('Front Camera', '16 MP'), ('Battery', '3300 mAh'), ('Processor', 'Q
```

As you can see, the `items()` function has returned a list of key-value pairs contained in the `my_dict` dictionary. Each key-value pair is a tuple because a pair is written inside the common brackets, i.e., `()`.

Note that the list returned by the `items()` function is not the same as a Python list. This list behaves differently as compared to a conventional Python list.

You can also use a `for` loop to print each item contained in the list returned by the `items()` function.

```
# Loop through each item stored in the list returned by the '
for i in my_dict.items():
  print(i)
```

```
('Front Camera', '16 MP')
('Battery', '3300 mAh')
('Processor', 'Qualcomm Snapdragon 845')
('Display', '6.28 inches')
('RAM', '6 GB')
('Rear Camera', '16 MP + 20 MP')
('Price', 28990)
('Fast Charge', True)
```

```
for i, j in my_dict.items():
  print(i,":", j)
```

```
Front Camera : 16 MP
Battery : 3300 mAh
Processor : Qualcomm Snapdragon 845
Display : 6.28 inches
RAM : 6 GB
Rear Camera : 16 MP + 20 MP
Price : 28990
Fast Charge : True
```

```
for i, j in my_dict.items():
  print(j,":",i )
```

As you can see, using `for` loop, we have retrieved every tuple containing a key-value pair, from the `my_dict` dictionary.

You can also simultaneously iterate through each key-value contained in the list returned by the `items()` function.

```
# Simultaneously iterate through each key-value pair containe
```

As you can see, in the `for` loop, the `key` variable iterates through each key in each item and the `value` variable iterates through each value in each item contained in the list,

`dict_items([('Front Camera', '16 MP'), ('Battery', '3300 mAh'), ('Processor', 'Qualcomm Snapdragon 845'), ('Display', '6.28 inches'), ('RAM', '6 GB'), ('Rear Camera', '16 MP + 20 MP'), ('Price', 28990), ('Fast Charge', True)])`

Then, the `print()` function prints the values stored in the `key` and `value` variables.

If you apply the same method of retrieving key-value pairs simultaneously directly from the `my_dict` dictionary, then you will get `ValueError`.

```
#  Simultaneously retrieve all the key-value pairs directly f
for i in my_dict:
  print(i," : ",  my_dict.get(i))
```

```
# value: key
```

Activity : Adding New Item Let's also add the storage capacity of the smartphone to the my_dict dictionary. So the new item is Memory and the corresponding value is '128 GB'.

To add a new item to a dictionary, you simply have to use the square brackets as shown in the syntax below.

Syntax: dictionary_name[key] = value

where dictionary_name is some variable storing a dictionary.

```
# Add 'Memory' and '128 GB' as keys and values respectively t
my_dict['Memory']='128 GB'
```

```
my_dict
```

```
{'Front Camera': '16 MP',
 'Battery': '3300 mAh',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP',
 'Price': 28990,
 'Fast Charge': True,
 'Memory': '128 GB'}
```

Activity : The `zip()` Function

Sometimes you may require to create a dictionary from two Python lists. Consider the two lists shown below.

```
car_companies = ['Maruti Suzuki', 'Honda', 'Hyundai', 'Ford', 'V
car_models = [
                ['Baleno', 'Vitara Brezza', 'Ciaz'],
                'City',
                ['Verna', 'Creta'],
                ['Figo', 'EcoSport', 'Aspire'],
                ['Polo', 'Tiguan', 'Vento']
              ]
```

The `car_companies` list contains the name of the car manufacturers. The `car_models` list contains the models of cars manufactured by them.

Let's create a dictionary which maps every car model to its corresponding manufacturer. Let's store it in the `cars_dict`.

To create a dictionary from two lists, use the `zip()` and `dict()` function using the following syntax.

Syntax: `dictionary_name = dict(zip(list1, list2))`

where the `zip()` function joins the lists `list1` and `list2` but the `dict()` function converts the resulting object into a dictionary.

```
# Create a new dictionary by joining the 'car_companies' & 'c
car_companies = ['Maruti Suzuki', 'Honda', 'Hyundai', 'Ford',
car_models = [
                ['Baleno', 'Vitara Brezza', 'Ciaz'],
                'City',
                ['Verna', 'Creta'],
                ['Figo', 'EcoSport', 'Aspire'],
                ['Polo', 'Tiguan', 'Vento']
              ]
```

```
cars=dict(zip(car_companies,car_models))
cars
```

```
cl=tuple(zip(car_companies,car_models))
cl
```

```
cln=list(zip(car_companies,car_models))
cln
```

As you can see, a new dictionary called `cars_dict` has been created using the `zip()` and `dict()` functions.

The `zip()` function maps items of the first list with the items of the second list such that former become **keys** and latter become the corresponding **values** of the keys.

Note that the first item contained in the first list becomes **key** for the first item in the second list and the second item in the first list becomes **key** for the second item in the second list and so on.

Instead of using the `dict()` function, if you use the `list()` function on top of the `zip()` function, you will get a Python list containing the items of the two lists as a collection of key-value pairs in tuples.

```
#  Create a new list by joining the 'car_companies' & 'car_mo
```

Activity: Removing Item From Dictionary

To remove an item from a dictionary, you can use the `pop()` function. Inside the `pop()` function, you have to provide the key for which the key-value pair must be removed from the dictionary.

Let's remove the item `'Battery' : '3300 mAh'` from the `my_dict` dictionary, using the `pop()` function.

```
# Remove the item 'Battery' : '3300 mAh' from the 'my_dict' d
my_dict.pop('Battery')
my_dict
```

```
{'Front Camera': '16 MP',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP',
 'Price': 28990,
```

```
 'Fast Charge': True,
 'Memory': '128 GB'}
```

As you can see, the `pop()` function has removed `'Battery' : '3300 mAh'` item from the `my_dict` dictionary.

Note: The `pop()` function for a Python dictionary is different from the `pop()` function in Python list. In the case of dictionary, it takes the key of a key-value pair to remove the pair.

Syntax: `python_dictionary.pop(key)`

There is another function called the `popitem()` to remove an item from a dictionary. However, it removes only the last item from a dictionary.

```
#  Remove the last item from the dictionary.
my_dict.popitem()
my_dict
```

```
{'Front Camera': '16 MP',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP'}
```

```
my_dict['Price']= 10000
my_dict
```

```
{'Front Camera': '16 MP',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP',
 'Price': 10000}
```

```
Start coding or generate with AI.
```

Activity 5: Update Dictionary Item

Let's say the price of the smartphone, whose specifications are listed in the `my_dict` dictionary, changes to INR 25000. So, the `'Price': 28990` item in the `my_dict` dictionary must be updated.

To update an item in a dictionary, you need to enclose the key for that item in square brackets followed by the equals to sign, followed by the updated value as shown in the

following syntax.

Syntax: `dictionary_name[key] = new_value`

```
# Update the price of the smartphone in the 'my_dict' diction
my_dict['Price']= 25000
my_dict
```

```
{'Front Camera': '16 MP',
 'Processor': 'Qualcomm Snapdragon 845',
 'Display': '6.28 inches',
 'RAM': '6 GB',
 'Rear Camera': '16 MP + 20 MP',
 'Price': 25000}
```

```
'RAM' not in my_dictq1`
```

```
False
```

```
type(my_dict)
```

```
dict
```

Activity : The `in` & `not in` Keywords^^^

To check whether a key exists in a dictionary or not, you can use the `in` keyword. If a key exists in the dictionary, then the `in` keyword will return `True`, else it will return `False`. Similarly, if a key exists in the dictionary, then the `not in` keyword will return `False`, else it will return `True`.

You have already used them in `for` loops many times.

Syntax: `key in dictionary_name`

Syntax: `key not in dictionary_name`

Let's find out whether the keys, `'Honda'` and `'Audi'` exist in the `cars_dict` or not.

```
# Check whether the item 'Honda' exists in the 'cars_dict' di
'Honda' in cars
```

```
# Check whether the item 'Audi' exists in the 'cars_dict' dic
'Ciaz' in cars
```

Activity : The `update()` Function

The `update()` function is applied on a dictionary and it's input is another dictionary wherein both the dictionary may or may not contain the same keys. The `update()` function performs two operations:

1. It updates the existing key-value pair in a dictionary if a key exists.
2. It adds the key-value pairs which do not exist in a dictionary.

Consider the two dictionaries:

```
cars_dict = {'Ford': ['Figo', 'EcoSport', 'Aspire'],
             'Honda': 'City',
             'Hyundai': ['Verna', 'Creta'],
             'Maruti Suzuki': ['Baleno', 'Vitara Brezza', 'Ciaz
             'Volkswagen': ['Polo', 'Tiguan', 'Vento']}

supercars = {"Ford": "Mustang GT",
            "Lamborghini": ["Huracan", "Aventador", "Elemento"]
            "Koenigsegg": ["Agera", "Gemera", "Regera"],
            "Bugatti": "Veyron"}
```

When the `update()` function is applied on the `cars_dict` dictionary with the `supercars` dictionary as input, it updates the value of the `Ford` key and adds the other key-value pairs of the `supercars` dictionary to the `car_dict` dictionary.

```
# Create the 'supercars' dictionary and update the 'cars_dict
cars_dict = {'Ford': ['Figo', 'EcoSport', 'Aspire'],
             'Honda': 'City',
             'Hyundai': ['Verna', 'Creta'],
             'Maruti Suzuki': ['Baleno', 'Vitara Brezza', 'C
             'Volkswagen': ['Polo', 'Tiguan', 'Vento']}

supercars = {"Ford": "Mustang GT",
             "Lamborghini": ["Huracan", "Aventador", "Element
             "Koenigsegg": ["Agera", "Gemera", "Regera"],
             "Bugatti": "Veyron"}
```

```
supercars.update(cars_dict)
supercars
```

```
cars_dict.update(supercars)
cars_dict
```

```
supercars['MG']='Hector'
```

Activity: The `clear()` Function

The `clear()` function simply clears or deletes all the key-values pairs from a dictionary.

```
my_dict
```

```
# Clear the contents of the dictionary.
my_dict.clear()
```