DICTIONARIES

- **Creation of New Dictionary**
  - You can define it like any other variable, just make sure to use curly braces `{}` and separate key-value pairs with colons `:`.

    ```
    # Creating a new dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
    ```

  - **l# Accessing values**

    ```
    lprint(new_dict['key1'])  # Output: value1
    ```

  - You can then use this `**new_dict**` variable throughout your Jupyter Notebook session.

- **Accessing Items in the Dictionary**
  - You can do this by directly referencing the key within square brackets `[ ]`.

    ```
    # Accessing items in the dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

    # Accessing values using keys
    print(new_dict['key1'])  # Output: value1
    print(new_dict['key2'])  # Output: value2
    print(new_dict['key3'])  # Output: value3
    ```

    If the key doesn't exist, it will raise a `KeyError`. So, make sure the key exists in the dictionary before accessing it.

- **Change Values in the Dictionary**
  - You can change the values associated with keys in a dictionary by simply assigning a new value to the corresponding key.

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
    ```

```
# Changing the value associated with 'key2'
new_dict['key2'] = 'new_value2'

# Displaying the updated dictionary
print(new_dict)
```

- **Loop Through a Dictionary Values**
  - To loop through the values of a dictionary in Python, you can use a `for` loop.

```
# Original dictionary
new_dict = {'key1': 'value1', 'key2': 'value2', 'key3':
'value3'}

# Looping through the values
for value in new_dict.values():
    print(value)
```

In this loop, `**new_dict.values()**` returns a view object that displays a list of all the values in the dictionary. The `**for**` loop then iterates over each value, allowing you to perform operations on them.

- **Check if Key Exists in the Dictionary**
  - To check if a key exists in a dictionary, you can use the in keyword. Here's how you can do it:

```
# Original dictionary
new_dict = {'key1': 'value1', 'key2': 'value2', 'key3':
'value3'}

# Check if a key exists
if 'key2' in new_dict:
 print('Key "key2" exists in the dictionary')
else:
 print('Key "key2" does not exist in the dictionary')
```

Alternatively, you can use the get() method of dictionaries, which returns None if the key is not found (or a specified default value if provided):

```
# Using the get() method
value = new_dict.get('key2')
```

```
if value is not None:
 print('Key "key2" exists in the dictionary')
else:
 print('Key "key2" does not exist in the dictionary')
```

- **Checking for Dictionary Length**
  - To check the length (number of key-value pairs) of a dictionary in Python, you can use the `len()` function.

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3':
    'value3'}

    # Get the length of the dictionary
    dictionary_length = len(new_dict)

    print("The dictionary length is:", dictionary_length)
    ```

    The dictionary length is: 3

    The `**len()**` function returns the number of items (key-value pairs) in the dictionary. In this case, it returns `3`, indicating that there are three items in the dictionary.

- **Adding Items in the Dictionary**
  - You can add items to a dictionary in Python by simply assigning a value to a new key, or by using the `update()` method to add multiple items from another dictionary. Here's how you can do it:

    Adding a single item:

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2'}

    # Adding a new key-value pair
    new_dict['key3'] = 'value3'

    print(new_dict)
    ```

Adding multiple items:

```
# Original dictionary
new_dict = {'key1': 'value1', 'key2': 'value2'}

# Another dictionary with items to add
additional_dict = {'key3': 'value3', 'key4': 'value4'}


# Adding items from another dictionary
new_dict.update(additional_dict)

print(new_dict)
```

- **Removing Items in the Dictionary**
  - You can remove items from a dictionary in Python using the `del` keyword to delete a specific key-value pair or using the `pop()` method to remove an item and return its value.

    Removing a specific item by key using `del`:

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3':
    'value3'}

    # Removing a specific key-value pair
    del new_dict['key2']

    print(new_dict)
    ```

    Removing a specific item by key using `pop()`:

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3':
    'value3'}

    # Removing and returning the value of a specific key
    value = new_dict.pop('key2')

    print("Removed value:", value)
    print("Updated dictionary:", new_dict)
    ```

    In both cases, the specified item is removed from the dictionary. However, using **`del`** does not return the

removed value, while `**pop()**` does. Choose the appropriate method based on whether you need to keep track of the removed value or not.

- **Remove an Item Using del Statement**
  - removing an item from a dictionary using the `del` statement:

    ```
    # Original dictionary
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

    # Removing a specific key-value pair using del
    del new_dict['key2']

    # Displaying the updated dictionary
    print(new_dict)
    ```

    In this example, the key `'key2'` along with its corresponding value `'value2'` is removed from the dictionary `new_dict` using the `del` statement.

- **The dict() Constructor**
  - The `dict()` constructor in Python is used to create a new dictionary object. It can be called without arguments to create an empty dictionary or with arguments to create a dictionary with initial key-value pairs.

    Creating an empty dictionary:

    ```
    empty_dict = dict()
    # Or equivalently:
    empty_dict = {}
    ```

    Creating a dictionary with initial key-value pairs:

    ```
    new_dict = dict(key1='value1', key2='value2', key3='value3')
    # Or equivalently:
    new_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
    ```

    Creating a dictionary from an iterable of key-value pairs:

    ```
    key_value_pairs = [('key1', 'value1'), ('key2', 'value2'), ('key3', 'value3')]
    ```

```
constructed_dict = dict(key_value_pairs)
```

Creating a dictionary from two lists, one for keys and one for values:

```
keys = ['key1', 'key2', 'key3']
values = ['value1', 'value2', 'value3']
constructed_dict = dict(zip(keys, values))
```

Copying a dictionary:

```
original_dict = {'key1': 'value1', 'key2': 'value2'}
copied_dict = dict(original_dict)
```

The `dict()` constructor provides a flexible way to create dictionaries in Python, allowing you to construct dictionaries from various data types and representations.

- **Dictionary Methods**
  - Here are some common methods that you can use with dictionaries in Python:

    dict.keys(): returns a view object that displays a list of all the keys in the dictionary.

    ```
    my_dict = {'a': 1, 'b': 2, 'c': 3}
    keys = my_dict.keys()
    print(keys)  # Output: dict_keys(['a', 'b', 'c'])
    ```

    dict.values(): returns a view object that displays a list of all the values in the dictionary.

    ```
    my_dict = {'a': 1, 'b': 2, 'c': 3}
    values = my_dict.values()
    print(values)  # Output: dict_values([1, 2, 3])
    ```

    dict.items(): returns a view object that displays a list of tuples, where each tuple contains a key-value pair.

    ```
    my_dict = {'a': 1, 'b': 2, 'c': 3}
    items = my_dict.items()
    print(items)  # Output: dict_items([('a', 1), ('b', 2), ('c', 3)])
    ```

dict.get(key[, default]): returns the value for the specified key in the dictionary. If the key is not found, it returns the default value or `None` if not provided.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
value_a = my_dict.get('a')
value_d = my_dict.get('d', 'Key not found')
print(value_a)  # Output: 1
print(value_d)  # Output: Key not found
```

dict.update(other_dict): updates the dictionary with elements from another dictionary or from an iterable of key-value pairs.

```
my_dict = {'a': 1, 'b': 2}
other_dict = {'b': 3, 'c': 4}
my_dict.update(other_dict)
print(my_dict)  # Output: {'a': 1, 'b': 3, 'c': 4}
```

Each method provides different functionality for working with dictionaries efficiently.