In Python, a list is a versatile and widely used data structure that allows you to store and manipulate a collection of items. Lists are mutable, meaning you can modify their elements by adding, removing, or updating values. Each element in a list can be of any data type, and a list can contain a mix of different data types.

Here's a basic example of a Python list:

```python
my_list = [1, 2, 3, 'four', 5.0]
```

In this example, `my_list` contains integers, a string, and a floating-point number.

Python provides several built-in functions for working with lists. Some commonly used list functions include:

`len()`: Returns the number of elements in a list.

```python
length = len(my_list)
```

`append()`: Adds an element to the end of the list.

```python
my_list.append(6)
```

`insert()`: Inserts an element at a specified position in the list.

```python
my_list.insert(2, 'inserted')
```

`remove()`: Removes the first occurrence of a specified value from the list.

```python
my_list.remove('four')
```

`pop()`: Removes and returns the element at a specified index. If no index is provided, it removes and returns the last element.

```python
popped_element = my_list.pop(3)
```

`index()`: Returns the index of the first occurrence of a specified value in the list.

```python
index_of_five = my_list.index(5.0)
```

`count()`: Returns the number of occurrences of a specified value in the list.

```python
number_of_twos = my_list.count(2)
```

`sort()`: Sorts the elements of the list in ascending order. You can use the **`reverse`** parameter to sort in descending order.

```python
my_list.sort()
```

`reverse()`: Reverses the order of the elements in the list.

```python
my_list.reverse()
```

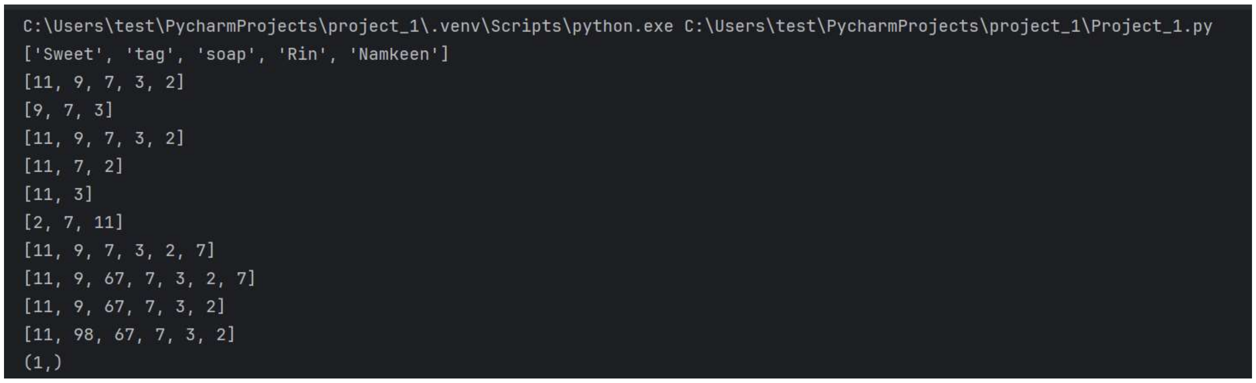
These are just a few examples of the functions you can use with Python lists. Lists and their associated functions provide a flexible way to work with collections of data in Python.

Code:

```python
product = ["Sweet","tag","soap","Rin","Namkeen"]
print(product)

#print(product [0])
numbers = [2,7,9,11,3]
#print(numbers[2])
numbers = [2,7,9,11,3]
numbers.sort()
#print(numbers)

numbers.reverse()
#print(numbers)

print(numbers[:5])
print(numbers[1:4])

print(numbers[::1])
print(numbers[::2])
print(numbers[::3])
print(numbers[::-2])

numbers.append(7)
print(numbers)

numbers.insert(2,67)
print(numbers)

numbers.pop()
print(numbers)

numbers[1] = 98
```

```
print(numbers)

#tp[1] = 8 #this will give an error as its
immutable
tp = (1,)
print(tp)
```

Output:

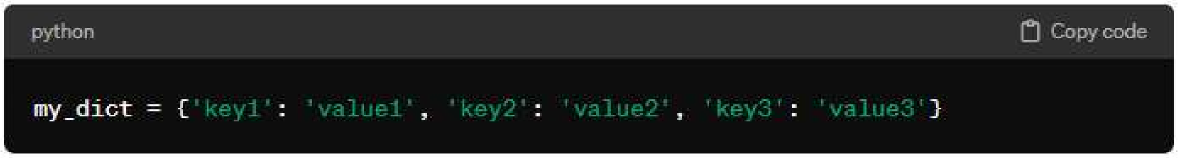


```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\Project_1.py
['Sweet', 'tag', 'soap', 'Rin', 'Namkeen']
[11, 9, 7, 3, 2]
[9, 7, 3]
[11, 9, 7, 3, 2]
[11, 7, 2]
[11, 3]
[2, 7, 11]
[11, 9, 7, 3, 2, 7]
[11, 9, 67, 7, 3, 2, 7]
[11, 9, 67, 7, 3, 2]
[11, 98, 67, 7, 3, 2]
(1,)
```

Dictionary:

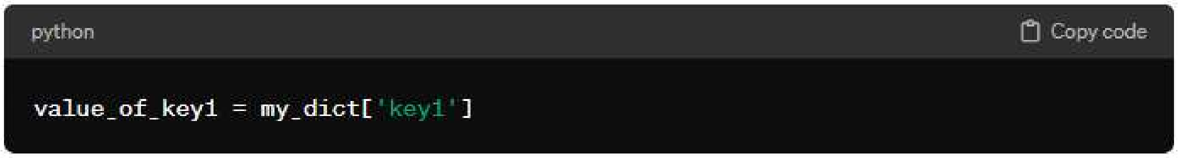
In Python, a dictionary is a built-in data type that allows you to store and retrieve key-value pairs. It is a flexible and powerful data structure that is commonly used for tasks like indexing, mapping, and organizing data. Here are some basic operations and functions associated with dictionaries in Python:

Creating a Dictionary:



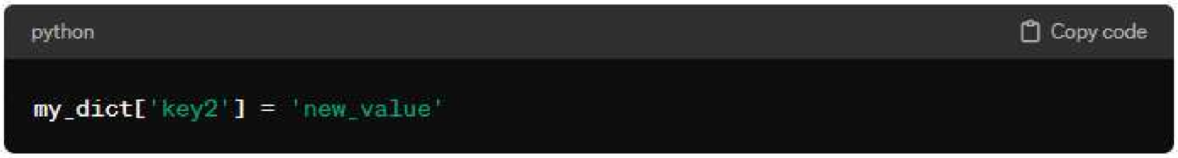
```python
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

Accessing Values:



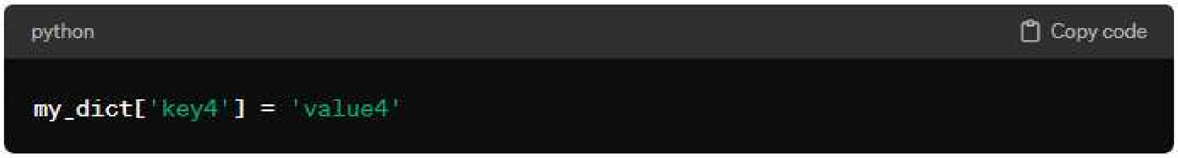
```python
value_of_key1 = my_dict['key1']
```

Modifying Values:



```python
my_dict['key2'] = 'new_value'
```

Adding a New Key-Value Pair:



```python
my_dict['key4'] = 'value4'
```

Removing a Key-Value Pair:

```python
del my_dict['key3']
```

Checking if a Key Exists:

```python
if 'key2' in my_dict:
    print('Key2 exists in the dictionary')
```

Dictionary Methods:

1. `keys()`: Returns a view of all keys in the dictionary.

   ```python
   keys = my_dict.keys()
   ```

2. `values()`: Returns a view of all values in the dictionary.

   ```python
   values = my_dict.values()
   ```

3. `items()`: Returns a view of all key-value pairs as tuples.

   ```python
   items = my_dict.items()
   ```

4. `get()`: Returns the value for a given key. If the key is not found, it can return a default value (or `None` if not specified).

   ```python
   value = my_dict.get('key1', 'default_value')
   ```

5. `pop()`: Removes and returns the value for a given key. If the key is not found, it can return a default value (or raise a `KeyError` if not specified).

```python
popped_value = my_dict.pop('key1', 'default_value')
```

6. `update()`: Updates the dictionary with key-value pairs from another dictionary or an iterable of key-value pairs.

```python
other_dict = {'key5': 'value5', 'key6': 'value6'}
my_dict.update(other_dict)
```

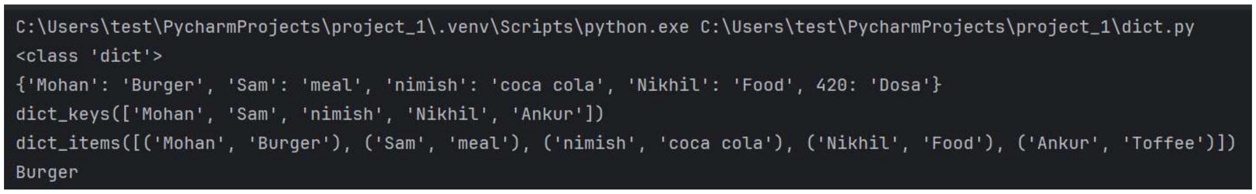
These are some of the basic functions and methods associated with dictionaries in Python. Dictionaries are widely used in various applications due to their efficiency in mapping and retrieving data.

Code:

```
d1 = {}
print(type(d1))

d2 = {"Mohan" : "Burger",
      "Sam" : "meal",
      "nimish" : "coca cola",
     }

d2["Nikhil"] = "Food"
d2[420] = "Dosa"
print(d2)

del d2[420]

d3 = d2.copy()
del d3["Mohan"]
d2.update({"Ankur":"Toffee"})
print(d2.keys())
print(d2.items())
```

```python
print(d2.get("Mohan"))
```

Output:



```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\dict.py
<class 'dict'>
{'Mohan': 'Burger', 'Sam': 'meal', 'nimish': 'coca cola', 'Nikhil': 'Food', 420: 'Dosa'}
dict_keys(['Mohan', 'Sam', 'nimish', 'Nikhil', 'Ankur'])
dict_items([('Mohan', 'Burger'), ('Sam', 'meal'), ('nimish', 'coca cola'), ('Nikhil', 'Food'), ('Ankur', 'Toffee')])
Burger
```