Accessing Elements

```
print(my_dict["name"])      # Output: John
print(my_dict.get("age"))   # Output: 25
```

Adding/Modifying Elements

```
my_dict["email"] = "john@example.com"    # Add new key-value pair
my_dict["age"] = 26                      # Modify existing value
```

Removing Elements

```
del my_dict["city"]       # Delete the key-value pair 'city'
my_dict.pop("email")      # Remove and return value of 'email'
my_dict.clear()           # Remove all elements
```

Iterating Over a Dictionary

```
for key, value in my_dict.items():
    print(key, value)
```

Dictionary Methods

```
my_dict.keys()        # Get all keys
my_dict.values()      # Get all values
my_dict.items()       # Get all key-value pairs
```

---

2. OBJECT ORIENTED PROGRAMMING

1. Creating a Class

```
class MyClass:
    def __init__(self, name):  # Constructor
        self.name = name       # Instance attribute

    def greet(self):           # Instance method
        print(f"Hello, {self.name}!")

# Creating an object (instance of MyClass)
obj = MyClass("Alice")
obj.greet()   # Output: Hello, Alice!
```

`__init__` Method (Constructor)

- Used to initialize objects.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

---

2. Class Attributes vs Instance Attributes

Positional Arguments

- Passed by position in the function call.

```
def add(a, b):
    return a + b

add(2, 3)  # Output: 5
```

Keyword Arguments

- Passed by name during function call.

```
def describe(name, age):
    return f"{name} is {age} years old."

describe(age=25, name="Alice")  # Output: Alice is 25 years old.
```

Default Arguments

- Assign default values to parameters.

```
def greet(name, message="Hello"):
    return f"{message}, {name}!"

greet("Alice")          # Output: Hello, Alice!
greet("Alice", "Hi")    # Output: Hi, Alice!
```

---

3. Variable-length Arguments

`*args` (Positional Variable-Length)

- Allows passing multiple arguments.

```
def add(*args):
    return sum(args)

add(1, 2, 3, 4)  # Output: 10
```

`**kwargs` (Keyword Variable-Length)

- Allows passing multiple keyword arguments.

```
def greet(**kwargs):
    return f"{kwargs['greeting']}, {kwargs['name']}!"

greet(greeting="Hi", name="Bob")  # Output: Hi, Bob!
```

---

4. Scope