Key Points About Python Dictionaries

- **Definition**:
  - A dictionary is a collection of key-value pairs, where keys are unique, and values can be of any data type.
  - It is optimized for fast value retrieval based on keys, unlike lists that use index positions.
- **Declaration**:
  - Use curly braces `{}` to create a dictionary.
  - Example:

```
my_dict = {1: "coffee", 2: "tea", 3: "juice"}
```

- **Accessing Values**:
  - Use square brackets `[]` with the key to retrieve the corresponding value.
  - Example: `my_dict[1]` returns `"coffee"`.
- **Adding and Updating Items**:
  - Add or update values using key assignment.
  - Example:

```
my_dict[4] = "water"  # Adds a new key-value pair
my_dict[2] = "mint tea"  # Updates the value of key 2
```

- **Deleting Items**:
  - Use the `del` operator with the key to remove an item.
  - Example: `del my_dict[3]` removes the key-value pair with key 3.
- **Iterating Through a Dictionary**:
  - **Keys Only**:

```
for key in my_dict:
    print(key)
```

  - **Keys and Values**: Use the `.items()` method.

```
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")
```

- **Duplicate Keys**:

    - Dictionaries do not allow duplicate keys. Assigning a new value to an existing key overwrites the old value.

- **Performance**:
    - Dictionary lookups are faster than searching through a list because of how keys are hashed internally.

- **Built-in Methods**:
    - `.keys()` : Returns all keys in the dictionary.
    - `.values()` : Returns all values in the dictionary.
    - `.items()` : Returns key-value pairs as tuples.

Example Code:

```
my_dict = {1: "coffee", 2: "tea", 3: "juice"}

# Accessing values
print(my_dict[1])  # Output: "coffee"

# Adding a new key-value pair
my_dict[4] = "water"

# Updating a value
my_dict[2] = "mint tea"

# Deleting a key-value pair
del my_dict[3]

# Iterating through keys and values
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")
```