The [] notation in Python is the standard way to interact with dictionaries. It allows you to access, add, update, or delete key-value pairs in a dictionary. Below are examples of its various applications:

1. Accessing Values

To retrieve the value associated with a specific key:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Access values
print(my_dict['name'])  # Output: Alice
print(my_dict['age'])   # Output: 30
```

Note: If the key doesn't exist, it will raise a KeyError.

2. Adding New Key-Value Pairs

To add a new key-value pair:

```
my_dict = {'name': 'Alice'}

# Add new key-value pair
my_dict['age'] = 30
print(my_dict)  # Output: {'name': 'Alice', 'age': 30}
```

3. Updating Values

To modify the value of an existing key:

```
my_dict = {'name': 'Alice', 'age': 30}

# Update value
my_dict['age'] = 31
print(my_dict)  # Output: {'name': 'Alice', 'age': 31}
```

4. Deleting Key-Value Pairs

To remove a key-value pair:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Delete a key-value pair
del my_dict['city']
print(my_dict)  # Output: {'name': 'Alice', 'age': 30}
```

5. Checking for Key Existence

To check if a key exists before accessing it:

```
my_dict = {'name': 'Alice', 'age': 30}

# Check if a key exists
if 'name' in my_dict:
    print(my_dict['name'])  # Output: Alice
if 'city' not in my_dict:
    print('Key not found')  # Output: Key not found
```

6. Using the get() Method

To avoid KeyError, use the get() method:

```
my_dict = {'name': 'Alice', 'age': 30}

# Safe access
print(my_dict.get('name'))     # Output: Alice
print(my_dict.get('city'))     # Output: None
print(my_dict.get('city', 'Unknown'))  # Output: Unknown
```

7. Iterating Through a Dictionary

a. Iterate Over Keys

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
for key in my_dict:
    print(key)
# Output:
# name
# age
# city
```

b. Iterate Over Values

```
for value in my_dict.values():
    print(value)
# Output:
# Alice
# 30
# New York
```

c. Iterate Over Key-Value Pairs

```
for key, value in my_dict.items():
    print(f"{key}: {value}")
# Output:
# name: Alice
```

```
# age: 30
# city: New York
```

8. Access Nested Dictionaries

If a dictionary contains another dictionary:

```
my_dict = {
    'person': {
        'name': 'Alice',
        'age': 30
    },
    'city': 'New York'
}

# Access nested values
print(my_dict['person']['name'])  # Output: Alice
print(my_dict['person']['age'])   # Output: 30
```

9. Default Dictionary Using collections.defaultdict

To avoid KeyError for missing keys:

```
from collections import defaultdict

# Create a defaultdict with a default value of 0
my_dict = defaultdict(int)

my_dict['a'] += 1
print(my_dict['a'])  # Output: 1
print(my_dict['b'])  # Output: 0 (default value)
```

10. Practical Examples

Example: Counting Word Frequencies

```
text = "hello world hello"
word_count = {}

for word in text.split():
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)
# Output: {'hello': 2, 'world': 1}
```

Example: Grouping Items

```
from collections import defaultdict
```

```
data = [('fruit', 'apple'), ('fruit', 'banana'), ('vegetable', 'carrot')]
grouped = defaultdict(list)

for category, item in data:
    grouped[category].append(item)

print(dict(grouped))
# Output: {'fruit': ['apple', 'banana'], 'vegetable': ['carrot']}
```

Let me know if you'd like further clarification or examples!