Dictionary in Python

Dictionaries in Python are collections of key-value pairs. Each key is unique, and the values can be any data type. Dictionaries are unordered and mutable, meaning you can change the values of existing keys or add new key-value pairs.

Creating a Dictionary

You can create a dictionary by placing a comma-separated sequence of key-value pairs within curly braces `{}`.

```
# Example of a dictionary
my_dict = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}
print(my_dict)  # Output: {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Accessing Dictionary Values

Values in a dictionary can be accessed using the keys.

```
# Accessing values
name = my_dict["name"]
print(f"Name: {name}")  # Output: Name: Alice
```

Adding and Modifying Dictionary Items

You can add new key-value pairs or update existing ones by assigning a value to the key.

```
# Adding a new key-value pair
my_dict["job"] = "Engineer"
print(my_dict)  # Output: {'name': 'Alice', 'age': 25, 'city': 'New York', 'job':
'Engineer'}

# Modifying an existing key-value pair
my_dict["age"] = 30
print(my_dict)  # Output: {'name': 'Alice', 'age': 30, 'city': 'New York', 'job':
'Engineer'}
```

Dictionary Methods

1. **get()**

The `get()` method returns the value of a key. If the key does not exist, it returns `None` (or a specified default value).

```
age = my_dict.get("age")
print(f"Age: {age}")  # Output: Age: 30

# With default value if key is missing
salary = my_dict.get("salary", "Not Available")
print(f"Salary: {salary}")  # Output: Salary: Not Available
```

2. **keys()**

The `keys()` method returns all the keys in the dictionary.

```
keys = my_dict.keys()
print(f"Keys: {keys}")  # Output: Keys: dict_keys(['name', 'age', 'city', 'job'])
```

3. **values()**

The `values()` method returns all the values in the dictionary.

```
values = my_dict.values()
print(f"Values: {values}")  # Output: Values: dict_values(['Alice', 30, 'New York', 'Engineer'])
```

4. **items()**

The `items()` method returns a list of key-value pairs (as tuples).

```
items = my_dict.items()
print(f"Items: {items}")  # Output: Items: dict_items([('name', 'Alice'), ('age', 30), ('city', 'New York'), ('job', 'Engineer')])
```

5. **update()**

The `update()` method updates the dictionary with the specified key-value pairs.

```
# Adding multiple key-value pairs
my_dict.update({"hobbies": ["reading", "traveling"], "city": "San Francisco"})
print(my_dict)
# Output: {'name': 'Alice', 'age': 30, 'city': 'San Francisco', 'job': 'Engineer', 'hobbies': ['reading', 'traveling']}
```

6. **pop()**

The `pop()` method removes the item with the specified key and returns its value.

```
age = my_dict.pop("age")
print(f"Removed age: {age}")  # Output: Removed age: 30
print(my_dict)  # Output: {'name': 'Alice', 'city': 'San Francisco', 'job':
'Engineer', 'hobbies': ['reading', 'traveling']}
```

7. **clear()**

The `clear()` method removes all the items from the dictionary.

```
my_dict.clear()
print(my_dict)  # Output: {}
```

8. **fromkeys()**

The `fromkeys()` method creates a new dictionary from the given sequence of keys, each having the same value.

```
keys = ["a", "b", "c"]
default_value = 0
new_dict = dict.fromkeys(keys, default_value)
print(new_dict)  # Output: {'a': 0, 'b': 0, 'c': 0}
```

Looping Through a Dictionary

You can iterate over the keys or the key-value pairs in a dictionary using a loop.

```
# Loop through keys
for key in my_dict:
    print(key)

# Loop through key-value pairs
for key, value in my_dict.items():
    print(f"{key}: {value}")
```

Dictionary Comprehension

You can use dictionary comprehension to create dictionaries in a concise way.

```python
squares = {x: x**2 for x in range(1, 6)}
print(squares)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Conclusion

Dictionaries are versatile data structures that allow you to store and manage data using key-value pairs. They provide various methods to access, update, and manipulate the stored data effectively.