Dictionaries

September 17, 2023

1 Dictionaries (hash maps)

Dictionaries in Python are versatile data structures that store collections of key-value pairs, where each key is unique. They are called "hash maps" because they typically use a hashing function to efficiently map keys to their corresponding values. This hashing process allows for rapid retrieval of values associated with specific keys, making dictionaries highly efficient for tasks like searching, indexing, and data retrieval. Python dictionaries are implemented as hash tables, making them suitable for applications requiring fast access to data with a known key, and their versatility makes them a fundamental component in many Python programs for organizing and managing data.

A hashing function is a mathematical function that takes an input (or "key") and transforms it into a fixed-size string of characters, typically a hexadecimal number or a binary value. The output, often referred to as a hash code or hash value, is deterministic, meaning that the same input will always produce the same hash value.

Hashing functions have several important characteristics:

1. Deterministic: As mentioned earlier, the same input will consistently yield the same hash value.
2. Fixed Output Length: The hash function produces a fixed-size output, regardless of the input's size. For example, the SHA-256 hash function always produces a 256-bit (32-byte) hash value.
3. Fast Computation: Hash functions are designed to be computationally efficient, allowing for quick calculation of hash values even for large inputs.
4. Avalanche Effect: A small change in the input should result in a significantly different hash value. This property ensures that similar inputs produce distinct hash codes.
5. Pre-image Resistance: It should be computationally infeasible to reverse the hash function and determine the original input from its hash value.

Hashing functions find extensive use in various computer science and security applications, including data integrity verification, password storage, data indexing, and cryptography. In Python, you can find built-in hash functions like `hash()` for basic hashing needs, and libraries like hashlib provide access to more advanced hash functions such as MD5, SHA-1, and SHA-256.

1.1 Creating Dictionaries

1.1.1 Using Curly Braces {}:

[ ]:
```
my_dictionary = {'name': 'John', 'age': 30, 'city': 'New York'}
print(my_dictionary)
```

1.1.2 Using the dict() Constructor:

[ ]:
```
my_dictionary = dict(name='Jane', age=25, city='Los Angeles')
print(my_dictionary)
```

1.1.3 Using a List of Tuples:

[ ]:
```
data = [('name', 'Alice'), ('age', 35), ('city', 'Chicago')]
my_dictionary = dict(data)
print(my_dictionary)
```

1.1.4 Using a List of Lists:

[ ]:
```
data = [['name', 'Bob'], ['age', 28], ['city', 'San Francisco']]
my_dictionary = {key: value for key, value in data}
print(my_dictionary)
```

1.1.5 Using zip() Function:

[ ]:
```
keys = ['fruit', 'color', 'quantity']
values = ['apple', 'red', 5]
my_dictionary = dict(zip(keys, values))
print(my_dictionary)
```

[ ]:
```
my_dictionary.values()
```

1.2 Operations with Dictionaries

1.2.1 Creating a dictionary

[ ]:
```
my_dictionary = {'name': 'John', 'age': 30, 'city': 'New York'}
```

1.2.2 Accessing Values

[ ]:
```
value = my_dictionary['name']  # Accessing a value
```

1.2.3 Adding or Updating Key-Value Pairs

```
[ ]: my_dictionary['city'] = 'Los Angeles'  # Adding a new key-value pair
     my_dictionary['age'] = 31  # Updating an existing value
```

1.2.4 Removing Key-Value Pairs

```
[ ]: del my_dictionary['age']  # Deleting a key-value pair
```

1.2.5 Checking for Key Existence

```
[ ]: if 'name' in my_dictionary:
         print('Key "name" exists in the dictionary')
```

1.2.6 Getting Dictionary Length

```
[ ]: length = len(my_dictionary)  # Getting the length of the dictionary
```

1.2.7 Iterating Over Keys and Values

```
[ ]: for key in my_dictionary:
         print(key)  # Iterating over keys

     for value in my_dictionary.values():
         print(value)  # Iterating over values

     for key, value in my_dictionary.items():
         print(key, value)  # Iterating over key-value pairs
```

1.2.8 Getting Keys and Values

```
[ ]: keys = list(my_dictionary.keys())  # Getting keys as a list
     values = list(my_dictionary.values())  # Getting values as a list
```

1.2.9 Copying Dictionaries

```
[ ]: new_dictionary = my_dictionary.copy()  # Creating a shallow copy
```

1.2.10 Nested Dictionaries

```
[ ]: nested_dict = {'person': {'name': 'Alice', 'age': 25}}
     name = nested_dict['person']['name']  # Accessing a value in a nested dictionary
```

```
[ ]:
```