**Hashing in Python**

Hashing is a technique used to convert data into a fixed-size numerical value (called a **hash** or **hash value**) using a mathematical function known as a **hash function**. Hashing is widely used in data structures, cryptography, and data integrity checks.

**1. Hash Functions**

A **hash function** takes an input (e.g., a string, number, or object) and returns a fixed-length integer. In Python, common hash functions include:

* **hash()** (Built-in function)
* **hashlib** (Module for cryptographic hashing)
* **Custom hash functions** (Using \_\_hash\_\_ method)

**Example using hash()**

print(hash("Hello"))

print(hash(42))

* hash() generates different values for different objects.
* Hash values may change across different Python sessions.

**2. Hashing with hashlib (Cryptographic Hashing)**

For secure hashing, Python provides the **hashlib** module with algorithms like MD5, SHA-1, SHA-256.

**Example using SHA-256**

import hashlib

message = "Hello, World!"

hash\_object = hashlib.sha256(message.encode()) # Encoding string before hashing

print(hash\_object.hexdigest()) # Prints the hash in hexadecimal format

**Common Cryptographic Hash Algorithms**

* hashlib.md5() – 128-bit hash (insecure)
* hashlib.sha1() – 160-bit hash (deprecated)
* hashlib.sha256() – 256-bit hash (secure)
* hashlib.sha512() – 512-bit hash (secure)

**3. Hashing in Data Structures**

**(a) Hashing in Dictionaries**

Python **dictionaries** use hashing internally to provide **O(1) average-time complexity** for key lookups.

my\_dict = {"name": "Alice", "age": 25}

print(my\_dict["name"]) # Uses hashing internally for quick access

**(b) Hashing in Sets**

Python **sets** also use hashing to allow **fast lookups** and prevent duplicate values.

my\_set = {1, 2, 3, 4, 1} # Duplicates are removed automatically

print(my\_set)

**4. Custom Hashing with \_\_hash\_\_()**

You can define a custom hash function for a class by overriding \_\_hash\_\_().

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_hash\_\_(self):

return hash((self.name, self.age)) # Hashing a tuple of attributes

person1 = Person("Alice", 25)

print(hash(person1))

**5. Collision Handling**

A **hash collision** occurs when two different inputs produce the same hash value. To resolve collisions, techniques like **chaining** and **open addressing** are used in hash tables.

Python’s built-in dictionaries use **open addressing** with **dynamic resizing** to handle collisions.

**6. Applications of Hashing**

* **Data structures** (Hash tables, dictionaries, sets)
* **Cryptography** (Password hashing, digital signatures)
* **Data integrity checks** (File checksums, integrity verification)
* **Load balancing** (Consistent hashing for distributed systems)
* **Database indexing** (Fast lookups)

**Summary**

| **Feature** | **hash()** | **hashlib** |
| --- | --- | --- |
| Hash Type | Non-cryptographic | Cryptographic (SHA, MD5) |
| Used in | Dictionaries, Sets | Security, password hashing |
| Fixed Output Size | No | Yes |
| Security | Weak | Strong (SHA-256, SHA-512) |