Q1.A Demonstrate the working of Hash function for the following data: Integer, float, character, string, tuple, list

In [4]:
```python
# Integer
print("Integer:", hash(42))

# Float
print("Float:", hash(3.14))

# Character
print("Character:", hash('a'))

# String
print("String:", hash("OpenAI"))

# Tuple
print("Tuple:", hash((1, 2, 3)))

# List (Lists are mutable and not hashable)
try:
    print("List:", hash([1, 2, 3]))
except TypeError as e:
    print("List: Error -", e)
```

```
Integer: 42
Float: 1846836513
Character: -1686884150
String: -570877195
Tuple: -2022708474
List: Error - unhashable type: 'list'
```

Q1.B Take a number list and check the hash index assigned to the set of values of the list.

In [5]:
```python
num_list = [10, 20, 30]
tuple_form = tuple(num_list)
print("Hash index (tuple):", hash(tuple_form))
```

```
Hash index (tuple): 1478614530
```

Q1.C Take any expression of your choice (e.g. hello world), and nd the hash value assigned to the expression using ord()function used to get the ordinal value of any character.

In [8]:
```python
expression = "hello world"
hash_value = sum(ord(char) for char in expression)
print("hash_value for expression is :",hash_value)
```

```
hash_value for expression is : 1116
```

Q1.D Demonstrate that Mutable objects like lists, dictionaries, and sets cannot be hashed with the hash() function.

```
In [12]: val = {
         "list": [1, 2, 3],
         "dict": {"name": "vishal", "age": 22},
         "sets": {"vishal", "pal"}
         }
         for i in val:
             try:
                 print("Hash value:", hash(val[i]))
             except TypeError as e:
                 print("Error:", e)
```

```
Error: unhashable type: 'list'
Error: unhashable type: 'dict'
Error: unhashable type: 'set'
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Q.2 Create a Hash Table with Collision and Demonstrate how Separate Chaining is used for Collision Handling

```
In [16]: class Node:
             # A simple node class to store key-value pairs in a linked list
             def __init__(self, key, value):
                 self.key = key
                 self.value = value
                 self.next = None

         class HashTable:
             def __init__(self, size=10):
                 self.size = size
                 self.table = [None] * size

             def hash(self, key):
                 return hash(key) % self.size

             def insert(self, key, value):
                 index = self.hash(key)
                 new_node = Node(key, value)

                 # If the bucket is empty, insert the node
                 if self.table[index] is None:
                     self.table[index] = new_node
                 else:
                     current = self.table[index]
                     while current.next:
                         current = current.next
                     current.next = new_node

             def search(self, key):
                 index = self.hash(key)
```

```
        current = self.table[index]

        while current:
            if current.key == key:
                return current.value
            current = current.next
        return None

    def display(self):
        for i in range(self.size):
            print(f"Bucket {i}: ", end="")
            current = self.table[i]
            while current:
                print(f"({current.key}: {current.value}) -> ", end="")
                current = current.next
            print("None")
```

In [17]:

```
hash_table = HashTable(5)


hash_table.insert("apple", 10)
hash_table.insert("banana", 20)
hash_table.insert("orange", 30)
hash_table.insert("mango", 40)
hash_table.insert("melon", 50)
hash_table.insert("grape", 60)


print(f"Search for 'banana': {hash_table.search('banana')}")
print(f"Search for 'grape': {hash_table.search('grape')}")


hash_table.display()
```

```
Search for 'banana': 20
Search for 'grape': 60
Bucket 0: None
Bucket 1: (apple: 10) -> (melon: 50) -> None
Bucket 2: (orange: 30) -> (mango: 40) -> None
Bucket 3: (banana: 20) -> (grape: 60) -> None
Bucket 4: None
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: