Name :- Deepawali . B. Mhaisagar
Assignment no 5

1. What does an empty dictionary's code look like?
ANS
In Python, an empty dictionary is represented by a pair of curly braces ({}). Here's how the code for an empty dictionary looks like:

```
my_dict = {}
```

The above code assigns an empty dictionary to the variable my_dict. The curly braces {} indicate an empty dictionary literal.

You can also create an empty dictionary using the dict() constructor function:

```
my_dict = dict()
```

2. What is the value of a dictionary value with the key 'foo' and the value 42?

ANS

The value of a dictionary with the key 'foo' and the value 42 would be 42. In a dictionary, values are associated with unique keys. You can access the value by specifying the key within square brackets ([]) or by using the get() method of the dictionary.

```
my_dict = {'foo': 42}

value = my_dict['foo']

print(value)  # Output: 42
```

Dictionaries provide a way to store and retrieve data using key-value pairs, allowing efficient and convenient data access based on unique keys.

3. What is the most significant distinction between a dictionary and a list?

ANS

The most significant distinction between a dictionary and a list in Python is how they organize and access their elements. Here are the key differences:

Organization of Elements:

- List: A list is an ordered collection of elements. The elements in a list are indexed and ordered based on their position in the list. Each element in a list is accessed by its index, starting from 0.
- Dictionary: A dictionary is an unordered collection of key-value pairs. The elements in a dictionary are not ordered by their position, but rather by their unique keys. Each element (value) in a dictionary is associated

with a unique key, allowing efficient access to values based on their keys.

Accessing Elements:

- List: In a list, elements are accessed using integer indices. You can retrieve an element from a list by specifying its index within square brackets ([]). For example, my_list[0] would retrieve the first element of the list.
- Dictionary: In a dictionary, elements are accessed using their keys. You can retrieve a value from a dictionary by specifying its key within square brackets ([]). For example, my_dict['key'] would retrieve the value associated with the key 'key' in the dictionary.

Mutable vs. Immutable:

- List: Lists are mutable, meaning their elements can be modified, added, or removed after the list is created. You can modify elements in a list using their indices or use various list methods to manipulate the list.
- Dictionary: Dictionaries are mutable as well. You can modify the values associated with existing keys, add new key-value pairs, or remove key-value pairs from the dictionary. However, the keys themselves are usually immutable (e.g., strings, numbers, tuples), as they need to be hashable to ensure efficient dictionary operations.

Use Cases:

- List: Lists are suitable for storing an ordered collection of elements when the order and position of elements matter. They are commonly used when you need to maintain a sequence, perform operations like sorting or appending, or when you want to access elements by their position in the list.
- Dictionary: Dictionaries are suitable when you need to associate values with unique keys and efficiently retrieve values based on those keys. They are commonly used when you want to store and access data based on specific identifiers or keys rather than their position.

In summary, while both lists and dictionaries are used to store and organize data, their key distinction lies in how they organize and access their elements. Lists are ordered and accessed by indices, while dictionaries are unordered and accessed by unique keys. The choice between them depends on the specific requirements of your program and the nature of the data you need to store and access.

4. What happens if you try to access spam[‘foo’] if spam is {‘bar’: 100}?

ANS

If you try to access spam['foo'] and spam is { 'bar': 100 }, you will encounter a KeyError because the key 'foo' does not exist in the dictionary spam.

```
spam = {'bar': 100}

value = spam['foo']  # This line raises a KeyErrorpam =]  # This line raises a KeyError
```

When the code attempts to access the key 'foo' in the dictionary spam, Python raises a KeyError because the key is not present in the dictionary. This error occurs because dictionaries require keys to be present in order to retrieve their corresponding values.

To avoid the KeyError, you can either ensure that the key 'foo' exists in the dictionary or use alternative techniques like the get() method to handle missing keys. Here's an example using get():

```
spam = {'bar': 100}

value = spam.get('foo')  # Returns None as 'foo' is not present
```

5. If a dictionary is stored in spam, what is the difference between the expressions ‘cat’ in spam and ‘cat’ in spam.keys()?

ANS

The expressions 'cat' in spam and 'cat' in spam.keys() check for the presence of the key 'cat' in the dictionary spam, but they differ in their approach.

'cat' in spam:

- This expression checks whether the key 'cat' exists directly in the dictionary spam.
- It returns a boolean value (True or False) indicating whether the key is present in the dictionary.
- It is a concise way to check for the existence of a key in a dictionary.

'cat' in spam.keys():

- This expression checks whether the key 'cat' exists in the keys of the dictionary spam.
- spam.keys() returns a view object that provides a dynamic view of the dictionary's keys.
- By using 'cat' in spam.keys(), the expression checks if the key 'cat' is present within that view of keys.

- It also returns a boolean value (True or False) indicating the presence of the key.

In terms of functionality, both expressions determine if the key 'cat' is present in the dictionary spam. However, 'cat' in spam directly checks the dictionary itself, while 'cat' in spam.keys() checks the view object of keys derived from the dictionary.

The difference is mainly in the approach and performance characteristics. 'cat' in spam is generally more efficient as it directly accesses the dictionary, while 'cat' in spam.keys() involves creating a view object of keys before performing the check.

It's worth noting that in Python, when you iterate over a dictionary, it is implicitly iterating over the keys. So, in many cases, checking for the presence of a key in a dictionary using 'cat' in spam is sufficient and more concise.

6. If a dictionary is stored in spam, what is the difference between the expressions ‘cat’ in spam and ‘cat’ in spam.Values()?

ANS

The expressions 'cat' in spam and 'cat' in spam.values() check for the presence of the value 'cat' in the dictionary spam, but they differ in their approach.

'cat' in spam:

- This expression checks whether the value 'cat' exists as a value directly in the dictionary spam.
- It returns a boolean value (True or False) indicating whether the value is present in the dictionary.
- It checks for the value across all the values in the dictionary.

'cat' in spam.values():

- This expression checks whether the value 'cat' exists as a value within the view object of values obtained from the dictionary spam.
- spam.values() returns a view object that provides a dynamic view of the dictionary's values.
- By using 'cat' in spam.values(), the expression checks if the value 'cat' is present within that view of values.
- It also returns a boolean value (True or False) indicating the presence of the value.

In summary, 'cat' in spam checks for the presence of the value 'cat' directly within the dictionary's values, while 'cat' in spam.values() checks for the presence of the value within the view object of values obtained from the dictionary. Both expressions determine if the value 'cat' is present in the dictionary spam, but they approach it from different angles.

It's important to note that while 'cat' in spam checks for the value across all values, it does so using a linear search that can be less efficient for larger dictionaries. On the other hand, 'cat' in spam.values() utilizes the view object of values, which can provide a more efficient membership check

7. What is a shortcut for the following code?

If ‘color’ not in spam:

spam[‘color’] = ‘black’

.ANS
A shortcut for the given code is to use the dict.setdefault() method. This method allows you to set a default value for a key in a dictionary only if the key doesn't already exist. Here's how you can use it as a shortcut:

spam.setdefault('color', 'black')

The setdefault() method takes two arguments: the key you want to check or set, and the default value to set if the key is not present in the dictionary. In this case, if the key 'color' is not already present in the dictionary spam, it will set the value 'black' for that key. If the key already exists, it will leave the value unchanged.

8. How do you "pretty print" dictionary values using which module and function?

ANS

To "pretty print" dictionary values in Python, you can use the pprint module and its pprint() function. The pprint module provides a way to format and display data structures in a more readable and organized manner.

Here's an example of how you can use the pprint module to pretty print a dictionary:

```
import pprint



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

pprint.pprint(my_dict)
```

The above code, the pprint.pprint() function is used to pretty print the dictionary my_dict. The function takes the dictionary as an argument and formats it in a visually appealing way, with each key-value pair on a separate line and indentation to represent nested structures.

The output will be something like:

```
{'age': 30,

 'city': 'New York',

 'name': 'John'}
```

The pprint module is particularly useful when dealing with complex data structures like dictionaries, nested dictionaries, or lists of dictionaries. It provides a clear and well-formatted representation of the data, making it easier to read and understand.

Note that the pprint module is part of the Python standard library, so there is no need to install any additional packages.

```
spam turns None as 'foo' is not present
```