**ASSIGNMENT 5**

1. What does an empty dictionary’s code look like?

Ans:

An empty dictionary in Python is represented by a pair of curly braces {}. Here's how an empty dictionary's code looks:

my\_dict = {}

1. 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 look like this:

my\_dict = {'foo': 42}

In this example, the dictionary my\_dict has a single key-value pair, where the key is 'foo' and the corresponding value is 42. The key-value pair is specified within curly braces {}, with the key and value separated by a colon :.

You can access the value associated with the key 'foo' using square bracket notation:

value = my\_dict['foo']

print(value)

In this case, value will be assigned the value 42, which is the value associated with the key 'foo' in the dictionary my\_dict.

1. 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 their underlying structure and how they store and access data:

i) Structure:

A dictionary is an unordered collection of key-value pairs, where each key is unique within the dictionary. Keys are used to access and retrieve the corresponding values.

A list is an ordered collection of elements that can be of any type. Elements in a list are ordered based on their position/index, starting from 0.

ii) Accessing Elements:

Dictionary elements are accessed by their unique keys. You use the keys to retrieve the corresponding values associated with them.

List elements are accessed by their indices. You use the index position to access specific elements in the list.

iii) Mutability:

Dictionaries are mutable, meaning you can modify, add, or remove key-value pairs after the dictionary is created.

Lists are also mutable, allowing you to modify, add, or remove elements within the list.

iv) Order:

Dictionaries do not have a defined order for their key-value pairs. The order of the items can be arbitrary and may change during operations.

Lists maintain the order of their elements, preserving the sequence in which they were added or modified.

v) Element Types:

Dictionary keys can be of any immutable type, such as strings, numbers, or tuples. Values within a dictionary can be of any type, including mutable types like lists or other dictionaries.

List elements can be of any type, including both immutable and mutable types.

Dictionaries are often used when you have data that can be uniquely identified by keys, allowing for efficient lookup and retrieval. Lists are used when you need an ordered collection of elements, typically accessed by their indices or when the order of the elements matters.

1. 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.

Here's an example of the error:

spam = {'bar': 100}

value = spam['foo'] # Raises a KeyError

When you attempt to access a key in a dictionary that doesn't exist, Python raises a KeyError indicating that the specified key is not found in the dictionary. In this case, trying to access the key 'foo' in the dictionary spam will raise a KeyError because 'foo' is not a valid key in the dictionary.

To avoid a KeyError, you can use the get() method or check for the existence of a key using the in keyword. For example:

spam = {'bar': 100}

value = spam.get('foo') # Returns None

if 'foo' in spam:

value = spam['foo']

else:

value = None

Using get() with a default value or checking for key existence allows you to handle cases where a key might not be present in the dictionary.

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

Ans:

If a dictionary is stored in spam, there is a difference between the expressions 'cat' in spam and 'cat' in spam.keys() in terms of what they check:

i) 'cat' in spam:

This expression checks whether the key 'cat' exists in the dictionary spam.

It returns True if the key 'cat' is present in spam, and False otherwise.

Example:

spam = {'cat': 42, 'dog': 99}

result = 'cat' in spam

print(result) # Output: True

In this example, 'cat' in spam evaluates to True because the key 'cat' is present in the dictionary spam.

ii) 'cat' in spam.keys():

This expression checks whether the string 'cat' is present as a key in the dictionary spam.

It returns True if the key 'cat' is present in spam, and False otherwise.

The .keys() method returns a view object containing all the keys of the dictionary.

Example:

spam = {'cat': 42, 'dog': 99}

result = 'cat' in spam.keys()

print(result) # Output: True

In this example, 'cat' in spam.keys() also evaluates to True because the key 'cat' is present in the dictionary spam.

In summary, both expressions check for the presence of the key 'cat' in the dictionary spam. The difference is that 'cat' in spam directly checks for the key in the dictionary, while 'cat' in spam.keys() first retrieves all the keys of the dictionary using .keys() and then checks for the presence of the key in the resulting list of keys.

Using 'cat' in spam is generally preferred as it directly checks the dictionary for the key, which is more efficient compared to creating a list of keys using .keys().

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

Ans:

In the expressions 'cat' in spam and 'cat' in spam.values(), there is a difference in what they check within the dictionary spam:

i) 'cat' in spam:

This expression checks whether the string 'cat' exists as a key in the dictionary spam.

It returns True if the key 'cat' is present in spam as a key, and False otherwise.

Example:

spam = {'cat': 42, 'dog': 99}

result = 'cat' in spam

print(result)

In this example, 'cat' in spam evaluates to True because the key 'cat' is present as a key in the dictionary spam.

ii) 'cat' in spam.values():

This expression checks whether the string 'cat' exists as a value in the dictionary spam.

It returns True if the value 'cat' is found among the values in spam, and False otherwise.

The .values() method returns a view object containing all the values of the dictionary.

Example:

spam = {'cat': 42, 'dog': 99}

result = 'cat' in spam.values()

print(result)

In this example, 'cat' in spam.values() evaluates to False because the value 'cat' is not present among the values in the dictionary spam.

In summary, 'cat' in spam checks for the presence of the key 'cat' in the dictionary spam, while 'cat' in spam.values() checks for the presence of the value 'cat' among the values in spam.

1. What is a shortcut for the following code?

if ‘color’ not in spam:

spam[‘color’] = ‘black’

Ans:

A shortcut for the given code can be achieved using the dict.setdefault() method. This method provides a concise way to set a default value for a key in a dictionary if the key is not already present.

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

1. How do you “pretty print” dictionary values using which module and function?

Ans:

i) Import the ‘pprint’ module:

import pprint

ii) Call the pprint() function with the dictionary you want to pretty print:

my\_dict = {'key1': 'value1', 'key2': 'value2'}

pprint.pprint(my\_dict)

The pprint() function will print the dictionary my\_dict in a visually pleasing format, with indentation and line breaks for better readability.

The pprint() function is especially useful when dealing with complex nested data structures, as it handles them gracefully, making it easier to analyze and debug your code.

Here's an example of pretty printing a dictionary:

import pprint

my\_dict = {'name': 'John Doe', 'age': 30, 'address': '123 Main St'}

pprint.pprint(my\_dict)

Output:

{'address': '123 Main St',

'age': 30,

'name': 'John Doe'}