**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 what it looks like:

empty\_dict = {}

This creates an empty dictionary named ‘empty\_dict’. You can then add key-value pairs to this dictionary as needed.

**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 is 42. In Python, you can access the value associated with a specific key in a dictionary using square brackets [] with the key inside. Here's how you would access the value:

my\_dict = {'foo': 42}

value\_of\_foo = my\_dict['foo']

print(value\_of\_foo) # Output: 42

So, the value associated with the key 'foo' in the dictionary my\_dict is 42.

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 data:

Organization: Lists are ordered collections of items, where each item is identified by its position or index. Lists are typically used when the order of elements matters or when you want to access elements by their index.

Dictionaries are unordered collections of key-value pairs. Each element in a dictionary is accessed by its key rather than by its position. Dictionaries are used when you want to store data in a way that allows for fast lookup based on a unique identifier (the key).

Accessing Elements: In a list, elements are accessed by their index, which is an integer starting from 0. For example, my\_list[0] would access the first element of the list my\_list.

In a dictionary, elements are accessed by their keys. For example, my\_dict['key'] would access the value associated with the key 'key' in the dictionary my\_dict.

Mutability: Both lists and dictionaries are mutable, meaning you can change their contents after they've been created. You can add, remove, or modify elements in both data structures.

Usage: Lists are typically used when you need a collection of elements that maintain their order and you want to access elements by their position.

Dictionaries are used when you need to associate each element with a unique identifier (the key) and want to access elements by these identifiers efficiently.

**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. This is because there is no key 'foo' in the dictionary spam.

In Python, when you attempt to access a key in a dictionary that doesn't exist, Python raises a KeyError to indicate that the key is not found in the dictionary.

spam = {'bar': 100}

value = spam['foo'] # This line will raise a KeyError

Attempting to access the key 'foo' in the dictionary spam will result in a KeyError because the key 'foo' does not exist in the dictionary.

**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 operate slightly differently:

'cat' in spam:

This expression checks whether the key 'cat' is present directly in the dictionary spam.

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

It directly checks the keys of the dictionary without explicitly calling the keys() method.

'cat' in spam.keys():

This expression explicitly calls the keys() method on the dictionary spam to retrieve a view of all the keys.

It then checks whether the key 'cat' is present in this view of keys.

It also returns True if the key 'cat' exists in spam, and False otherwise.

In summary, both expressions determine if the key 'cat' is present in the dictionary spam, but the first expression directly checks the keys of the dictionary, while the second expression first retrieves a view of keys before performing the check. In terms of functionality, they are essentially equivalent, but the second expression involves an additional method call (keys()).

**6. If a dictionary is stored in spam, what is the difference between the expressions 'cat' in spam and 'cat' in spam.values()?**

Ans: he expressions 'cat' in spam and 'cat' in spam.values() also check for the presence of the key 'cat' in the dictionary spam, but they have different meanings:

'cat' in spam:

This expression checks whether the key 'cat' is present directly in the dictionary spam.

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

It directly checks the keys of the dictionary.

'cat' in spam.values():

This expression explicitly calls the values() method on the dictionary spam to retrieve a view of all the values.

It then checks whether the value 'cat' is present in this view of values.

It returns True if the value 'cat' exists in spam, and False otherwise.

**7. What is a shortcut for the following code?**

**if 'color' not in spam:**

**spam['color'] = 'black'**

Ans: A shortcut for the provided code is to use the dict.setdefault() method. This method sets the value of a key in a dictionary if the key does not exist, and returns the value of the key whether it already existed or was just set.

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

This line of code checks if the key 'color' exists in the dictionary spam. If it does not exist, it sets the value of the key 'color' to 'black'. If the key already exists, it does nothing and leaves 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, specifically the pprint() function. The pprint module provides a way to pretty-print data structures such as dictionaries, lists, and tuples in a more human-readable format.

Here's how you can use it to pretty-print dictionary values:

import pprint

my\_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Pretty print the dictionary

pprint.pprint(my\_dict)

This will print the dictionary my\_dict in a more visually appealing and organized format, making it easier to read, especially when dealing with complex nested structures.