***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 {} with nothing inside them. So the code for an empty dictionary looks like this:

**{}**

It's important to note that a dictionary is a data structure in Python that stores key-value pairs, where each key is unique and associated with a corresponding value. When the dictionary is empty, it means there are no key-value pairs stored in it.

**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 simply 42. In Python dictionaries, values are associated with keys, allowing you to access the value by referencing its corresponding key. So, in this case, if you have a dictionary like this:

**{'foo': 42}**

You can retrieve the value by using the key 'foo', like this:

**value = my\_dict['foo']**

After executing this code, the variable value will contain the value **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 the way they store and retrieve data.

A dictionary is a collection of key-value pairs, where each key is unique and associated with a value. The key-value pairs in a dictionary are not ordered, meaning they are stored in an unordered manner. To access a value in a dictionary, you use its corresponding key.

On the other hand, a list is an ordered collection of elements. The elements in a list are stored in a specific order, and each element is assigned an index starting from 0. You can access elements in a list by their index.

To summarize:

Dictionaries store data as key-value pairs, while lists store data as a sequence of elements.

Dictionaries are accessed by keys, while lists are accessed by indices.

Dictionaries are unordered, while lists maintain the order of elements.

In general, you would use a dictionary when you have data that needs to be accessed or retrieved based on specific keys, while a list is suitable when you need to maintain the order of elements or access them by their positions.

**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.

In Python, when you try to access a dictionary using a key that does not exist in the dictionary, a KeyError is raised. In this case, since the key 'foo' does not exist in the spam dictionary, trying to access spam['foo'] will raise a KeyError indicating that the key 'foo' is not found.

Here's an example of the error message you would see:

KeyError: 'foo'

To avoid this error, it's recommended to either ensure that the key you're accessing exists in the dictionary or use error handling techniques such as try and except blocks to handle potential KeyError exceptions.

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

The expressions 'cat' in spam and 'cat' in spam.keys() check for the presence of the key 'cat' in a dictionary stored in the variable spam. However, there is a slight difference between these expressions:

'cat' in spam: This expression checks if the key 'cat' exists in the dictionary spam. It returns a Boolean value True if the key is present and False otherwise. This is a direct check for the key within the dictionary.

'cat' in spam.keys(): This expression checks if the key 'cat' exists in the list of keys of the dictionary spam. The keys() method returns a list containing all the keys of the dictionary. So 'cat' in spam.keys() returns True if the key 'cat' is present in the list of keys, and False otherwise.

In essence, the main difference lies in the object being searched. 'cat' in spam searches for the key directly within the dictionary, while 'cat' in spam.keys() searches for the key within the list of keys obtained from the dictionary using the keys() method.

However, it's important to note that in practice, these two expressions will usually yield the same result because both methods check for the presence of a key within the dictionary.

**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() have different meanings and return different results:

'cat' in spam: This expression checks if the key 'cat' exists in the keys of the dictionary spam. It returns True if the key is present in the keys, and False otherwise. It searches for the key within the dictionary's keys.

'cat' in spam.values(): This expression checks if the value 'cat' exists in the values of the dictionary spam. It returns True if the value is found in any of the values, and False otherwise. It searches for the value within the dictionary's values.

In summary:

'cat' in spam checks for the key 'cat' within the dictionary's keys.

'cat' in spam.values() checks for the value 'cat' within the dictionary's values.

It's important to note that dictionary values do not have to be unique, so 'cat' in spam.values() will return True if at least one of the values in the dictionary is equal to 'cat'. On the other hand, 'cat' in spam will return True only if 'cat' is a key in the dictionary.

**7. 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 allows you to set a default value for a key in a dictionary if the key does not already exist. Here's the equivalent shortcut code:

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

In this code, the setdefault() method checks if the key 'color' exists in the spam dictionary. If the key is present, it returns the corresponding value. If the key is not found, it sets the key-value pair 'color': 'black' in the dictionary. So, if 'color' does not exist in spam, it will be added with the value 'black'.

This provides a more concise way of achieving the same functionality as the original code you provided.

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

***ANS:***

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

Here's an example of how to use the pprint() function to pretty print dictionary values:

import pprint

my\_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

pprint.pprint(my\_dict)

By calling **pprint.pprint(my\_dict)**, the dictionary my\_dict will be printed in a formatted and more readable manner. The output will present the dictionary keys and values in a well-organized structure, making it easier to analyze and understand the contents of the dictionary.

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