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

In Python, an empty dictionary is represented by curly braces `{}`. Here's what the code looks like:

empty\_dict = {}

This creates a variable named `empty\_dict` and assigns an empty dictionary to it. You can later add key-value pairs to the dictionary as needed.

**2. What is the value of a dictionary value with the key ‘foo’ and the value 42?**

In a dictionary, if you have a key named ‘foo’ and its associated value is 42, then the value of the dictionary with the key ‘foo’ is **42**. Here’s how it would typically be represented in Python:

my\_dict = {'foo': 42}

print(my\_dict['foo']) # Output: 42

If you access my\_dict['foo'], it will return the value **42**.

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

In Python, the most significant distinction between a dictionary and a list is how they store and access data:

* **Dictionaries**:
  + Store data in **key-value pairs**.
  + Access elements by **keys**.
  + Are **unordered** (as of Python 3.6, dictionaries maintain insertion order, but they are not considered ordered collections).
  + Keys must be **unique** and **immutable** types, such as strings, numbers, or tuples.
* **Lists**:
  + Store a **sequence of elements**.
  + Access elements by **integer indices**.
  + Are **ordered**, meaning the order in which items are added is preserved.
  + Can contain **duplicates** and elements of **different types**.

Here’s a simple example to illustrate the difference:

# Dictionary with key-value pairs

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

print(my\_dict['key1']) # Output: value1

# List with a sequence of elements

my\_list = ['element1', 'element2', 'element3']

print(my\_list[0]) # Output: element1

In summary, use a **dictionary** when you need to associate unique keys with values, and use a **list** when you need an ordered collection of items.

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

If you try to access spam['foo'] when spam is {'bar': 100}, you will get a KeyError. This is because ‘foo’ is not a key in the dictionary spam. Here’s what the error might look like:

spam = {'bar': 100}

print(spam['foo'])

Output:

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

<ipython-input-1-8f47a2494864> in <module>

1 spam = {'bar': 100}

----> 2 print(spam['foo'])

KeyError: 'foo'

This error occurs because Python raises a KeyError whenever a dictionary is accessed with a key that does not exist in the dictionary. To avoid this, you can use the dictionary’s get method, which allows you to return a default value when the key is not found.

print(spam.get('foo', 'default\_value'))

In this case, 'default\_value' will be returned since 'foo' is not a key in spam. If you don’t provide a default value, None will be returned.

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

In Python, 'cat' in spam and 'cat' in spam.keys() are functionally equivalent. Both of these expressions check if 'cat' is a key in the dictionary spam.

Here’s how they work:

* 'cat' in spam: This checks if 'cat' is a key in the dictionary spam.
* 'cat' in spam.keys(): This also checks if 'cat' is a key in the dictionary spam. The keys() method returns a view object that displays a list of all the keys in the dictionary.

So, there is no difference between the two expressions in terms of their result. However, 'cat' in spam is more pythonic and efficient because it doesn’t need to generate a list of keys. It can check directly if 'cat' is a key in spam. On the other hand, 'cat' in spam.keys() generates a list of keys first, which can be slower for large dictionaries. Therefore, it’s generally recommended to use 'cat' in spam for checking if a key exists in a dictionary. Here’s an example:

spam = {'cat': 'fluffy', 'dog': 'furry'}

print('cat' in spam) # Output: True

print('cat' in spam.keys()) # Output: True

In both cases, the output is True because 'cat' is a key in the dictionary spam. If 'cat' was not a key in spam, both expressions would return False.

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

In Python, 'cat' in spam and 'cat' in spam.values() are used for different purposes:

* 'cat' in spam: This checks if 'cat' is a **key** in the dictionary spam.
* 'cat' in spam.values(): This checks if 'cat' is a **value** in the dictionary spam.

Here’s an example to illustrate the difference:

spam = {'animal': 'cat', 'cat': 'fluffy'}

print('cat' in spam) # Output: True

print('cat' in spam.values()) # Output: True

In this case, 'cat' is both a key and a value in the dictionary spam, so both expressions return True. However, if 'cat' was only a key or only a value, one of the expressions would return False. For example:

spam = {'animal': 'dog', 'cat': 'fluffy'}

print('cat' in spam) # Output: True

print('cat' in spam.values()) # Output: False

In this case, 'cat' is a key but not a value in the dictionary spam, so 'cat' in spam returns True and 'cat' in spam.values() returns False. This shows the difference between the two expressions. The first checks for keys, while the second checks for values.

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

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

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

The code you provided can be shortened using the dict.setdefault() method in Python. Here’s how you can do it:

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

The setdefault() method returns the value of the item with the specified key. If the key does not exist, it inserts the key with the specified value. So in this case, if ‘color’ is not a key in the dictionary spam, it will add ‘color’ with the value ‘black’. If ‘color’ is already a key in spam, it will do nothing. This is equivalent to your original code but in a more concise way.

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

In Python, you can “pretty print” dictionary values using the pprint module and specifically the pprint() function. Here’s an example:

import pprint

spam = {'cat': 'fluffy', 'dog': 'furry', 'mouse': 'squeaky'}

pprint.pprint(spam)

In this code, pprint.pprint(spam) will print the spam dictionary in a “pretty” way, which means it’s formatted for readability. The pprint module is part of the Python Standard Library, so you don’t need to install anything extra to use it. It’s very useful when you’re working with large dictionaries or other complex data structures, as it can make them easier to read.

Here’s how the output might look:

{'cat': 'fluffy', 'dog': 'furry', 'mouse': 'squeaky'}

In this case, it doesn’t look much different from a regular print() statement because the dictionary is small. But for larger dictionaries, pprint() will format the output in a more readable way. For example, it might print each key-value pair on a new line. You can also customize the indentation and other aspects of the formatting by passing additional arguments to the pprint() function. For example, pprint.pprint(spam, indent=4) will indent each line by 4 spaces.