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

   An empty dictionary in Python is created using a pair of curly braces with nothing inside them.

In [2]:
```
#The code for creating an empty dictionary looks like this:
my_dict = {}
#we can also create an empty dictionary using the built-in dict() function, like this:
my_dict = dict()


#Both of these methods will create a new, empty dictionary that you can then add key-v
```

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

   The value of a dictionary value with the key 'foo' and the value 42 would simply be the integer value 42.

In [4]:
```
#Here's an example of how you could create a dictionary with the key 'foo' and the val
my_dict = {'foo': 42}
```

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

   The most significant distinction between a dictionary and a list is the way they store and organize data.

   A list is an ordered collection of elements, where each element has an index (starting from 0) that indicates its position in the list. Lists are represented using square brackets, and elements in a list are separated by commas. Lists are used to store multiple items of the same data type or different data types.

   On the other hand, a dictionary is an unordered collection of key-value pairs, where each key is unique and associated with a value. Dictionaries are represented using curly braces, and key-value pairs are separated by commas. Dictionaries are used to store data in a way that can be easily looked up by key.

   In summary, lists are used to store ordered collections of elements, whereas dictionaries are used to store unordered collections of key-value pairs. Additionally, since dictionaries are indexed by keys, they allow for efficient lookup of values based on keys, which is not possible with lists.

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

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

In [5]:
```
spam = {'bar': 100}
value = spam['foo']
# This will raise a KeyError because 'foo' does not exist in spam
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13920\3014362772.py in <module>
      1 spam = {'bar': 100}
----> 2 value = spam['foo']
      3 # This will raise a KeyError because 'foo' does not exist in spam

KeyError: 'foo'
```

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

There is no difference between the expressions 'cat' in spam and 'cat' in spam.keys(), as they both check for the presence of the key 'cat' in the dictionary spam.

In Python, when we use the in operator with a dictionary, it checks for the presence of the key in the dictionary's keys. So, 'cat' in spam and 'cat' in spam.keys() are equivalent expressions.

In [6]:
```
spam = {'cat': 5, 'dog': 2, 'bird': 1}

# Using 'in' with the dictionary directly
if 'cat' in spam:
    print('Cat is in the dictionary')

# Using 'in' with the keys of the dictionary
if 'cat' in spam.keys():
    print('Cat is in the keys of the dictionary')
#Both if statements are true, indicating that 'cat' is present in the keys of the spam
```

```
Cat is in the dictionary
Cat is in the keys of 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()?

The expression 'cat' in spam checks if the string 'cat' is a key in the dictionary spam, whereas the expression 'cat' in spam.values() checks if the string 'cat' is a value in the dictionary spam.

In Python, the in operator can be used with a dictionary to check if a key is present in the dictionary, but it cannot be used to check if a value is present. To check if a value is present in a dictionary, you can use the values() method, which returns a list of all the values in the dictionary.

In [8]:
```
spam = {'cat': 5, 'dog': 2, 'bird': 1}

# Using 'in' with the dictionary to check for a key
if 'cat' in spam:
    print('Cat is a key in the dictionary')

# Using 'in' with the values of the dictionary to check for a value
if 5 in spam.values():
    print('5 is a value in the dictionary')
#The first if statement is true because 'cat' is a key in the spam dictionary.
#The second if statement is also true because the value 5 is a value in the spam dicti
```

```
Cat is a key in the dictionary
5 is a value in the dictionary
```

7. What is a shortcut for the following code? if 'color' not in spam: spam['color'] = 'black'

we can use the setdefault() method of the dictionary to set a default value for a key if it doesn't exist. Here's how we can use it as a shortcut for the code you provided:

In [9]:
```
spam.setdefault('color', 'black')
```

Out[9]:
```
'black'
```

This code checks if the key 'color' exists in the spam dictionary. If it does not exist, it sets the value of the 'color' key to 'black'. If the key already exists, it leaves the value unchanged.

The setdefault() method is a convenient way to add a default value for a key in a dictionary if it does not exist without having to write an if statement.

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

To "pretty print" a dictionary in Python, we can use the pprint module's pprint() function.

The pprint() function prints a Python data structure in a more readable and organized way than the regular print() function. It prints the data structure with indentation and line breaks, making it easier to read, especially for larger and nested data structures like dictionaries.

In [10]:
```
import pprint

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

pprint.pprint(my_dict)
```

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

{'age': 30, 'city': 'New York', 'name': 'John'} As we can see, the pprint() function printed the dictionary in a more organized and readable way than the regular print() function.

In [ ]:
```

```