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

An empty dictionary's code looks like this:

```

my\_dict = {}

```

In Python, a dictionary is defined by enclosing a comma-separated list of key-value pairs in curly braces `{}`, like this:

```

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

```

To create an empty dictionary, you simply use the same syntax, but without any key-value pairs inside the braces:

```

my\_dict = {}

```

This creates an empty dictionary that you can later add key-value pairs to.

**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 is simply 42. In Python dictionaries, each key is associated with a value, and you can retrieve the value for a given key using the syntax `my\_dict[key]`.

So, if you have a dictionary like this:

```

my\_dict = {'foo': 42, 'bar': 'hello'}

```

You can retrieve the value associated with the key 'foo' by using the syntax `my\_dict['foo']`, like this:

```

value = my\_dict['foo']

```

This will set the variable `value` to 42, which is the value associated with the key 'foo'.

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

The most significant distinction between a dictionary and a list is how the elements are stored and accessed.

In a list, elements are stored and accessed using an integer index, which starts from 0. Each element in the list is associated with a unique index, and you can access an element by specifying its index using square brackets (`[]`) with the index value inside.

In contrast, in a dictionary, elements are stored and accessed using key-value pairs. Each element in the dictionary is associated with a unique key, and you can access an element by specifying its key using square brackets (`[]`) with the key value inside. The key can be of any immutable type, such as a string, integer, or tuple.

Another difference is that lists are ordered, meaning that the elements have a specific order and can be accessed based on their position, while dictionaries are unordered, meaning that the elements do not have a specific order and cannot be accessed based on their position. Instead, elements are accessed based on their key.

In summary, the main difference between a dictionary and a list is that a dictionary stores elements as key-value pairs and is accessed by key, while a list stores elements as an ordered sequence and is accessed by index.

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

If you try to access `spam['foo']` and `spam` is `{'bar': 100}`, a `KeyError` will be raised.

In Python dictionaries, each key is associated with a value, and you can retrieve the value for a given key using the syntax `my\_dict[key]`. If you try to access a key that does not exist in the dictionary, Python will raise a `KeyError` to indicate that the key is not found.

In the case of `spam['foo']`, the key 'foo' does not exist in the dictionary `spam`, which only contains the key 'bar'. Therefore, Python will raise a `KeyError` when you try to access `spam['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 practical difference between the expressions `'cat' in spam` and `'cat' in spam.keys()`.

In Python, the `in` operator can be used to check if a given key exists in a dictionary. When used with a dictionary, it checks if the specified key is present in the dictionary and returns `True` if it is, and `False` otherwise.

The expressions `'cat' in spam` and `'cat' in spam.keys()` are equivalent, since the `keys()` method of a dictionary returns a view object of the dictionary's keys, which is used to check if the key 'cat' is in the dictionary.

Both expressions will return `True` if the key 'cat' is present in the dictionary `spam`, and `False` otherwise.

**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`.

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

In other words, the first expression checks for the presence of a key in the dictionary, while the second expression checks for the presence of a value.

For example, consider the following dictionary:

```

spam = {'name': 'Alice', 'age': 30, 'city': 'New York'}

```

The expression `'name' in spam` would return `True`, since 'name' is a key in the dictionary.

The expression `'Alice' in spam` would return `False`, since 'Alice' is not a key in the dictionary, but it would also return `True` if we use `'Alice' in spam.values()`, since 'Alice' is a value in the dictionary.

Similarly, the expression `30 in spam.values()` would return `True`, since 30 is a value in the dictionary.

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

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

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

The `setdefault()` method can be used as a shortcut for the following code:

```

if 'color' not in spam:

spam['color'] = 'black'

```

The `setdefault()` method returns the value of the key if it exists in the dictionary, and if it doesn't exist, it sets the key to the specified default value and returns that value.

Using this method, the above code can be simplified to a single line:

```

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

```

This code checks if the key 'color' exists in the dictionary `spam`, and if it doesn't, it sets the value of the key to 'black'. If the key already exists, the method returns the current value of the key.

**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 its `pprint()` function.

The `pprint()` function is used to print Python data structures, such as dictionaries, in a more readable and organized format. It prints the data structure with each key-value pair on a separate line, and indents the values for nested keys.

Here's an example of how to use `pprint()` to pretty print a dictionary:

```

import pprint

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

pprint.pprint(my\_dict)

```

This will produce the following output:

```

{'age': 30,

'city': 'New York',

'name': 'John'}

```

As you can see, the `pprint()` function has printed the dictionary in a more readable format, with each key-value pair on a separate line and indented for nested keys.