**Assignment 5**

What does an empty dictionary's code look like?

In Python, an empty dictionary is represented using a pair of curly braces **{}**.

Here's an example of how you can create an empty dictionary:

pythonCopy code

my\_dict = {}

You can also use the **dict()** constructor to create an empty dictionary:

pythonCopy code

my\_dict = dict()

Both of these methods will create an empty dictionary with no key-value pairs.

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

If a dictionary has a key-value pair of **'foo': 42**, then the value associated with the key **'foo'** is **42**.

In Python syntax, it would look like this:

pythonCopy code

my\_dict = {'foo': 42} value = my\_dict['foo'] print(value) # Output: 42

So, the value of the dictionary value with the key **'foo'** is **42**.

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

The most significant distinction between a dictionary and a list in Python is that a dictionary is a collection of key-value pairs, whereas a list is an ordered collection of values.

In a list, each element is assigned an index based on its position in the list. You can access elements in a list using their index. For example:

pythonCopy code

my\_list = [1, 2, 3, 4] print(my\_list[0]) # Output: 1

In contrast, in a dictionary, values are stored and accessed using their associated keys. Each key in a dictionary must be unique. You can access values in a dictionary using their keys. For example:

pythonCopy code

my\_dict = {'a': 1, 'b': 2, 'c': 3} print(my\_dict['a']) # Output: 1

Another difference is that the elements in a list are ordered and can be modified by changing their position or by adding or removing elements, whereas the keys in a dictionary are unordered and cannot be changed once they are set. However, the values in a dictionary can be changed.

Overall, the choice between using a dictionary and a list depends on the specific use case and the type of data you need to work with.

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 because the key **'foo'** is not present in the dictionary.

Here's an example:

pythonCopy code

spam = {'bar': 100} value = spam['foo'] # This will raise a KeyError

The output of this code will be:

vbnetCopy code

KeyError: 'foo'

This is because the dictionary **spam** does not have a key called **'foo'**. To avoid a **KeyError**, you can use the **get()** method of the dictionary, which returns **None** if the key is not present:

pythonCopy code

spam = {'bar': 100} value = spam.get('foo') # This will return None

In this case, **value** will be **None**.

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

In Python, when you use the **in** operator to check for membership in a dictionary, it checks for membership in the keys of the dictionary. Therefore, the expressions **'cat' in spam** and **'cat' in spam.keys()** are equivalent and will produce the same result.

Here's an example:

pythonCopy code

spam = {'cat': 1, 'dog': 2, 'mouse': 3} # Check if 'cat' is a key in the dictionary print('cat' in spam) # Output: True print('cat' in spam.keys()) # Output: True # Check if 'snake' is a key in the dictionary print('snake' in spam) # Output: False print('snake' in spam.keys()) # Output: False

In this example, both **spam** and **spam.keys()** return the keys of the dictionary, so the expressions **'cat' in spam** and **'cat' in spam.keys()** are equivalent and both return **True**.

However, it's worth noting that you can also check for membership in the values of a dictionary using the **in** operator with the **.values()** method. For example, **1 in spam.values()** would return **True** in the above example because **1** is a value in the **spam** dictionary.

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

In Python, when you use the **in** operator to check for membership in a dictionary, the expression **'cat' in spam** checks if **'cat'** is a key in the dictionary **spam**, while the expression **'cat' in spam.values()** checks if **'cat'** is a value in the dictionary **spam**.

So, if **spam** is a dictionary that contains **'cat'** as a key, then **'cat' in spam** would return **True**, while **'cat' in spam.values()** would return **False**, because **'cat'** is not a value in the dictionary.

On the other hand, if **spam** is a dictionary that contains **'cat'** as a value, then **'cat' in spam** would still return **False**, because **'cat'** is not a key in the dictionary, but **'cat' in spam.values()** would return **True**.

Here's an example:

pythonCopy code

spam = {'a': 'cat', 'b': 'dog', 'c': 'bird'} print('cat' in spam) # Output: False print('cat' in spam.values()) # Output: True spam = {'a': 1, 'b': 2, 'c': 'cat'} print('cat' in spam) # Output: False print('cat' in spam.values()) # Output: True

In the first example, **'cat'** is a value in the dictionary, but not a key, so **'cat' in spam** returns **False**, while **'cat' in spam.values()** returns **True**.

In the second example, **'cat'** is both a value and a string, so **'cat' in spam** still returns **False**, while **'cat' in spam.values()** returns **True**.

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

The following code block:

pythonCopy code

if 'color' not in spam: spam['color'] = 'black'

can be shortened to a single line using Python's **dict.setdefault()** method:

pythonCopy code

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

This line of code checks if the key **'color'** is in the dictionary **spam**. If it is not, then it adds the key **'color'** with the value **'black'** to the dictionary. If the key **'color'** is already in the dictionary, then the method does nothing and returns the current value of the key.

This method is a concise way to add a key-value pair to a dictionary only if the key is not already present.

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

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

Here's an example:

pythonCopy code

import pprint my\_dict = {'name': 'John', 'age': 30, 'city': 'New York'} pprint.pprint(my\_dict)

This will output the dictionary **my\_dict** in a more human-readable format:

arduinoCopy code

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

The **pprint()** function formats the dictionary with each key-value pair on a separate line, and also indents nested dictionaries or lists.

If you want to print the output to a file instead of to the console, you can use the **pprint()** function with the **stream** parameter:

pythonCopy code

import pprint my\_dict = {'name': 'John', 'age': 30, 'city': 'New York'} with open('my\_dict.txt', 'w') as f: pprint.pprint(my\_dict, stream=f)

This will write the dictionary **my\_dict** to the file **my\_dict.txt** in a human-readable format.