**Python Assignment 5**

Q1. What does an empty dictionary's code look like?

Ans. An empty dictionary's code in Python looks like this:

my\_dict = {}

This creates an empty dictionary object named my\_dict with no key-value pairs. An empty dictionary can also be created using the dict() constructor:

my\_dict = dict()

Both of these methods create an empty dictionary that can be populated with key-value pairs later on.

Q2. What is the value of a dictionary value with the key 'foo' and the value 42?

Ans. The value of a dictionary value with the key 'foo' and the value 42 would be 42. In Python, dictionaries are used to store key-value pairs. Keys are unique and immutable, while values can be of any type and mutable. To access a value in a dictionary, you can use the key as the index:

my\_dict = {'foo': 42}

value = my\_dict['foo']

print(value) # Output: 42

In the above example, my\_dict is a dictionary with one key-value pair where the key is 'foo' and the value is 42. We can access the value associated with the key 'foo' by indexing the dictionary with the key. The value 42 is then assigned to the variable value.

Q3. What is the most significant distinction between a dictionary and a list?

Ans. The most significant distinction between a dictionary and a list is that a dictionary is a collection of key-value pairs, while a list is an ordered collection of elements.

In a dictionary, each element is a key-value pair, where each key is unique and used to access its corresponding value. Dictionaries are commonly used to store data in a way that allows for fast lookup and retrieval of information based on a specific key.

In contrast, a list is an ordered collection of elements, where each element is identified by its position, or index, in the list. Lists are commonly used for storing and manipulating sequences of related data, such as a list of numbers or a list of names.

Overall, dictionaries are useful when you need to access values based on a specific key, while lists are useful when you need to work with an ordered collection of elements.

Q4. 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}, a KeyError will be raised because 'foo' is not a valid key in the dictionary spam.

Dictionaries in Python use keys to access their corresponding values. In this case, the dictionary spam only contains the key 'bar' and its corresponding value 100, so attempting to access any other key, such as 'foo', will result in a KeyError.

To avoid this error, you can either make sure to only access keys that are present in the dictionary or use the get() method to retrieve a value with a default return value if the key is not found, like this:

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

This code will return 'default\_value' because 'foo' is not a key in the spam dictionary.

Q5. If a dictionary is stored in spam, what is the difference between the expressions 'cat' in spam and 'cat' in spam.keys()?

Ans. There is no difference between the expressions 'cat' in spam and 'cat' in spam.keys(), as they both check if the key 'cat' is present in the dictionary spam.

In Python, the in keyword can be used to check if a key is present in a dictionary. When used with just the dictionary name, like 'cat' in spam, Python checks if the key 'cat' is present in the keys of the dictionary spam.

When used with the keys() method, like 'cat' in spam.keys(), Python first retrieves a list of all the keys in the dictionary spam, and then checks if the key 'cat' is present in that list.

Both expressions will return True if the key 'cat' is present in the dictionary spam, and False otherwise. However, using 'cat' in spam is more efficient because it avoids the creation of an intermediate list of keys, which can be expensive for large dictionaries.

Q6. 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() are different because they check for the presence of 'cat' in different parts of the dictionary spam.

The expression 'cat' in spam checks if the string 'cat' is a key in the dictionary spam. If 'cat' is a key in spam, then the expression evaluates to True, otherwise, it evaluates to False.

The expression 'cat' in spam.values() checks if the string 'cat' is a value in the dictionary spam. If 'cat' is a value in spam, then the expression evaluates to True, otherwise, it evaluates to False.

For example, if spam is a dictionary containing key-value pairs like this: {'animal': 'cat', 'color': 'black'}, then 'cat' in spam would evaluate to False because 'cat' is not a key in the dictionary spam, while 'cat' in spam.values() would evaluate to True because 'cat' is a value in the dictionary spam.

In general, the expression 'cat' in spam checks for the presence of 'cat' as a key in the dictionary, while 'cat' in spam.values() checks for the presence of 'cat' as a value in the dictionary.

Q7. What is a shortcut for the following code?

if 'color' not in spam:

spam['color'] = 'black'

Ans. A shortcut for the following code:

if 'color' not in spam:

spam['color'] = 'black'

is to use the setdefault() method of the dictionary object, like this:

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

The setdefault() method returns the value of the specified key if it exists in the dictionary. If the key is not present in the dictionary, the method adds the key with the specified default value to the dictionary and returns the default value.

So in this case, if the key 'color' is not present in the dictionary spam, spam.setdefault('color', 'black') will add the key-value pair ('color', 'black') to the dictionary, and if the key 'color' is already present in the dictionary, the method will simply return the existing value of the key.

This code achieves the same result as the original code, but it is more concise and reads more naturally.

Q8. How do you "pretty print" dictionary values using which module and function?

Ans. To "pretty print" dictionary values in Python, you can use the pprint module, which provides a pprint() function that formats the dictionary in a human-readable way.

Here is an example of how to use pprint to print a dictionary:

import pprint

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

pprint.pprint(my\_dict)

This will output the dictionary in a formatted and readable way:

{'age': 30,

'city': 'New York',

'name': 'John'}

The pprint() function can also be used with optional arguments to customize the output format, such as the width argument to specify the maximum width of the output, and the indent argument to specify the number of spaces to use for indentation.

pprint.pprint(my\_dict, width=1, indent=4)

This will output the dictionary with each key-value pair on a separate line, indented by 4 spaces, with a maximum width of 1 character per line:

{ 'age': 30,

'city': 'New York',

'name': 'John'}

Using pprint is a convenient way to print complex data structures like dictionaries in a readable way.value if no exception is raised.