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

An empty dictionary can be initialised by using empty curly braces {}
Example: empty_dict = {}

Another method to initialise an empty dictionary is by using the keyword: dict()
Example: empty_dict = dict()

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

Dictionary representation (assuming it has only one Key-Value Pair): dict_= {'foo':42}
Here, KEY is 'foo', its corresponding value is: 42. The KEY: VALUE pair is 'foo':42
Hence, VALUE is 42.

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

Lists and Dictionaries store heterogeneous data types. Both Lists and Dictionaries are MUTABLE, i.e. the object and elements of the object can be modified during the execution of the program.

But there are two main differences:

- **Sequence of Data Stored**
  - **Lists** are ordered data types. In other terms, the order (sequence) of the elements stored in a list is preserved. This is the basis of the indexing in lists. For example, list[0] represents the first element and list[len-1] represents the last element of the list (len: number of elements in the list)
  - **Dictionaries** are unordered; there is NO persistent way of storing the elements of a dictionary. The elements of a dictionary cannot be indexed by using 0 indexed numbering systems as in case of a list. However, the KEY: VALUE pairs are always persistent. The only anarchy is w.r.t the order in which multiple KEY: VALUE pairs are stored.
- **Indexing Method**
  - **Lists** are indexed based on the element index. Python's lists are is 0 indexed. Any list has its elements indexed from 0 to (len-1) where 'len' represents the length of the list, i.e. the number of elements in the list. **For example: list[x] refers to the element stored at the x^{th} index in the list.**
  - As dictionaries are not ordered datatypes, the elements are not stored based on 0 indexing. But, in contrast to the lists, where each element is just a value, dictionaries have KEY: VALUE pairs as elements. In this combination, the KEYs perform the role of an INDEX. To access a particular value from a dictionary, indexing is done using KEY like: dict[key]. **The command dict[key] returns the value corresponding to the KEY: key in the dictionary 'dict'.**

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

On trying to access a value corresponding to a non-existent KEY in the dictionary, Python throws a KeyError. For the example her, the KEY 'foo' is not available in the dictionary 'spam', hence, the statement will result in KeyError

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

By default, when an iterator or a logical check like ‘in’ is applied on a dictionary, it is applied on the ‘KEYs’ and not on the VALUEs. So in the expression above, for the dictionary ‘spam’, the expressions ‘cat’ in spam and ‘cat’ in spam.keys() are equivalent to each other. In case of both the expressions, If ‘cat’ is there is the either of the key of the dictionary ‘spam’, the expression will result in True, else False

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

By default, when an iterator or a logical check like ‘in’ is applied on a dictionary, it is applied on the ‘KEYs’ and not on the VALUEs.
If the keyword, .values() is used, the logical operation applies over the values rather than the keys. Hence, ‘cat’ in spam checks over the keys while ‘cat’ in spam.values() checks over the values.

7. What is a shortcut for the following code?

if 'color' not in spam:

spam['color'] = 'black'

Please clarify, if I use any sort of dictionary comprehension, it is mandated to use a for loop. A back to back for and if loop checking and imposing the above condition inside a comprehension replaces all the contents of the dictionary with this one. In case, I just want to check if the KEY : ‘color’ exists or not, and want to add a KEY: Value pair if ‘color’ is unavailable, in that case, I feel, the best way is whatever is given in the question itself. Please help me with this.

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

The module used for pretty print is ‘**pprint’**. The function used to pretty print is ‘**PrettyPrinter’**.
Example :

```
import pprint
pp = pprint.PrettyPrinter(indent=1, compact=True)
pp.pprint(dict)
```