In [ ]:
```
1. What does an empty dictionary's code look like?

Two curly brackets: {}
```

In [ ]:
```
2. What is the value of a dictionary value with the key 'foo' and
the value 42?

{'foo':42}
```

In [ ]:
```
3. What is the most significant distinction between a dictionary
and a list?

The items stored in a dictionary are unordered, while the items
in a list are ordered.
```

In [ ]:
```
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 encounter a ‘KeyError’ because the key ‘'foo'’ does not
exist in the dictionary ‘spam’.

In Python, accessing a dictionary using square brackets ([]) with
a key that doesn't exist in the dictionary raises a ‘KeyError’.

In this case, since ‘'foo'’ is not a key in the dictionary‘spam’,
Python will raise a ‘KeyError’ with an error message indicating
that ‘'foo'’ is not a valid key in ‘spam’.
```

In [ ]:

```
5. 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 a dictionary is stored in the variable spam,
there are two expressions that can be used to check if a key
'cat' exists in the dictionary: 'cat' in spam and 'cat' in spam.keys().

The expressions 'cat' in spam and 'cat' in spam.keys() are
functionally equivalent and will yield the same result.
Both expressions check if the key 'cat' exists in the dictionary spam.
If the key is present, both expressions will evaluate to True;
otherwise, they will evaluate to False.

The difference between the two expressions lies in their underlying
implementation.

When we use 'cat' in spam, Python internally checks if the key
'cat' exists in the dictionary spam. It does this by performing a
membership test directly on the dictionary's keys. This approach
is more efficient because it avoids unnecessary creation of an
intermediate list of keys.

On the other hand, when you use 'cat' in spam.keys(),
Python explicitly calls the keys() method on the dictionary spam.
The keys() method returns a view object that represents a dynamic
view of the dictionary's keys. Python then performs the membership
test on this view object. Although this approach works correctly,
it incurs the overhead of creating the view object.

In summary, both 'cat' in spam and 'cat' in spam.keys() will provide
the same result, but 'cat' in spam is generally more efficient
since it performs the membership test directly on the dictionary's
keys without creating an intermediate view object.
```

In [ ]:
```
6. 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 a dictionary is stored in the variable spam,
the expressions 'cat' in spam and 'cat' in spam.values() have
different meanings and yield different results.

The expression 'cat' in spam checks if the key 'cat' exists in
the dictionary spam. It performs a membership test on the keys
of the dictionary. If the key is present, the expression evaluates
to True; otherwise, it evaluates to False. This expression does
not consider the values stored in the dictionary.

On the other hand, the expression 'cat' in spam.values() checks
if the value 'cat' exists in any of the values of the dictionary
spam. It performs a membership test on the values of the dictionary.
If the value is present in any of the values, the expression evaluates
to True; otherwise, it evaluates to False.

Here's an example to illustrate the difference:
```

In [1]:
```
spam = {'animal1':'dog','animal2':'cat','animal3':'fish'}

'cat' in spam # Evaluates to True, as 'cat' is a key in spam
```

Out[1]:
```
True
```

In [2]:
```
'cat' in spam.values() # Evaluates to True, as 'cat' is a value in spam
```

Out[2]:
```
True
```

In [ ]:
```
In the example above, 'cat' is a key in the dictionary spam,
so 'cat' in spam evaluates to True. Additionally, 'cat' is
also a value in the dictionary spam, so 'cat' in spam.values()
also evaluates to True.

In summary, 'cat' in spam checks if 'cat' is a key in the
dictionary spam, while 'cat' in spam.values() checks if 'cat'
is a value in any of the values of the dictionary spam.
```

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

A shortcut for the given code is to use the dict.setdefault() method.
It allows to set a default value for a key in a dictionary
if the key does not already exist.
Here's how we can use it as a shortcut:
```

In [3]:
```
spam.setdefault('color', 'black')
```

Out[3]:
```
'black'
```

In [ ]:
```
In this case, if the key 'color' does not exist in the
spam dictionary, setdefault() will add the key 'color'
with the value 'black'. If the key already exists,
it will not modify the existing value.

This code is functionally equivalent to the if statement
 provided, but it achieves the same result in a more concise way.
```

In [ ]:
```
8. How do you 'pretty print' dictionary values using which
module and function?
```

In [ ]:
```
To "pretty print" dictionary values in Python, we can make use
of the pprint module and its pprint function. The pprint module
provides a way to format and display data structures in a more
readable and visually appealing manner.

Here's an example of how we can use the pprint module to
pretty print dictionary values:
```

In [4]:
```
import pprint

my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

pprint.pprint(my_dict)
```

```
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

In [ ]:
```
The pprint.pprint() function takes a dictionary as an argument
and prints it in a morestructured and visually appealing format.
It automatically formats the dictionary with indentation and
line breaks, making it easier to read.

When I run the code above, I saw the dictionary
my_dict printed in a pretty printed format:
```

In [5]:
```
{'key1': 'value1',
 'key2': 'value2',
 'key3': 'value3'}
```

Out[5]:
```
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

In [ ]:
```
The pprint module also provides other functions,
such as pprint.pformat(), which returns a formatted string
instead of printing directly to the console. This can be
useful if I want to store or manipulate the pretty printed
output as a string.

Remember to import the pprint module before using the
pprint function or any other function from the module.
```