Python Class Variables

Summary: in this tutorial, you'll learn how the Python class variables (or attributes) work.

Introduction to the Python class variables

Everything in Python is an object including a class. In other words, a class is an object in Python.

When you define a class using the `class` keyword, Python creates an object with the name the same as the class's name. For example:

```
class HtmlDocument:
    pass
```

This example defines the `HtmlDocument` class and the `HtmlDocument` object. The `HtmlDocument` object has the `__name__` property:

```
print(HtmlDocument.__name__) # HtmlDocument
```

And the `HTMLDocument` has the type of `type`:

```
print(type(HtmlDocument))  # <class 'type'>
```

It's also an instance of the `type` class:

```
print(isinstance(HtmlDocument, type)) # True
```

Class variables are bound to the class. They're shared by all instances of that class.

The following example adds the `extension` and `version` class variables to the `HtmlDocument` class:

```
class HtmlDocument:
    extension = 'html'
    version = '5'
```

Both `extension` and `version` are the class variables of the `HtmlDocument` class. They're bound to the `HtmlDocument` class.

Get the values of class variables

To get the values of class variables, you use the dot notation (`.`). For example:

```
print(HtmlDocument.extension) # html
print(HtmlDocument.version) # 5
```

If you access a class variable that doesn't exist, you'll get an `AttributeError` exception. For example:

```
HtmlDocument.media_type
```

Error:

```
AttributeError: type object 'HtmlDocument' has no attribute 'media_type'
```

Another way to get the value of a class variable is to use the `getattr()` function. The `getattr()` function accepts an object and a variable name. It returns the value of the class variable. For example:

```
extension = getattr(HtmlDocument, 'extension')
version = getattr(HtmlDocument, 'version')

print(extension)  # html
print(version)  # 5
```

If the class variable doesn't exist, you'll also get an `AttributeError` exception. To avoid the exception, you can specify a default value if the class variable doesn't exist like this:

```
media_type = getattr(HtmlDocument, 'media_type', 'text/html')
print(media_type) # text/html
```

Set values for class variables

To set a value for a class variable, you use the dot notation:

```
HtmlDocument.version = 10
```

or you can use the `setattr()` built-in function:

```
setattr(HtmlDocument, 'version', 10)
```

Since Python is a dynamic language, you can add a class variable to a class at runtime after you have created it. For example, the following adds the `media_type` class variable to the `HtmlDocument` class:

```
HtmlDocument.media_type = 'text/html'
print(HtmlDocument.media_type)
```

Similarly, you can use the `setattr()` function:

```
setattr(HtmlDocument, media_type, 'text/html')
print(HtmlDocument.media_type)
```

Delete class variables

To delete a class variable at runtime, you use the `delattr()` function:

```
delattr(HtmlDocument, 'version')
```

Or you can use the `del` keyword:

```
del HtmlDocument.version
```

Class variable storage

Python stores class variables in the `__dict__` attribute. The `__dict__` is a mapping proxy, which is a dictionary. For example:

```
from pprint import pprint


class HtmlDocument:
    extension = 'html'
    version = '5'


HtmlDocument.media_type = 'text/html'

pprint(HtmlDocument.__dict__)
```

Output:

```
mappingproxy({'__dict__': <attribute '__dict__' of 'HtmlDocument' objects>,
              '__doc__': None,
              '__module__': '__main__',
              '__weakref__': <attribute '__weakref__' of 'HtmlDocument' objects>,
              'extension': 'html',
              'media_type': 'text/html',
              'version': '5'})
```

As clearly shown in the output, the `__dict__` has three class variables: `extension`, `media_type`, and `version` besides other predefined class variables.

Python does not allow you to change the `__dict__` directly. For example, the following will result in an error:

```
HtmlDocument.__dict__['released'] = 2008
```

Output:

```
TypeError: 'mappingproxy' object does not support item assignment
```

However, you can use the `setattr()` function or dot notation to indirectly change the `__dict__` attribute.

Also, the key of the `__dict__` are strings that will help Python speeds up the variable lookup.

Although Python allows you to access class variables through the `__dict__` dictionary, it's not a good practice. Also, it won't work in some situations. For example:

```
print(HtmlDocument.__dict__['type']) # BAD CODE
```

Callable class attributes

Class attributes can be any object such as a function.

When you add a function to a class, the function becomes a class attribute. Since a function is callable, the class attribute is called a callable class attribute. For example:

```
from pprint import pprint


class HtmlDocument:
    extension = 'html'
    version = '5'

    def render():
        print('Rendering the Html doc...')


pprint(HtmlDocument.__dict__)
```

Output:

```
mappingproxy({'__dict__': <attribute '__dict__' of 'HtmlDocument' objects>,
              '__doc__': None,
              '__module__': '__main__',
              '__weakref__': <attribute '__weakref__' of 'HtmlDocument' objects>,
              'extension': 'html',
              'render': <function HtmlDocument.render at 0x0000010710030310>,
```

```
                'version': '5'})
Rendering the Html doc...
```

In this example, the `render` is a class attribute of the `HtmlDocument` class. Its value is a function.

Summary

- A class is an object which is an instance of the type class.
- Class variables are attributes of the class object.
- Use dot notation or `getattr()` function to get the value of a class attribute.
- Use dot notation or `setattr()` function to set the value of a class attribute.
- Python is a dynamic language. Therefore, you can assign a class variable to a class at runtime.
- Python stores class variables in the `__dict__` attribute. The `__dict__` attribute is a dictionary.