Python Enumeration

Summary: in this tutorial, you'll learn about Python enumeration and how to use it effectively.

Introduction to the Python Enumeration

By definition, an enumeration is a set of members that have associated unique constant values. Enumeration is often called enum.

Python provides you with the `enum` module that contains the `Enum` type for defining new enumerations. And you define a new enumeration type by subclassing (https://www.pythontutorial.net/python-oop/python-inheritance/) the `Enum` class.

The following example shows how to create an enumeration called `Color` :

```
from enum import Enum


class Color(Enum):
    RED = 1
    GREEN = 2
    BLUE = 3
```

How it works.

First, import the `Enum` type from the `enum` module:

```
from enum import Enum
```

Second, define the `Color` class that inherits from the `Enum` type:

```
class Color(Enum):
```

Third, define the members of the `Color` enumeration:

```
RED = 1
GREEN = 2
BLUE = 3
```

Note that the enumeration's members are constants. Therefore, their names are in uppercase letters by convention.

In this example, the `Color` is an enumeration. The `RED` , `GREEN` , and `BLUE` are members of the `Color` enumeration. They have associated values 1, 2, and 3.

The type of a member is the enumeration to which it belongs.

The following illustrates that the type of `Color.RED` is the `Color` enumeration:

```
print(type(Color.RED))
```

Output:

```
<enum 'Color'>
```

The `Color.RED` is also an instance of the `Color` enumeration:

```
print(isinstance(Color.RED, Color))
```

Output:

```
True
```

And it has the name and value attributes:

```
print(Color.RED.name)
print(Color.RED.value)
```

Output:

```
RED
1
```

Membership and equality

To check if a member is in an enumeration, you use the `in` operator. For example:

```
if Color.RED in Color:
    print('Yes')
```

Output:

```
Yes
```

To compare two members, you can use either `is` or `==` operator. For example:

```
if Color.RED is Color.BLUE:
    print('red is blue')
else:
    print('red is not blue')
```

Output:

```
red is not blue
```

Note that a member and its associated value are not equal. The following example returns `False` :

```
if Color.RED == 1:
    print('Color.RED == 1')
else:
    print('Color.RED != 1')
```

Output:

```
Color.RED != 1
```

Enumeration members are hashable

Enumeration members are always hashable (https://www.pythontutorial.net/python-oop/python-__hash__/) . It means that you can use the enumeration members as keys in a dictionary (https://www.pythontutorial.net/python-basics/python-dictionary/) or as elements of a Set (https://www.pythontutorial.net/python-basics/python-set/) .

The following example uses the members of the `Color` enumeration in a dictionary:

```
rgb = {
    Color.RED: '#ff0000',
    Color.GREEN: '#00ff00',
    Color.BLUE: '#0000ff'
}
```

Access an enumeration member by name and value

The typical way to access an enumeration member is to use the dot notation (.) syntax as you have seen so far:

```
Color.RED
```

Because the `Enum` implements the `__getitem__` method, you can also use a square brackets `[]` syntax to get a member by its name.

For example, the following uses the square brackets `[]` syntax to get the `RED` member of the `Color` enumeration by its name:

```
print(Color['RED'])
```

Output:

```
Color.RED
```

Since an enumeration is callable (https://www.pythontutorial.net/python-built-in-functions/python-callable/) , you can get a member by its value. For example, the following return the `RED` member of the `Color` enumeration by its value:

```
print(Color(1))
```

Output:

```
Color.RED
```

The following expression returns `True` because it accesses the same enumeration member using name and value:

```
print(Color['RED'] == Color(1))
```

Output:

```
True
```

Iterate enumeration members

Enumerations are iterables so you can iterate them using a `for` loop. For example:

```
for color in Color:
    print(color)
```

Output:

```
Color.RED
Color.GREEN
Color.BLUE
```

Notice that the order of the members is the same as in the enumeration definition.

Also, you can use the `list()` function to return a list of members from an enumeration:

```
print(list(Color))
```

Output:

```
[<Color.RED: 1>, <Color.GREEN: 2>, <Color.BLUE: 3>]
```

Enumerations are immutable

Enumerations are immutable (https://www.pythontutorial.net/advanced-python/python-mutable-and-immutable/) . It means you cannot add or remove members once an enumeration is defined. And you also cannot change the member values.

The following example attempts to assign a new member to the `Color` enumeration and causes a `TypeError` :

```
Color['YELLOW'] = 4
```

Error:

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

The following example attempts the change the value of the `RED` member of the `Color` enumeration and causes an `AttributeError` :

```
Color.RED.value = 100
```

Output:

```
AttributeError: can't set attribute
```

Inherits from an enumeration

An enumeration cannot be inherited unless it contains no members. The following example works fine because the `Color` enumeration contains no members:

```
class Color(Enum):
    pass
```

```
class RGB(Color):
    RED = 1
    GREEN = 2
    BLUE = 3
```

However, the following example won't work because the `RGB` enumeration has members:

```
class RGBA(RGB):
    ALPHA = 4
```

Error:

```
TypeError: Cannot extend enumerations
```

Python enumeration example

The following example defines an enumeration called `ResponseStatus` :

```
class ResponseStatus(Enum):
    PENDING = 'pending'
    FULFILLED = 'fulfilled'
    REJECTED = 'rejected'
```

Suppose you receive a response from an HTTP request with the following string:

```
response = '''{
    "status":"fulfilled"
}'''
```

And you want to look up the `ResponseStatus` enumeration by the `status` . To do that, you need to convert the response's string to a dictionary and get the value of the status:

```
import json

data = json.loads(response)
status = data['status']
```

And then you look up the member of the `ResponseStatus` enumeration by the status' value:

```
print(ResponseStatus(status))
```

Output:

```
PromiseStatus.FULFILLED
```

Here's the complete program:

```
from enum import Enum
import json


class ResponseStatus(Enum):
    PENDING = 'pending'
    FULFILLED = 'fulfilled'
    REJECTED = 'rejected'


response = '''{
    "status":"fulfilled"
}'''

data = json.loads(response)
status = data['status']
```

```
print(ResponseStatus(status))
```

What if the `status` is not one of the values of the `ResponseStatus` members? then you'll get an error. For example:

```
from enum import Enum
import json


class ResponseStatus(Enum):
    PENDING = 'pending'
    FULFILLED = 'fulfilled'
    REJECTED = 'rejected'


response = '''{
    "status":"ok"
}'''

data = json.loads(response)
status = data['status']

print(ResponseStatus(status))
```

Error:

```
ValueError: 'ok' is not a valid ResponseStatus
```

To catch the exception, you can use the try...except (https://www.pythontutorial.net/python-basics/python-try-except/) statement like the following:

```python
from enum import Enum
import json


class ResponseStatus(Enum):
    PENDING = 'pending'
    FULFILLED = 'fulfilled'
    REJECTED = 'rejected'


response = '''{
    "status":"ok"
}'''

data = json.loads(response)
status = data['status']

try:
    if ResponseStatus(status) is ResponseStatus.FULFILLED:
        print('The request completed successfully')
except ValueError as error:
    print(error)
```

Summary

- An enumeration is a set of members that have associated unique constant values.
- Create a new enumeration by defining a class that inherits from the `Enum` type of the enum module.
- The members have the same types as the enumeration to which they belong.
- Use the `enumeration[member_name]` to access a member by its name and `enumeration(member_value)` to access a member by its value.
- Enumerations are iterable (https://www.pythontutorial.net/python-basics/python-iterables/) .

- Enumeration members are hashable (https://www.pythontutorial.net/python-oop/python-__hash__/) .
- Enumerable are immuable.
- Cannot inherits from an enumeration unless it has no members.