Enum HOWTO

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 7)

Unknown directive type "currentmodule".

```
.. currentmodule:: enum
```

An :class:`Enum` is a set of symbolic names bound to unique values. They are similar to global variables, but they offer a more useful :func:`repr()`, grouping, type-safety, and a few other features.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 9); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 9); *backlink*

Unknown interpreted text role "func".

They are most useful when you have a variable that can take one of a limited selection of values. For example, the days of the week:

```
>>> from enum import Enum
>>> class Weekday(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
```

Or perhaps the RGB primary colors:

```
>>> from enum import Enum
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3
```

As you can see, creating an :class:`Enum` is as simple as writing a class that inherits from :class:`Enum` itself.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 34); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 34); *backlink*

Unknown interpreted text role "class".

Note

Case of Enum Members

Because Enums are used to represent constants we recommend using UPPER_CASE names for members, and will be using that style in our examples.

Depending on the nature of the enum a member's value may or may not be important, but either way that value can be used to get the corresponding member:

```
>>> Weekday(3)
<Weekday.WEDNESDAY: 3>
```

As you can see, the `repr()` of a member shows the enum name, the member name, and the value. The `str()` of a member shows only the enum name and member name:

```
>>> print(Weekday.THURSDAY)
Weekday.THURSDAY
```

The *type* of an enumeration member is the enum it belongs to:

```
>>> type(Weekday.MONDAY)
<enum 'Weekday'>
>>> isinstance(Weekday.FRIDAY, Weekday)
True
```

Enum members have an attribute that contains just their :attr:`name`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 63); *backlink*

Unknown interpreted text role "attr".

```
>>> print(Weekday.TUESDAY.name)
TUESDAY
```

Likewise, they have an attribute for their :attr:`value`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 68); backlink

Unknown interpreted text role "attr".

```
>>> Weekday.WEDNESDAY.value
3
```

Unlike many languages that treat enumerations solely as name/value pairs, Python Enums can have behavior added. For example, :class:`datetime.date` has two methods for returning the weekday: :meth:`weekday` and :meth:`isoweekday`. The difference is that one of them counts from 0-6 and the other from 1-7. Rather than keep track of that ourselves we can add a method to the :class:`Weekday` enum to extract the day from the :class:`date` instance and return the matching enum member:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 74); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 74); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 74); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 74); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 74); backlink

Unknown interpreted text role "class".

```
@classmethod
def from_date(cls, date):
    return cls(date.isoweekday())
```

The complete :class:`Weekday` enum now looks like this:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 86); backlink

Unknown interpreted text role "class".

```
>>> class Weekday(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
...     #
...     @classmethod
...     def from_date(cls, date):
...         return cls(date.isoweekday())
```

Now we can find out what today is! Observe:

```
>>> from datetime import date
>>> Weekday.from_date(date.today())     # doctest: +SKIP
<Weekday.TUESDAY: 2>
```

Of course, if you're reading this on some other day, you'll see that day instead.

This :class:`Weekday` enum is great if our variable only needs one day, but what if we need several? Maybe we're writing a function to plot chores during a week, and don't want to use a :class:`list` -- we could use a different type of :class:`Enum`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 109); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 109); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 109); backlink

Unknown interpreted text role "class".

```
>>> from enum import Flag
>>> class Weekday(Flag):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 4
...     THURSDAY = 8
...     FRIDAY = 16
...     SATURDAY = 32
...     SUNDAY = 64
```

We've changed two things: we're inherited from :class:`Flag`, and the values are all powers of 2.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 124); *backlink*

Unknown interpreted text role "class".

Just like the original :class:`Weekday` enum above, we can have a single selection:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 127); *backlink*

Unknown interpreted text role "class".

```
>>> first_week_day = Weekday.MONDAY
>>> first_week_day
<Weekday.MONDAY: 1>
```

But :class:`Flag` also allows us to combine several members into a single variable:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 133); *backlink*

Unknown interpreted text role "class".

```
>>> weekend = Weekday.SATURDAY | Weekday.SUNDAY
>>> weekend
<Weekday.SATURDAY|SUNDAY: 96>
```

You can even iterate over a :class:`Flag` variable:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 140); *backlink*

Unknown interpreted text role "class".

```
>>> for day in weekend:
...     print(day)
Weekday.SATURDAY
Weekday.SUNDAY
```

Okay, let's get some chores set up:

```
>>> chores_for_ethan = {
...     'feed the cat': Weekday.MONDAY | Weekday.WEDNESDAY | Weekday.FRIDAY,
...     'do the dishes': Weekday.TUESDAY | Weekday.THURSDAY,
...     'answer SO questions': Weekday.SATURDAY,
...     }
```

And a function to display the chores for a given day:

```
>>> def show_chores(chores, day):
...     for chore, days in chores.items():
...         if day in days:
...             print(chore)
>>> show_chores(chores_for_ethan, Weekday.SATURDAY)
answer SO questions
```

In cases where the actual values of the members do not matter, you can save yourself some work and use :func:`auto()` for the values:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 164); *backlink*

Unknown interpreted text role "func".

```
>>> from enum import auto
>>> class Weekday(Flag):
...     MONDAY = auto()
...     TUESDAY = auto()
...     WEDNESDAY = auto()
...     THURSDAY = auto()
...     FRIDAY = auto()
...     SATURDAY = auto()
...     SUNDAY = auto()
```

Programmatic access to enumeration members and their attributes

Sometimes it's useful to access members in enumerations programmatically (i.e. situations where `Color.RED` won't do because the exact color is not known at program-writing time). `Enum` allows such access:

```
>>> Color(1)
<Color.RED: 1>
>>> Color(3)
<Color.BLUE: 3>
```

If you want to access enum members by *name*, use item access:

```
>>> Color['RED']
<Color.RED: 1>
>>> Color['GREEN']
<Color.GREEN: 2>
```

If you have an enum member and need its :attr:`name` or :attr:`value`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 200); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 200); *backlink*

Unknown interpreted text role "attr".

```
>>> member = Color.RED
>>> member.name
'RED'
>>> member.value
1
```

Duplicating enum members and values

Having two enum members with the same name is invalid:

```
>>> class Shape(Enum):
...     SQUARE = 2
...     SQUARE = 3
...
Traceback (most recent call last):
...
TypeError: 'SQUARE' already defined as 2
```

However, an enum member can have other names associated with it. Given two entries `A` and `B` with the same value (and `A` defined first), `B` is an alias for the member `A`. By-value lookup of the value of `A` will return the member `A`. By-name lookup of `A` will return the member `A`. By-name lookup of `B` will also return the member `A`:

```
>>> class Shape(Enum):
...     SQUARE = 2
...     DIAMOND = 1
...     CIRCLE = 3
...     ALIAS_FOR_SQUARE = 2
...
>>> Shape.SQUARE
<Shape.SQUARE: 2>
>>> Shape.ALIAS_FOR_SQUARE
<Shape.SQUARE: 2>
>>> Shape(2)
<Shape.SQUARE: 2>
```

Note

Attempting to create a member with the same name as an already defined attribute (another member, a method, etc.) or attempting to create an attribute with the same name as a member is not allowed.

Ensuring unique enumeration values

By default, enumerations allow multiple names as aliases for the same value. When this behavior isn't desired, you can use the :func:`unique` decorator:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 251); *backlink*

Unknown interpreted text role "func".

```
>>> from enum import Enum, unique
>>> @unique
... class Mistake(Enum):
...     ONE = 1
...     TWO = 2
...     THREE = 3
...     FOUR = 3
...
Traceback (most recent call last):
...
ValueError: duplicate values found in <enum 'Mistake'>: FOUR -> THREE
```

Using automatic values

If the exact value is unimportant you can use :class:`auto`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 270); *backlink*

Unknown interpreted text role "class".

```
>>> from enum import Enum, auto
>>> class Color(Enum):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> [member.value for member in Color]
[1, 2, 3]
```

The values are chosen by :func:`_generate_next_value_`, which can be overridden:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 281); *backlink*

Unknown interpreted text role "func".

```
>>> class AutoName(Enum):
...     def _generate_next_value_(name, start, count, last_values):
...         return name
...
>>> class Ordinal(AutoName):
...     NORTH = auto()
...     SOUTH = auto()
...     EAST = auto()
...     WEST = auto()
...
>>> [member.value for member in Ordinal]
['NORTH', 'SOUTH', 'EAST', 'WEST']
```

Note

The :meth:`_generate_next_value_` method must be defined before any members.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 299); *backlink*

Unknown interpreted text role "meth".

Iteration

Iterating over the members of an enum does not provide the aliases:

```
>>> list(Shape)
[<Shape.SQUARE: 2>, <Shape.DIAMOND: 1>, <Shape.CIRCLE: 3>]
```

The special attribute `__members__` is a read-only ordered mapping of names to members. It includes all names defined in the enumeration, including the aliases:

```
>>> for name, member in Shape.__members__.items():
...     name, member
...
('SQUARE', <Shape.SQUARE: 2>)
('DIAMOND', <Shape.DIAMOND: 1>)
('CIRCLE', <Shape.CIRCLE: 3>)
('ALIAS_FOR_SQUARE', <Shape.SQUARE: 2>)
```

The `__members__` attribute can be used for detailed programmatic access to the enumeration members. For example, finding all the aliases:

```
>>> [name for name, member in Shape.__members__.items() if member.name != name]
['ALIAS_FOR_SQUARE']
```

Comparisons

Enumeration members are compared by identity:

```
>>> Color.RED is Color.RED
True
>>> Color.RED is Color.BLUE
False
>>> Color.RED is not Color.BLUE
True
```

Ordered comparisons between enumeration values are *not* supported. Enum members are not integers (but see IntEnum below):

```
>>> Color.RED < Color.BLUE
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'Color' and 'Color'
```

Equality comparisons are defined though:

```
>>> Color.BLUE == Color.RED
False
>>> Color.BLUE != Color.RED
True
>>> Color.BLUE == Color.BLUE
True
```

Comparisons against non-enumeration values will always compare not equal (again, :class:`IntEnum` was explicitly designed to behave differently, see below):

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 357); *backlink*

Unknown interpreted text role "class".

```
>>> Color.BLUE == 2
False
```

Allowed members and attributes of enumerations

Most of the examples above use integers for enumeration values. Using integers is short and handy (and provided by default by the Functional API), but not strictly enforced. In the vast majority of use-cases, one doesn't care what the actual value of an enumeration is. But if the value *is* important, enumerations can have arbitrary values.

Enumerations are Python classes, and can have methods and special methods as usual. If we have this enumeration:

```
>>> class Mood(Enum):
...     FUNKY = 1
...     HAPPY = 3
...
...     def describe(self):
...         # self is the member here
...         return self.name, self.value
...
...     def __str__(self):
...         return 'my custom str! {0}'.format(self.value)
...
...     @classmethod
...     def favorite_mood(cls):
...         # cls here is the enumeration
...         return cls.HAPPY
...
```

Then:

```
>>> Mood.favorite_mood()
<Mood.HAPPY: 3>
>>> Mood.HAPPY.describe()
('HAPPY', 3)
>>> str(Mood.FUNKY)
'my custom str! 1'
```

The rules for what is allowed are as follows: names that start and end with a single underscore are reserved by enum and cannot be used; all other attributes defined within an enumeration will become members of this enumeration, with the exception of special methods (:meth:`__str__`, :meth:`__add__`, etc.), descriptors (methods are also descriptors), and variable names listed in :attr:`_ignore_`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 403); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 403); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 403); *backlink*

Unknown interpreted text role "attr".

Note: if your enumeration defines :meth:`__new__` and/or :meth:`__init__` then any value(s) given to the enum member will be passed into those methods. See Planet for an example.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 410); *backlink*

Unknown interpreted text role "meth".

Restricted Enum subclassing

A new :class:`Enum` class must have one base enum class, up to one concrete data type, and as many :class:`object`-based mixin classes as needed. The order of these base classes is:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 418); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 418); *backlink*

Unknown interpreted text role "class".

```
class EnumName([mix-in, ...,] [data-type,] base-enum):
    pass
```

Also, subclassing an enumeration is allowed only if the enumeration does not define any members. So this is forbidden:

```
>>> class MoreColor(Color):
...     PINK = 17
...
Traceback (most recent call last):
...
TypeError: <enum 'MoreColor'> cannot extend <enum 'Color'>
```

But this is allowed:

```
>>> class Foo(Enum):
...     def some_behavior(self):
...         pass
...
>>> class Bar(Foo):
...     HAPPY = 1
...     SAD = 2
...
```

Allowing subclassing of enums that define members would lead to a violation of some important invariants of types and instances. On the other hand, it makes sense to allow sharing some common behavior between a group of enumerations. (See OrderedEnum for an example.)

Pickling

Enumerations can be pickled and unpickled:

```
>>> from test.test_enum import Fruit
>>> from pickle import dumps, loads
>>> Fruit.TOMATO is loads(dumps(Fruit.TOMATO))
True
```

The usual restrictions for pickling apply: picklable enums must be defined in the top level of a module, since unpickling requires them to be importable from that module.

> **Note**
>
> With pickle protocol version 4 it is possible to easily pickle enums nested in other classes.

It is possible to modify how enum members are pickled/unpickled by defining :meth:`__reduce_ex__` in the enumeration class.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 471); *backlink*
>
> Unknown interpreted text role "meth".

Functional API

The :class:`Enum` class is callable, providing the following functional API:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 478); *backlink*
>
> Unknown interpreted text role "class".

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG')
>>> Animal
<enum 'Animal'>
>>> Animal.ANT
<Animal.ANT: 1>
>>> list(Animal)
[<Animal.ANT: 1>, <Animal.BEE: 2>, <Animal.CAT: 3>, <Animal.DOG: 4>]
```

The semantics of this API resemble :class:`~collections.namedtuple`. The first argument of the call to :class:`Enum` is the name of the enumeration.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 488); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 488); *backlink*
>
> Unknown interpreted text role "class".

The second argument is the *source* of enumeration member names. It can be a whitespace-separated string of names, a sequence of names, a sequence of 2-tuples with key/value pairs, or a mapping (e.g. dictionary) of names to values. The last two options enable assigning arbitrary values to enumerations; the others auto-assign increasing integers starting with 1 (use the `start` parameter to specify a different starting value). A new class derived from :class:`Enum` is returned. In other words, the above assignment to :class:`Animal` is equivalent to:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 491); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 491); *backlink*

Unknown interpreted text role "class".

```
>>> class Animal(Enum):
...     ANT = 1
...     BEE = 2
...     CAT = 3
...     DOG = 4
...
```

The reason for defaulting to `1` as the starting number and not `0` is that `0` is `False` in a boolean sense, but by default enum members all evaluate to `True`.

Pickling enums created with the functional API can be tricky as frame stack implementation details are used to try and figure out which module the enumeration is being created in (e.g. it will fail if you use a utility function in a separate module, and also may not work on IronPython or Jython). The solution is to specify the module name explicitly as follows:

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG', module=__name__)
```

Warning

If `module` is not supplied, and Enum cannot determine what it is, the new Enum members will not be unpicklable; to keep errors closer to the source, pickling will be disabled.

The new pickle protocol 4 also, in some circumstances, relies on :attr:`~definition.__qualname__` being set to the location where pickle will be able to find the class. For example, if the class was made available in class SomeData in the global scope:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 525); *backlink*

Unknown interpreted text role "attr".

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG', qualname='SomeData.Animal')
```

The complete signature is:

```
Enum(
    value='NewEnumName',
    names=<...>,
    *,
    module='...',
    qualname='...',
    type=<mixed-in class>,
    start=1,
    )
```

value: What the new enum class will record as its name.

names: The enum members. This can be a whitespace- or comma-separated string (values will start at 1 unless otherwise specified):

```
'RED GREEN BLUE' | 'RED,GREEN,BLUE' | 'RED, GREEN, BLUE'
```

or an iterator of names:

```
['RED', 'GREEN', 'BLUE']
```

or an iterator of (name, value) pairs:

```
[('CYAN', 4), ('MAGENTA', 5), ('YELLOW', 6)]
```

or a mapping:

```
{'CHARTREUSE': 7, 'SEA_GREEN': 11, 'ROSEMARY': 42}
```

module: name of module where new enum class can be found.

qualname: where in module new enum class can be found.

type: type to mix in to new enum class.

start: number to start counting at if only names are passed in.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 571)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
   The *start* parameter was added.
```

Derived Enumerations

IntEnum

The first variation of :class:`Enum` that is provided is also a subclass of :class:`int`. Members of an :class:`IntEnum` can be compared to integers; by extension, integer enumerations of different types can also be compared to each other:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 581); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 581); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 581); *backlink*

Unknown interpreted text role "class".

```
>>> from enum import IntEnum
>>> class Shape(IntEnum):
...     CIRCLE = 1
...     SQUARE = 2
...
>>> class Request(IntEnum):
...     POST = 1
...     GET = 2
...
>>> Shape == 1
False
>>> Shape.CIRCLE == 1
True
>>> Shape.CIRCLE == Request.POST
True
```

However, they still can't be compared to standard :class:`Enum` enumerations:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 602); *backlink*

Unknown interpreted text role "class".

```
>>> class Shape(IntEnum):
...     CIRCLE = 1
...     SQUARE = 2
...
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...
>>> Shape.CIRCLE == Color.RED
False
```

:class:`IntEnum` values behave like integers in other ways you'd expect:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 615); *backlink*

Unknown interpreted text role "class".

```
>>> int(Shape.CIRCLE)
1
>>> ['a', 'b', 'c'][Shape.CIRCLE]
'b'
>>> [i for i in range(Shape.SQUARE)]
[0, 1]
```

StrEnum

The second variation of :class:`Enum` that is provided is also a subclass of :class:`str`. Members of a :class:`StrEnum` can be compared to strings; by extension, string enumerations of different types can also be compared to each other.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 628); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 628); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 628); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 633)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

IntFlag

The next variation of :class:`Enum` provided, :class:`IntFlag`, is also based on :class:`int`. The difference being :class:`IntFlag` members can be combined using the bitwise operators (&, |, ^, ~) and the result is still an :class:`IntFlag` member, if possible. Like :class:`IntEnum`, :class:`IntFlag` members are also integers and can be used wherever an :class:`int` is used.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 639); *backlink*

Unknown interpreted text role "class".

Note

Any operation on an :class:`IntFlag` member besides the bit-wise operations will lose the :class:`IntFlag` membership.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 647); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 647); *backlink*

Unknown interpreted text role "class".

Bit-wise operations that result in invalid :class:`IntFlag` values will lose the :class:`IntFlag` membership. See :class:`FlagBoundary` for details.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 650); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 650); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 650); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 654)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 655)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
```

Sample :class:`IntFlag` class:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 657); *backlink*

Unknown interpreted text role "class".

```
>>> from enum import IntFlag
>>> class Perm(IntFlag):
...     R = 4
...     W = 2
...     X = 1
...
>>> Perm.R | Perm.W
<Perm.R|W: 6>
>>> Perm.R + Perm.W
6
>>> RW = Perm.R | Perm.W
>>> Perm.R in RW
True
```

It is also possible to name the combinations:

```
>>> class Perm(IntFlag):
...     R = 4
...     W = 2
...     X = 1
...     RWX = 7
>>> Perm.RWX
<Perm.RWX: 7>
>>> ~Perm.RWX
<Perm: 0>
>>> Perm(7)
<Perm.RWX: 7>
```

Note

Named combinations are considered aliases. Aliases do not show up during iteration, but can be returned from by-value lookups.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 692)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
```

Another important difference between :class:`IntFlag` and :class:`Enum` is that if no flags are set (the value is 0), its boolean evaluation is :data:`False`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 694); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 694); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 694); *backlink*

Unknown interpreted text role "data".

```
>>> Perm.R & Perm.X
<Perm: 0>
>>> bool(Perm.R & Perm.X)
False
```

Because :class:`IntFlag` members are also subclasses of :class:`int` they can be combined with them (but may lose :class:`IntFlag`

membership:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 702); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 702); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 702); *backlink*

Unknown interpreted text role "class".

```
>>> Perm.X | 4
<Perm.R|X: 5>

>>> Perm.X | 8
9
```

Note

The negation operator, ~, always returns an :class:`IntFlag` member with a positive value:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 713); *backlink*

Unknown interpreted text role "class".

```
>>> (~Perm.X).value == (Perm.R|Perm.W).value == 6
True
```

:class:`IntFlag` members can also be iterated over:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 719); *backlink*

Unknown interpreted text role "class".

```
>>> list(RW)
[<Perm.R: 4>, <Perm.W: 2>]
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 724)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

Flag

The last variation is :class:`Flag`. Like :class:`IntFlag`, :class:`Flag` members can be combined using the bitwise operators (&, |, ^, ~). Unlike :class:`IntFlag`, they cannot be combined with, nor compared against, any other :class:`Flag` enumeration, nor :class:`int`. While it is possible to specify the values directly it is recommended to use :class:`auto` as the value and let :class:`Flag` select an appropriate value.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 737)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6
```

Like :class:`IntFlag`, if a combination of :class:`Flag` members results in no flags being set, the boolean evaluation is :data:`False`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 739); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 739); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 739); *backlink*

Unknown interpreted text role "data".

```
>>> from enum import Flag, auto
>>> class Color(Flag):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.RED & Color.GREEN
<Color: 0>
>>> bool(Color.RED & Color.GREEN)
False
```

Individual flags should have values that are powers of two (1, 2, 4, 8, ...), while combinations of flags won't:

```
>>> class Color(Flag):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...     WHITE = RED | BLUE | GREEN
...
>>> Color.WHITE
<Color.WHITE: 7>
```

Giving a name to the "no flags set" condition does not change its boolean value:

```
>>> class Color(Flag):
...     BLACK = 0
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.BLACK
<Color.BLACK: 0>
>>> bool(Color.BLACK)
False
```

:class:`Flag` members can also be iterated over:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 779); *backlink*

Unknown interpreted text role "class".

```
>>> purple = Color.RED | Color.BLUE
>>> list(purple)
[<Color.RED: 1>, <Color.BLUE: 2>]
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 785)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

Note

For the majority of new code, :class:`Enum` and :class:`Flag` are strongly recommended, since :class:`IntEnum` and :class:`IntFlag` break some semantic promises of an enumeration (by being comparable to integers, and thus by transitivity to other unrelated enumerations). :class:`IntEnum` and :class:`IntFlag` should be used only in cases where :class:`Enum` and :class:`Flag` will not do; for example, when integer constants are replaced with enumerations, or for interoperability with other systems.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 789); *backlink*

Unknown interpreted text role "class".

Others

While :class:`IntEnum` is part of the :mod:`enum` module, it would be very simple to implement independently:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 801); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst`, line 801); *backlink*

Unknown interpreted text role "mod".

```
class IntEnum(int, Enum):
    pass
```

This demonstrates how similar derived enumerations can be defined; for example a :class:`FloatEnum` that mixes in :class:`float` instead of :class:`int`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 807); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 807); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 807); backlink

Unknown interpreted text role "class".

Some rules:

1. When subclassing :class:`Enum`, mix-in types must appear before :class:`Enum` itself in the sequence of bases, as in the :class:`IntEnum` example above.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 812); backlink

   Unknown interpreted text role "class".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 812); backlink

   Unknown interpreted text role "class".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 812); backlink

   Unknown interpreted text role "class".

2. Mix-in types must be subclassable. For example, :class:`bool` and :class:`range` are not subclassable and will throw an error during Enum creation if used as the mix-in type.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 815); backlink

   Unknown interpreted text role "class".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 815); backlink

   Unknown interpreted text role "class".

3. While :class:`Enum` can have members of any type, once you mix in an additional type, all the members must have values of that type, e.g. :class:`int` above. This restriction does not apply to mix-ins which only add methods and don't specify another type.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 818); backlink

   Unknown interpreted text role "class".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 818); backlink

   Unknown interpreted text role "class".

4. When another data type is mixed in, the :attr:`value` attribute is *not the same* as the enum member itself, although it is equivalent and will compare equal.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 822); backlink

   Unknown interpreted text role "attr".

5. %-style formatting: `%s` and `%r` call the :class:`Enum` class's :meth:`__str__` and :meth:`__repr__` respectively; other codes (such as `%i` or `%h` for IntEnum) treat the enum member as its mixed-in type.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 825); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 825); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 825); *backlink*

Unknown interpreted text role "meth".

6. :ref:`Formatted string literals <f-strings>`, :meth:`str.format`, and :func:`format` will use the enum's :meth:`__str__` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 828); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 828); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 828); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 828); *backlink*

Unknown interpreted text role "meth".

Note

Because :class:`IntEnum`, :class:`IntFlag`, and :class:`StrEnum` are designed to be drop-in replacements for existing constants, their :meth:`__str__` method has been reset to their data types :meth:`__str__` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 833); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 833); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 833); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 833); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 833); *backlink*

Unknown interpreted text role "meth".

When to use :meth:`__new__` vs. :meth:`__init__`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 838); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 838); backlink

Unknown interpreted text role "meth".

:meth:`__new__` must be used whenever you want to customize the actual value of the :class:`Enum` member. Any other modifications may go in either :meth:`__new__` or :meth:`__init__`, with :meth:`__init__` being preferred.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 841); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 841); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 841); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 841); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 841); backlink

Unknown interpreted text role "meth".

For example, if you want to pass several items to the constructor, but only want one of them to be the value:

```
>>> class Coordinate(bytes, Enum):
...     """
...     Coordinate with binary codes that can be indexed by the int code.
...     """
...     def __new__(cls, value, label, unit):
...         obj = bytes.__new__(cls, [value])
...         obj._value_ = value
...         obj.label = label
...         obj.unit = unit
...         return obj
...     PX = (0, 'P.X', 'km')
...     PY = (1, 'P.Y', 'km')
...     VX = (2, 'V.X', 'km/s')
...     VY = (3, 'V.Y', 'km/s')
...

>>> print(Coordinate['PY'])
Coordinate.PY

>>> print(Coordinate(3))
Coordinate.VY
```

Finer Points

Supported `__dunder__` names

:attr:`__members__` is a read-only ordered mapping of `member_name`:`member` items. It is only available on the class.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 877); backlink

Unknown interpreted text role "attr".

:meth:`__new__`, if specified, must create and return the enum members; it is also a very good idea to set the member's :attr:`_value_` appropriately. Once all the members are created it is no longer used.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 880); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 880); backlink

Unknown interpreted text role "attr".

Supported `_sunder_` names

- `_name_` -- name of the member

- `_value_` -- value of the member; can be set / modified in `__new__`
- `_missing_` -- a lookup function used when a value is not found; may be overridden
- `_ignore_` -- a list of names, either as a :class:`list` or a :class:`str`, that will not be transformed into members, and will be removed from the final class

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 893); *backlink*

  Unknown interpreted text role "class".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 893); *backlink*

  Unknown interpreted text role "class".

- `_order_` -- used in Python 2/3 code to ensure member order is consistent (class attribute, removed during class creation)
- `_generate_next_value_` -- used by the Functional API and by :class:`auto` to get an appropriate value for an enum member; may be overridden

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 898); *backlink*

  Unknown interpreted text role "class".

Note

For standard :class:`Enum` classes the next value chosen is the last value seen incremented by one.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 904); *backlink*

Unknown interpreted text role "class".

For :class:`Flag` classes the next value chosen will be the next highest power-of-two, regardless of the last value seen.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 907); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 910)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6 ``_missing_``, ``_order_``, ``_generate_next_value_``
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 911)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7 ``_ignore_``
```

To help keep Python 2 / Python 3 code in sync an :attr:`_order_` attribute can be provided. It will be checked against the actual order of the enumeration and raise an error if the two do not match:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 913); *backlink*

Unknown interpreted text role "attr".

```
>>> class Color(Enum):
...     _order_ = 'RED GREEN BLUE'
...     RED = 1
...     BLUE = 3
...     GREEN = 2
...
Traceback (most recent call last):
...
TypeError: member order does not match _order_:
  ['RED', 'BLUE', 'GREEN']
  ['RED', 'GREEN', 'BLUE']
```

Note

In Python 2 code the :attr:`_order_` attribute is necessary as definition order is lost before it can be recorded.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 931); *backlink*

Unknown interpreted text role "attr".

_Private__names

:ref:`Private names <private-name-mangling>` are not converted to enum members, but remain normal attributes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 938); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 941)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
```

`Enum` member type

Enum members are instances of their enum class, and are normally accessed as `EnumClass.member`. In Python versions `3.5` to `3.10` you could access members from other members -- this practice was discouraged, and in `3.11` :class:`Enum` returns to not allowing it:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 947); *backlink*

Unknown interpreted text role "class".

```
>>> class FieldTypes(Enum):
...     name = 0
...     value = 1
...     size = 2
...
>>> FieldTypes.value.size
Traceback (most recent call last):
...
AttributeError: <enum 'FieldTypes'> member has no attribute 'size'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 963)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 964)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
```

Creating members that are mixed with other data types

When subclassing other data types, such as :class:`int` or :class:`str`, with an :class:`Enum`, all values after the `=` are passed to that data type's constructor. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 970); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 970); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 970); *backlink*

Unknown interpreted text role "class".

```
>>> class MyEnum(IntEnum):      # help(int) -> int(x, base=10) -> integer
...     example = '11', 16      # so x='11' and base=16
...
>>> MyEnum.example.value        # and hex(11) is...
```

```
17
```

Boolean value of `Enum` classes and members

Enum classes that are mixed with non-:class:`Enum` types (such as :class:`int`, :class:`str`, etc.) are evaluated according to the mixed-in type's rules; otherwise, all members evaluate as :data:`True`. To make your own enum's boolean evaluation depend on the member's value add the following to your class:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 984); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 984); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 984); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 984); *backlink*

Unknown interpreted text role "data".

```
def __bool__(self):
    return bool(self.value)
```

Plain :class:`Enum` classes always evaluate as :data:`True`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 993); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 993); *backlink*

Unknown interpreted text role "data".

`Enum` classes with methods

If you give your enum subclass extra methods, like the Planet class below, those methods will show up in a :func:`dir` of the member, but not of the class:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 999); *backlink*

Unknown interpreted text role "func".

```
>>> dir(Planet)                         # doctest: +SKIP
['EARTH', 'JUPITER', 'MARS', 'MERCURY', 'NEPTUNE', 'SATURN', 'URANUS', 'VENUS', '__class__', '__doc__', '__members__',
>>> dir(Planet.EARTH)                   # doctest: +SKIP
['__class__', '__doc__', '__module__', 'mass', 'name', 'radius', 'surface_gravity', 'value']
```

Combining members of `Flag`

Iterating over a combination of :class:`Flag` members will only return the members that are comprised of a single bit:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1012); *backlink*

Unknown interpreted text role "class".

```
>>> class Color(Flag):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...     MAGENTA = RED | BLUE
...     YELLOW = RED | GREEN
...     CYAN = GREEN | BLUE
...
>>> Color(3)  # named combination
<Color.YELLOW: 3>
>>> Color(7)      # not named combination
<Color.RED|GREEN|BLUE: 7>
```

`Flag` and `IntFlag` minutia

Using the following snippet for our examples:

```
>>> class Color(IntFlag):
...     BLACK = 0
...     RED = 1
...     GREEN = 2
...     BLUE = 4
...     PURPLE = RED | BLUE
```

```
...     WHITE = RED | GREEN | BLUE
...
```

the following are true:

- single-bit flags are canonical
- multi-bit and zero-bit flags are aliases
- only canonical flags are returned during iteration:

```
>>> list(Color.WHITE)
[<Color.RED: 1>, <Color.GREEN: 2>, <Color.BLUE: 4>]
```

- negating a flag or flag set returns a new flag/flag set with the corresponding positive integer value:

```
>>> Color.BLUE
<Color.BLUE: 4>

>>> ~Color.BLUE
<Color.RED|GREEN: 3>
```

- names of pseudo-flags are constructed from their members' names:

```
>>> (Color.RED | Color.GREEN).name
'RED|GREEN'
```

- multi-bit flags, aka aliases, can be returned from operations:

```
>>> Color.RED | Color.BLUE
<Color.PURPLE: 5>

>>> Color(7)  # or Color(-1)
<Color.WHITE: 7>

>>> Color(0)
<Color.BLACK: 0>
```

- membership / containment checking: zero-valued flags are always considered to be contained:

```
>>> Color.BLACK in Color.WHITE
True
```

  otherwise, only if all bits of one flag are in the other flag will True be returned:

```
>>> Color.PURPLE in Color.WHITE
True

>>> Color.GREEN in Color.PURPLE
False
```

There is a new boundary mechanism that controls how out-of-range / invalid bits are handled: `STRICT`, `CONFORM`, `EJECT`, and `KEEP`:

- STRICT --> raises an exception when presented with invalid values
- CONFORM --> discards any invalid bits
- EJECT --> lose Flag status and become a normal int with the given value
- KEEP --> keep the extra bits
  - keeps Flag status and extra bits
  - extra bits do not show up in iteration
  - extra bits do show up in repr() and str()

The default for Flag is `STRICT`, the default for `IntFlag` is `EJECT`, and the default for `_convert_` is `KEEP` (see `ssl.Options` for an example of when `KEEP` is needed).

How are Enums different?

Enums have a custom metaclass that affects many aspects of both derived :class:`Enum` classes and their instances (members).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1113); *backlink***
>
> Unknown interpreted text role "class".

Enum Classes

The :class:`EnumType` metaclass is responsible for providing the :meth:`__contains__`, :meth:`__dir__`, :meth:`__iter__` and other methods that allow one to do things with an :class:`Enum` class that fail on a typical class, such as *list(Color)* or *some_enum_var in Color*. :class:`EnumType` is responsible for ensuring that various other methods on the final :class:`Enum` class are correct (such as :meth:`__new__`, :meth:`__getnewargs__`, :meth:`__str__` and :meth:`__repr__`).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink***
>
> Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1120); *backlink*

Unknown interpreted text role "meth".

Enum Members (aka instances)

The most interesting thing about enum members is that they are singletons. :class:`EnumType` creates them all while it is creating the enum class itself, and then puts a custom :meth:`__new__` in place to ensure that no new ones are ever instantiated by returning only the existing member instances.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1132); *backlink*

Unknown interpreted text role "meth".

While :class:`Enum`, :class:`IntEnum`, :class:`StrEnum`, :class:`Flag`, and :class:`IntFlag` are expected to cover the majority of use-cases, they cannot cover them all. Here are recipes for some different types of enumerations that can be used directly, or as examples for creating one's own.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1141); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1141); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1141); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1141); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1141); *backlink*

Unknown interpreted text role "class".

Omitting values

In many use-cases, one doesn't care what the actual value of an enumeration is. There are several ways to define this type of simple enumeration:

- use instances of :class:`auto` for the value

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1153); *backlink*

  Unknown interpreted text role "class".

- use instances of :class:`object` as the value

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1154); *backlink*

  Unknown interpreted text role "class".

- use a descriptive string as the value
- use a tuple as the value and a custom :meth:`__new__` to replace the tuple with an :class:`int` value

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1156); *backlink*

  Unknown interpreted text role "meth".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1156); *backlink*

  Unknown interpreted text role "class".

Using any of these methods signifies to the user that these values are not important, and also enables one to add, remove, or reorder members without having to renumber the remaining members.

Using :class:`auto`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1164); *backlink*

Unknown interpreted text role "class".

Using :class:`auto` would look like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1167); *backlink*

Unknown interpreted text role "class".

```
>>> class Color(Enum):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.GREEN
<Color.GREEN: 3>
```

Using :class:`object`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1178); *backlink*

Unknown interpreted text role "class".

Using :class:`object` would look like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1181); *backlink*

Unknown interpreted text role "class".

```
>>> class Color(Enum):
...     RED = object()
...     GREEN = object()
...     BLUE = object()
...
>>> Color.GREEN                         # doctest: +SKIP
<Color.GREEN: <object object at 0x...>>
```

This is also a good example of why you might want to write your own :meth:`__repr__`:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1191); *backlink***
>
> Unknown interpreted text role "meth".

```
>>> class Color(Enum):
...     RED = object()
...     GREEN = object()
...     BLUE = object()
...     def __repr__(self):
...         return "<%s.%s>" % (self.__class__.__name__, self._name_)
...
>>> Color.GREEN
<Color.GREEN>
```

Using a descriptive string

Using a string as the value would look like:

```
>>> class Color(Enum):
...     RED = 'stop'
...     GREEN = 'go'
...     BLUE = 'too fast!'
...
>>> Color.GREEN
<Color.GREEN: 'go'>
```

Using a custom :meth:`__new__`

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1220); *backlink***
>
> Unknown interpreted text role "meth".

Using an auto-numbering :meth:`__new__` would look like:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1223); *backlink***
>
> Unknown interpreted text role "meth".

```
>>> class AutoNumber(Enum):
...     def __new__(cls):
...         value = len(cls.__members__) + 1
...         obj = object.__new__(cls)
...         obj._value_ = value
...         return obj
...
>>> class Color(AutoNumber):
...     RED = ()
...     GREEN = ()
...     BLUE = ()
...
>>> Color.GREEN
<Color.GREEN: 2>
```

To make a more general purpose `AutoNumber`, add `*args` to the signature:

```
>>> class AutoNumber(Enum):
...     def __new__(cls, *args):      # this is the only change from above
...         value = len(cls.__members__) + 1
...         obj = object.__new__(cls)
...         obj._value_ = value
...         return obj
...
```

Then when you inherit from `AutoNumber` you can write your own `__init__` to handle any extra arguments:

```
>>> class Swatch(AutoNumber):
...     def __init__(self, pantone='unknown'):
...         self.pantone = pantone
...     AUBURN = '3497'
...     SEA_GREEN = '1246'
...     BLEACHED_CORAL = () # New color, no Pantone code yet!
...
>>> Swatch.SEA_GREEN
<Swatch.SEA_GREEN: 2>
>>> Swatch.SEA_GREEN.pantone
'1246'
>>> Swatch.BLEACHED_CORAL.pantone
'unknown'
```

> **Note**
>
> The :meth:`__new__` method, if defined, is used during creation of the Enum members; it is then replaced by Enum's :meth:`__new__` which is used after class creation for lookup of existing members.
>
> > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1269); *backlink***
> >
> > Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1269); *backlink*

Unknown interpreted text role "meth".

OrderedEnum

An ordered enumeration that is not based on :class:`IntEnum` and so maintains the normal :class:`Enum` invariants (such as not being comparable to other enumerations):

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1277); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1277); *backlink*

Unknown interpreted text role "class".

```
>>> class OrderedEnum(Enum):
...     def __ge__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value >= other.value
...         return NotImplemented
...     def __gt__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value > other.value
...         return NotImplemented
...     def __le__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value <= other.value
...         return NotImplemented
...     def __lt__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value < other.value
...         return NotImplemented
...
>>> class Grade(OrderedEnum):
...     A = 5
...     B = 4
...     C = 3
...     D = 2
...     F = 1
...
>>> Grade.C < Grade.A
True
```

DuplicateFreeEnum

Raises an error if a duplicate member name is found instead of creating an alias:

```
>>> class DuplicateFreeEnum(Enum):
...     def __init__(self, *args):
...         cls = self.__class__
...         if any(self.value == e.value for e in cls):
...             a = self.name
...             e = cls(self.value).name
...             raise ValueError(
...                 "aliases not allowed in DuplicateFreeEnum:  %r --> %r"
...                 % (a, e))
...
>>> class Color(DuplicateFreeEnum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3
...     GRENE = 2
...
Traceback (most recent call last):
...
ValueError: aliases not allowed in DuplicateFreeEnum:  'GRENE' --> 'GREEN'
```

Note

This is a useful example for subclassing Enum to add or change other behaviors as well as disallowing aliases. If the only desired change is disallowing aliases, the :func:`unique` decorator can be used instead.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1338); *backlink*

Unknown interpreted text role "func".

Planet

If :meth:`__new__` or :meth:`__init__` is defined, the value of the enum member will be passed to those methods:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1346); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1346); *backlink*

Unknown interpreted text role "meth".

```
>>> class Planet(Enum):
...     MERCURY = (3.303e+23, 2.4397e6)
...     VENUS   = (4.869e+24, 6.0518e6)
...     EARTH   = (5.976e+24, 6.37814e6)
...     MARS    = (6.421e+23, 3.3972e6)
...     JUPITER = (1.9e+27,   7.1492e7)
...     SATURN  = (5.688e+26, 6.0268e7)
...     URANUS  = (8.686e+25, 2.5559e7)
...     NEPTUNE = (1.024e+26, 2.4746e7)
...     def __init__(self, mass, radius):
...         self.mass = mass       # in kilograms
...         self.radius = radius   # in meters
...     @property
...     def surface_gravity(self):
...         # universal gravitational constant  (m3 kg-1 s-2)
...         G = 6.67300E-11
...         return G * self.mass / (self.radius * self.radius)
...
>>> Planet.EARTH.value
(5.976e+24, 6378140.0)
>>> Planet.EARTH.surface_gravity
9.802652743337129
```

TimePeriod

An example to show the :attr:`_ignore_` attribute in use:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1377); *backlink*

Unknown interpreted text role "attr".

```
>>> from datetime import timedelta
>>> class Period(timedelta, Enum):
...     "different lengths of time"
...     _ignore_ = 'Period i'
...     Period = vars()
...     for i in range(367):
...         Period['day_%d' % i] = i
...
>>> list(Period)[:2]
[<Period.day_0: datetime.timedelta(0)>, <Period.day_1: datetime.timedelta(days=1)>]
>>> list(Period)[-2:]
[<Period.day_365: datetime.timedelta(days=365)>, <Period.day_366: datetime.timedelta(days=366)>]
```

Subclassing EnumType

While most enum needs can be met by customizing :class:`Enum` subclasses, either with class decorators or custom functions, :class:`EnumType` can be subclassed to provide a different Enum experience.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1398); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]enum.rst, line 1398); *backlink*

Unknown interpreted text role "class".