Topic: Python PEP 435 - Addition of the Enum type to the Python Standard Library

Members: Jonathan Song, Gabriela Mendoza, Thomas Wagner
Team: Song Mendoza Wagner

Intro

- An enumeration is defined by symbolic names that are bound to unique, constant values.
- Enumeration member is distinct from each other in name and in value and provides the programmer with unique, built-in methods to iterate .

Discussions about PEP

- Python mail, it was approved by Guido van Rossum.
- Special interest in IntEnum
- types of members such as Direction.up will always be of type Direction.

Terminology and concepts used throughout the course.

- Type matching/Type checking
- Immutable variables
- Alias Members

Type Checking

- Python is dynamically typed: Type checking of the program occurs as the program is being evaluated.
- When adding enumeration members to an Enum type, each of the members must match the type of the enumeration itself when they are used

- Enumeration members must follow the rules specified in their particular type environments, which are specified by their Enum types
- If expression is not well typed, throw typeerror to stop program from evaluating values any further

- “typeerror” token
- dynamic type error if it should reach some ill-typed expression.
- will not allow the program to finish evaluating due to this error.
- the Enum type is used to define unique, related sets of constant values

Immutable Variables

- Another way of stating this, is that when enumeration members are defined, they become immutable variables.
- Immutable variables: When a variable is defined and is set to value, then the variable cannot be re-set to a different value in the same environment

- We also cannot have two Enum members that have the same name or same value
- This would potentially go against the standards of immutability in the Enum type

Alias Members

- For example, something like this:

- from enum import Enum

```
class Color(Enum):
      red = 0
      white = 1
      black = 2
      black = 3
```

- Supports alias members if two members are set to the same value
- From enum import Enum

```
class Color(Enum):
    red = 1
    white = 2
    black = 2
```

- Because of this, the above code can also be rewritten as:

- from enum import Enum

```
class Color(Enum):
    red = 1
    white = 2
    alias_for_white = 2
```

- Alias members of an enumeration are treated differently

Is this change useful?

- first proposed in PEP 354, but was rejected
- Guido van Rossum
- There were already plenty “cookbook recipes, and PyPI packages”
- flufl.enum package
- “enums are not that useful in small programs.”

- Before the introduction of the Enum type, enumerations could be pseudo-implemented using classes:

- class Directions:

```
    up = 0
    down = 1
    left = 2
    right =3
```

- Enum type ensures that the values of each enumeration member are each distinct from each other.
- Changing the above Direction class to an Enum type is simple and does not require much more code:
- from enum import Enum
-

```
class Directions(Enum):
    up = 0
    down = 1
    left = 2
    right = 3
```

Basic Use

- Enum(name of enumeration, source of enumeration member names).
- A call such as this is of the form:

Animals = Enum(‘Animals’, ‘ant bee cat dog’, module = __name__)

IntEnum Type

The IntEnum subclass is defined similarly to the Enum class and allows comparisons with integer types:

```
from enum import IntEnum
```

Offers greater interoperability with integers

```
from enum import IntEnum

class VerticalDirections(IntEnum):
     up = 1
    down = 2

class HorizontalDirections(IntEnum):
    left = 1
    right = 2
```

Do not have to specify values: offers further convenience to programmer for repetitive tasks

```
from enum import Enum

class Directions(Enum):
   up, down, left, right
```

Conclusion

- well-typed, immutable variables
- The Enum type has many features that ensures the safety and convenience of the programmer
- applies concepts such as type checking and alias members to validate the correct usage of the type.
- Enum type has proved very useful in the way that it has increased the interoperability of many variable types