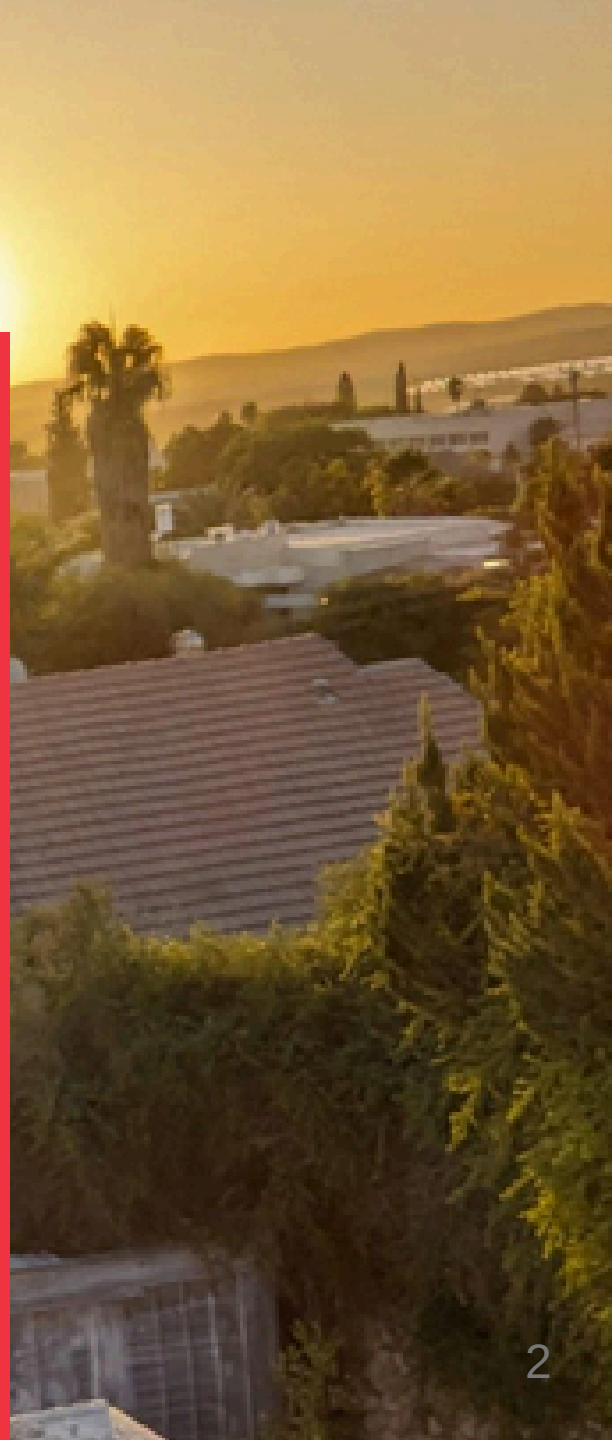


Beyond Constants: Mastering Python Enums



Tsvi Mostovicz, Intel | Pycon IL 2025 | Cinema City Glilot, Israel

Bio





What This Talk Is About

Two development stories ...

-  The hdate library (with real examples)
-  An internal Intel library (with all the secret sauce taken out 😊)

about ...

-  How Enums improved our code (with some cool tricks)
-  The late night debugging of our own stupidity

Part I - The story of the hdate library

Or "Why should I use Enums? 🤔"

This month shall mark for you the beginning of the months (Exodus 12:2)

- The hdate library started off as a Python port of some C-code back in April 2016.

```
>>> from datetime import date

>>> today = HDate(date(2016, 4, 26))
>>> today.get_hebrew_date()
(10, 1, 5776)
```

- 💡 Using numbers is not very user friendly to the user

```
>>> str(today)
"Monday 10 Nissan 5776"
```

But what about the programmer?

Guess what the following does?

```
if date.month == 13:  
    month = 12  
if date.month == 14:  
    month = 12  
    day += 30
```

Even better: debugging test code 🐈

A snippet from our tests codebase from 6 years ago

```
@pytest.mark.parametrize(("date", "holiday"), [
    ((21, 1), "pesach_vii"),
    ((6, 3), "shavuot"),
    ((25, 9), "chanukah"),
])
def test_holidays(date, holiday):
    ...
```

Not really friendly when debugging. 😞

Hey, we should use enums

A month is literally an enumerated type

```
class Months(Enum):  
    TISHREI = auto()  
    CHESHVAN = auto()  
    KISLEV = auto()  
    TEVET = auto()
```


Available since Python 3.4 (That's more than 10 years ago 😊)

Incrementing dates

 Our goal:

```
HebrewDate(5785, Months.AV, 7) + timedelta(days=35)
```

A simplified `__add__` method

```
1. def __add__(self, other: timedelta):
2.     _year, _month, _day = self.year, self.month, self.day
3.     days_remaining = other.days
4.
5.     while days_remaining > 0:
6.         days_left_in_month = get_month_length(_month, _year) - ...
7.
8.         if days_remaining > days_left_in_month:
9.             _month = get_next_month(_month, _year)
10.            ...
11.
12.     return HebrewDate(_year, _month, _day)
```

Enums are classes (and can have methods)

```
1. class Months(Enum):
2.
3.     def next_month(self, year) -> Months:
4.         """Return the next month."""
5.         if self == Months.ELUL:
6.             return Months.TISHREI
7.         if self in {Months.ADAR, Months.ADAR_II}:
8.             return Months.NISAN
9.         if is_leap_year(year) and self == Months.SHVAT:
10.            return Months.ADAR_I
11.        return Months(self._value_ + 1)
```

... and even attributes

```
1. class Months(Enum):
2.     TISHREI = 1, 30
3.     TEVET = 4, 29
4.
5.     def __new__(cls, value, length):
6.         obj = object.__new__(cls, value)
7.         obj._value_ = value
8.         obj._length = length
9.         return obj
10.
11. # Usage
12. print(Months.TISHREI._length)      # 30
13. print(Months.TEVET.value)          # 4
```

... which can be dynamic

```
1. class Months(Enum):
2.     MARCHESHVAN = 2, lambda year: 30 if long_cheshvan(year) else 29
3.     KISLEV = 3, lambda year: 30 if not short_kislev(year) else 29
4.
5.     def length(self, year = None):
6.         """Return the number of days in this month."""
7.         if callable(self._length):
8.             return self._length(year)
9.         return self._length
```

Part II - Creating Enums dynamically

The (simplified) Intel story: A YAML config with product-specific settings

```
1. - name: "feature_a"  
2.   products: ["process_y"]  
3.  
4. - name: "feature_b"  
5.   products: ["process_x"]  
6.  
7. - name: "debug_mode" # No products -> ALL
```


Problem 🤔

- 📝 Large changes when the manufacturing process changes
- 🐛 Typos in YAML cause silent failures (Non-existent `process_z`)

Solution ⚡

Create a project configuration ...

```
products:  
  - SERVER: process_x  
  - CLIENT: process_y
```

... mapped at runtime to an Enum:

```
with open("config.yaml") as f:  
    project_config = yaml.safe_load(f)  
  
mapping = project_config["products"]  
ProcessConfig = StrEnum("ProcessConfig", mapping)
```

A more streamlined approach

- ★ Automatic validation of process names
- ★ No need to change YAML configuration in multiple locations

🌟 Bonus: type-safety throughout our code

```
@dataclass
class Features:

    processes: list[ProcessConfig]
```

The pitfalls of using enums

Or "How we learned not to do stupid stuff the hard way 🤦"

Example #1: Setting the language

```
>>> today = HebrewDate(5785, Months.ELUL, 7)
>>> today.set_language("he")
>>> str(today)
'ז אלול תשפ"ה'
```

```
>>> tomorrow = HebrewDate(5785, Months.ELUL, 8)
>>> assert tomorrow - today == timedelta(days=1)
True
```

```
>>> str(today)
'ה"ה Elul תשפ' # WAIT... Why did the month change to English??
```

OOPS!

Enums are singletons 🐱

The language attribute of `Months` has been reset when `tomorrow` was created.

Example #2: Test pollution

Sometimes we want "different" Adar's to be considered the same.

```
def test_set_comparison_mode():  
    Months.ADAR.set_comparison_mode(ComparisonMode.ADAR_IS_ADAR_I)  
    assert Months.ADAR == Months.ADAR_I
```

```
def test_compare():  
    assert HebrewDate(5785, Months.ADAR_I, 4) \  
        != HebrewDate(5785, Months.ADAR, 4)
```

When to Use Enum attributes

DO use attributes:

- Behavior belongs to enum member
- Data is constant and well-defined

Examples:

- Pre-defined values (length, position)

DON'T use attributes:

- Attribute state will be modified during runtime
- Behavior dependent on context

Examples:

- Storing preferences

When to Use Dynamic Enums

Perfect for:

- Config that varies between runs
- External data sources

Examples:

- Product SKUs from files
- API endpoints (dev/staging/prod)

Not suitable for:

- Values changing during execution

Examples:

- Runtime feature toggles that can be switched

"Simple is better than complex. Complex is better than complicated."
— The Zen of Python (PEP20)

Enums should make your code more readable, not less!

Resources

- Python Enum Documentation: <https://docs.python.org/3/library/enum.html>
- PEP 435 -- Adding an Enum type to the Python standard library:
<https://peps.python.org/pep-0435/>
- hdate library: <https://github.com/py-libhdate/py-libhdate>

Thank you

