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

```python
@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

```python
def __add__(self, other: timedelta):
    _year, _month, _day = self.year, self.month, self.day
    days_remaining = other.days

    while days_remaining > 0:
        days_left_in_month = get_month_length(_month, _year) - ...

        if days_remaining > days_left_in_month:
            _month = get_next_month(_month, _year)
            ...

    return HebrewDate(_year, _month, _day)
```

Enums are classes (and can have methods)

```python
class Months(Enum):

    def next_month(self, year) -> Months:
        """Return the next month."""
        if self == Months.ELUL:
            return Months.TISHREI
        if self in {Months.ADAR, Months.ADAR_II}:
            return Months.NISAN
        if is_leap_year(year) and self == Months.SHVAT:
            return Months.ADAR_I
        return Months(self._value_ + 1)
```

... and even attributes

```python
class Months(Enum):
    TISHREI = 1, 30
    TEVET = 4, 29

    def __new__(cls, value, length):
        obj = object.__new__(cls, value)
        obj._value_ = value
        obj._length = length
        return obj

# Usage
print(Months.TISHREI._length)     # 30
print(Months.TEVET.value)         # 4
```

… which can be dynamic 🏃

```python
class Months(Enum):
    CHESHVAN = 2, lambda year: 30 if long_cheshvan(year) else 29
    KISLEV = 3, lambda year: 30 if not short_kislev(year) else 29

    def length(self, year = None):
        """Return the number of days in this month."""
        if callable(self._length):
            return self._length(year)
        return self._length

print(Months.CHESHVAN.length(5786))  # 29
```

Part II - Creating Enums dynamically

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

```yaml
- name: "feature_a"
  products: ["process_y"]

- name: "feature_b"
  products: ["process_x"]

- 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 ...

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

... mapped at runtime to an Enum:

```python
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

```
>>> process = ProcessConfig("process_z")
ValueError: 'process_z' is not a valid ProcessConfig
```

⭐ 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 == timdelta(days=1)
True
```

```
>>> str(today)
'ח׳ Elul תשפ"ה'  # 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