Dataclasses: A boilerplate for Python classes

Ammara Laeeq
PyCon 2018

Outline

- What are Dataclasses?
- Alternatives and Motivation
- How to use them?
- Comparison with NamedTuple
- Comparison Summary
- Flexibility
- Ordering and Immutability
- Inheritance
- Optimization
- References & Further Reading

What are Dataclasses?

- Two components
  - Data holder
  - Boilerplate/ code generator for Python classes

  - What is a code generator?
    - It writes code for you
    - Reduces errors, wordiness and saves time
    - Helps implementing best practices

Alternatives and Motivation

- Alternatives
  - Tuples
  - Dicts
  - Standard Python Classes
  - Named Tuples
  - ORMs (Like Djnago, SQLAlchemy, etc.)
  - A third party library ***attrs***

- Motivation
  - *attrs* package was one of the inspiration and motivation for dataclasses
  - Type annotation using a new syntax *count: int* introduced in Python 3.6 paved the way as well

How to Use Them?

- Base Case
  - Dataclass feature is implemented as a decorator
  - Removes wordiness
  - Clean syntax
  - Less prone to human errors

```
from dataclasses import dataclass

@dataclass
class Location:
    name: str
    latitude: float
    longitude: float
```

```
>>> loc = Location('Lahore', 10.8, 59.9)
>>> print(loc)
Location(name='Lahore', latitude=10.8, longitude=59.9)
>>> loc.latitude
10.8
>>> print(f'{loc.name} is at {loc.latitude}°N, {loc.longitude}°E')
Lahore is at 10.8°N, 59.9°E
```

Comparison with Named Tuples

```python
from dataclasses import dataclass

@dataclass
class Location:
    name: str
    latitude: float
    longitude: float
```

```python
from typing import NamedTuple

class Location(NamedTuple):
    name: str
    lattitude: float
    longitude: float
```

Working with Dataclasses

```
>>> loc = Location('Islamabad', 13, 40.3)
>>> loc
Location(name=Islamabad, latitude=13, longitude=40.3)
>>> loc.name
Islamabad

>>> replace(loc, name='Rawalpindi')
Location(name=Rawalpindi, latitude=13, longitude=40.3)
>>> asdict(loc)
{'name': 'Rawalpindi', 'latitude': 13, 'longitude': 40.3}
>>> astuple(loc)
(Rawalpindi, 13, 40.3)

>>> Location.__annotations__
{'name': <class 'str'>,
 'latitude': <class 'float'>,
 'longitude': <class 'float'>}

>>> loc.longitude = 35.5
>>> loc
Location(name=Rawalpindi, latitude=13, longitude=35.5)

>>> import sys
>>> sys.getsizeof(loc) + sys.getsizeof(vars(loc))
170

>>> import timeit
>>> min(timeit.repeat('loc.name', 'from __main__ import loc'))
0.0363401048834793
```

Working with Named Tuples

```
>>> loc = Location('Islamabad', 13, 40.3)
>>> loc
Location(name=Islamabad, latitude=13, longitude=40.3)
>>> loc.name
Islamabad

>>> loc._replace(loc, name='Rawalpindi')
Location(name=Rawalpindi, latitude=13, longitude=40.3)
>>> loc._asdict(loc)
OrderedDict{'name': 'Rawalpindi', 'latitude': 13, 'longitude': 40.3}
>>> tuple(loc)
(Rawalpindi, 13, 40.3)

>>> Location.__annotations__
OrderedDict{'name': <class 'str'>,
 'latitude': <class 'float'>,
 'longitude': <class 'float'>}

>>> name, lat, long = loc

>>> import sys
>>> sys.getsizeof(loc)
75

>>> import timeit
>>> min(timeit.repeat('loc.name', 'from __main__ import loc'))
0.0663501944572793
```

Comparison Summary

- **Dataclass**
  - Implemented as a decorator
  - Secure
  - Flexible
  - replace( ) function
  - asdict( ) function
  - astuple( ) function
  - Converts to dict
  - Mutable
  - Unhashable
  - Non-iterable
  - No comparison methods
  - Storage: instance dict
  - Space usage: 170 bytes
  - Attribute access time: 36 ns
- **NamedTuple**
  - Uses Inheritance
  - Less Secure
  - Not Flexible
  - _replace( ) method
  - _asdict( ) method
  - tuple( ) function
  - Converts to OrderedDict
  - Immutable
  - Hashable
  - Iterable/Unpackable
  - Sortable
  - Storage: tuple
  - Space usage: 75 bytes
  - Attribute access time: 66 ns

What code is generated?

```
class Location:
    'Location(name: str, latitude: float, longitude: float)'

    def __init__(self, name, latitude, longitude):
        self.name = name
        self.latitude = latitude
        self.longitude = longitude

    def __repr__(self):
        return (f'{self.__class__.__name__}'
                f'(name={self.name!r}, latitude={self.latitude!r}, \
                    longitude={self.longitude})')

    def __eq__(self, other):
        if other.__class__ is not self.__class__:
            return NotImplemented
        return (self.name, self.latitude, self.longitude) ==
                (other.name, other.latitude, other.longitude)

    __hash__ = None

    name: str
    latitude: float
    longitude: float

    __dataclass_params__ = _DataclassParams(
        init=True,
        repr=True,
        eq=True,
        order=False,
        unsafe_hash=False,
        frozen=False)
```

What code is generated?

```
__dataclass_fields__ = {
    'name': Field(default=_MISSING_TYPE,
                  default_factory=_MISSING_TYPE,
                  init=True,
                  repr=True,
                  hash=None,
                  compare=True,
                  metadata={}),
    'latitude': Field(default=_MISSING_TYPE,
                      default_factory=_MISSING_TYPE,
                      init=True,
                      repr=True,
                      hash=None,
                      compare=True,
                      metadata={}),
    'longitude': Field(default=_MISSING_TYPE,
                       default_factory=_MISSING_TYPE,
                       init=True,
                       repr=True,
                       hash=None,
                       compare=True,
                       metadata={})
}
__dataclass_fields__['name'].name = 'name'
__dataclass_fields__['name'].type = str
__dataclass_fields__['latitude'].name = 'latitude'
__dataclass_fields__['latitude'].type = float
__dataclass_fields__['longitude'].name = 'longitude'
__dataclass_fields__['longitude'].type = float
```

Ordering and Immutability

- By default, dataclasses are mutable
- Not Hashable
- Cannot be used as set elements and dictionary keys
- Not orderable by default
- Prevents TypeErrors when one or more fields are unorderable
- Can be changed in just a minute

Ordering and Immutability

```
from dataclasses import dataclass

@dataclass(order=True, frozen=True)
class Location:
    name: str
    latitude: float
    longitude: float
```

Ordering and Immutability

```
    def __lt__(self, other):
        if other.__class__ is self.__class__:
            return (self.name, self.latitude, self.longitude) <
                   (other.name, other.latitude, other.longitude)
        return NotImplemented

    def __le__(self, other):
        if other.__class__ is self.__class__:
            return (self.name, self.latitude, self.longitude) <=
                   (other.name, other.latitude, other.longitude)
        return NotImplemented

    def __gt__(self, other):
        if other.__class__ is self.__class__:
            return (self.name, self.latitude, self.longitude) >
                   (other.name, other.latitude, other.longitude)
        return NotImplemented

    def __ge__(self, other):
        if other.__class__ is self.__class__:
            return (self.name, self.latitude, self.longitude) >=
                   (other.name, other.latitude, other.longitude)
        return NotImplemented

    def __setattr__(self, name, value):
        if type(self) is cls or name in ('name', 'latitude', 'longitude'):
            raise FrozenInstanceError(f"cannot assign to field {name!r}")
        super(cls, self).__setattr__(name, value)
    def __delattr__(self, name):
        cls = self.__class__
        if type(self) is cls or name in ('name', 'latitude', 'longitude'):
            raise FrozenInstanceError(f"cannot delete field {name!r}")
        super(cls, self).__delattr__(name)

    def __hash__(self):
        return hash((self.name, self.latitude, self.longitude))
```

Flexibility

- What to Generate?
- Default Values
- Type Hints
- Adding Methods
- Custom Field Specification
  - Field Factories
  - Hashing for immutable fields only
  - Customising field display
  - Customising comparison fields
  - Attaching metadata

What to Generate?

- init: Default is True
- repr: Default is True
- eq: Default is True
- order: Default is False
- unsafe_hash: Default is False
- frozen: Default is False

Default Values

```python
from dataclasses import dataclass

@dataclass
class Location:
    name: str
    latitude: float = 0.0
    longitude: float = 0.0
```

```
>>> Location('Empty Location')
Location(name='Empty Location', latitude=0.0, longitude=0.0)
>>> Location('Karachi', longitude=31.5)
Location(name='Karachi', latitude=0.0, longitude=31.5)
>>> Location('Quetta', 15.9, 45.8)
Location(name='Quetta', latitude=15.9, longitude=45.8)
```

Type Hints

```
from dataclasses import dataclass

@dataclass
class Location:
    name: str
    latitude: float
    longitude: float
```

```
from dataclasses import dataclass
from typing import Any

@dataclass
class WithoutTypeHints:
    name: Any
    value: Any = 42
```

```
>>> Location(3.14, 'pi loc', 2018)
Location(name=3.14, latitude='pi loc', longitude=2018)
```

Adding Methods

```
from dataclasses import dataclass
from math import sqrt

@dataclass
class Location:
    name: str
    latitude: float = 0.0
    longitude: float = 0.0

    def calculate_distance(self, other):
        lat_diff = (other.latitude - self.latitude)**2
        lon_diff = (other.longitude - self.longitude)**2
        return sqrt(lat_diff+lon_diff)
```

Custom Field Specification

- Employee Class
  - employee_id
  - Name
  - Gender
  - Salary
  - Age
  - List of viewers

Field Factories

- Helps specify container types for fields like lists, dicts, etc.
- *field (default_factory=list)*
- Always excluded from the hash

Hashing for Immutable Fields Only

- Immutable parts of the Employee record are *employee_id*, *name* and *gender*
- Default is same as *compare*
- Set *field(hash=False)* for mutable fields

Customising Field Display

- By default, all fields are included in *__repr__()* method
- *field (repr=False)*

Customising Comparison Fields

- By default, all fields are included in the comparison methods
- *field (compare=False)*
- Examples are functions and complex numbers are not orderable

Attaching Metadata

- Data driven applications require information about type of data
- Dataclasses provide this opportunity through *metadata* parameter
- Set *metadata{‘currency’ : ‘dollar’}* for *salary*

Creating Employee Class

```
from dataclasses import dataclass, field
from datetime import datetime

@dataclass(order=True, unsafe_hash=True)
class Employee:
    emp_id: int
    name: str
    gender: str
    salary: int = field(hash=False, repr=False, metadata={'currency': 'dollar'})
    age: int = field(hash=False)
    viewed_by: list = field(default_factory=list, compare=False, repr=False)

    def monitor(self, viewer_id):
        self.viewed_by.append((viewer_id, datetime.now()))
```

Inheritance

```
from dataclass import dataclass

@dataclass
class Location:
    name: str
    latitude: float
    longitude: float

@dataclass
class Country(Location):
    country_name: str
```

```
>>> Country('Lahore', 10.8, 59.9, 'Pakistan')
Country(name='Lahore', latitude=10.8, longitude=59.9, country_name='Pakistan')
```

Inheritance

```python
from dataclass import dataclass

@dataclass
class Location:
    name: str
    latitude: float = 0.0
    longitude: float = 0.0

@dataclass
class Country(Location):
    country_name: str
```

```python
def __init__(name: str, latitude: float = 0.0, longitude: float = 0.0, country_name: str):
    ...
```

Inheritance

```
from dataclass import dataclass

@dataclass
class Location:
    name: str
    latitude: float = 0.0
    longitude: float = 0.0

@dataclass
class Country(Location):
    country_name: str = 'Empty'
    latitude: float = 10.0
```

```
>>> Country('Lahore', 'Pakistan')
Country(name='Lahore', latitude=10.0, longitude=0.0, country_name='Pakistan')
```

Optimization

```
from dataclass import dataclass

@dataclass
class Location:
    name: str
    latitude: float
    longitude: float

@dataclass
class SlotLocation:
    __slots__ = ['name', 'latitude', 'longitude']
    name: str
    latitude: float
    longitude: float
```

```
>>> simple = Location('Lahore', 10.0, 51.5)
>>> slot = SlotLocation('Karachi', 20.5, 35.4)
>>> import timeit
>>> min(timeit.repeat('simple.name', 'from __main__ import simple'))
0.0333401049999793
>>> min(timeit.repeat('slot.name', 'from __main__ import slot'))
0.022678440234935
```

References & Further Readings

- PEP 557 -- https://www.python.org/dev/peps/pep-0557/
- GitHub repo -- https://github.com/ericvsmith/dataclasses/issues?utf8=%E2%9C%93&q=
- Raymond Hettinger's PyCon 2018 talk -- https://www.youtube.com/watch?v=T-TwcmT6Rcw
- Dataclasses backport for Python 3.6 -- https://github.com/ericvsmith/dataclasses
- RealPython Tutorial -- https://realpython.com/python-data-classes/

Thank You!