⚡Dataclasses⚡

PEP 557 & Beyond

Anatomy of this ~~⚡~~ Talk

- Data Class - Was Ist Das?!
- Features - Shiny New Toys!
- HC SVNT DRACONES
- Pydantic - Turning it to 11

Data Class - Was Ist Das?!

Knowledge from the source

"Data Classes can be thought of as mutable namedtuples with defaults" - PEP 557

Data Class - Was Ist Das?!

PEP 557

- Accepted on Mon Dec 4 12:17:25 EST 2017
- Author: Eric V. Smith <eric@trueblade.com>
- Python 3.7 and beyond
- New way of writing classes specifically to store values

```
from dataclass import dataclass

@dataclass
class InventoryItem:
    '''Class for keeping track of an item in inventory.'''
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

Features - Default Methods

__init__

```
def __init__(self, name: str, unit_price: float,
quantity_on_hand: int = 0) -> None:
    self.name = name
    self.unit_price = unit_price
    self.quantity_on_hand = quantity_on_hand
```

__repr__

```
def __repr__(self):
    return f'InventoryItem(name={self.name!r},
unit_price={self.unit_price!r},
quantity_on_hand={self.quantity_on_hand!r})'
```

__eq__

```
def __eq__(self, other):
    if other.__class__ is self.__class__:
        return (self.name, self.unit_price,
self.quantity_on_hand) == (other.name,
other.unit_price, other.quantity_on_hand)
    return NotImplemented
```

__hash__ (If frozen or unsafe_hash)

```
@dataclass(frozen=True)
class ImmutableItem:
    '''Class representing an immutable item'''
...

@dataclass(unsafe_hash=True)
class ImmutableItem:
    '''Class representing an immutable item'''
...
```

Features - Field Objects

Type Hinting

- PEP 526 FTW!
- Type hinting is **required**!
- Another dataclass can be a type

Default Fields and Factories

- Defaults can be values (default) or callables (factories)

```
from dataclasses import dataclass, field
import sys


def get_argv():
   return sys.argv[1]

@dataclass
class SubObject(object):
   sub_field_a: int
   Sub_field_b: str

@dataclass
class SimpleDataObject(object):
   field_a: SubObject
   field_b: int = 5
   field_c: str = field(default_factory=get_argv)
```

Features - Post Init Processing

```python
@dataclass
class InventoryItem:
   '''Class for keeping track of an item in inventory.'''
   name: str
   rrp_price: float
   quantity_on_hand: int = 0
   percent_rebate: float = 1
   current_price: float = field(init=False)

   def __post_init__(self):
       assert self.percent_rebate <= 1, 'percent rebate must be less or equal to 1'
       assert self.percent_rebate > 0, 'percent rebate must be greater than 0'
       self.current_price = self.rrp_price * self.percent_rebate

   def total_cost(self) -> float:
       return self.current_price * self.quantity_on_hand
```

```
@dataclass
class SimpleBaseObject(object):
    field_0: str = 'default'
    field_b: str = 'original'

@dataclass
class SimpleDataObject(SimpleBaseObject):
    field_a: str
    field_b: str = 'new default'
```

✗

```
TypeError: non-default argument ‘field_a’ follows
default argument
```

Pydantic - Turning it to 11

```python
from datetime import datetime
from typing import List
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name = 'John Doe'
    signup_ts: datetime = None
    friends: List[int] = []
```

Pydantic

- Python 3.6 backwards compatibility for dataclasses
- Helpful JSON formatted Validation error messages
- JSON Schema generation from pydantic class
- Helper functions to import objects
- Helper functions to serialize objects
- Makes DataClasses go to 11

⚡Thank You⚡

No questions, I’m definitely out of time...