
India 09
PyCon


Attendee
PYCON
INDIA-2010


>>> import pytho
PyCon 2011
16th - 18th
September 2011


PyCon India
2012


10
Years of
PyCon India


PyCon India
Hyderabad 2018


PyCon India 2013
Bangalore • NIMHANS Convention Centre


PyCon India
2014 Bangalore


PyCon India
2015
Bangalore


pycon india
2016, New Delhi


PyCon India
2017 | Delhi-NCR
Annual conference happening this
November

PyCon India Hyderabad 2018

BangPypers

What you need to know about

Sasidhar Donaparthi (@sdonapar)

https://github.com/sdonapar/data_classes

A bit of history

Python 3.5 introduced Type hints (PEP 484)

```python
# PEP 484 - Type hints
# Python3.5
def greeting(name: str) -> str:
    return f"Hello {name}"
```

Experience with Python type hints

By Kracekumar Ramaraju

A bit of history

Python 3.6 introduced Syntax for variable annotation (PEP 526)

```python
# PEP 526 - Syntax for variable annotations
# Python3.6

class Vehicle:
    make: str = 'Maruti'
    year_of_manufacture: int = 2018

    def __init__(self, registration_number: str ):
        self.registration_number = registration_number

    def __repr__(self):
        return f"Vehicle({self.registration_number})"
```

Motivation

- There have been numerous attempts to define classes which exist primarily to store values which are accessible by attribute lookup.
- Standard Library
  - collections.namedtuple
  - typing.NamedTuple
- The popular attrs project.
- George Sakkis' recordType recipe, a mutable data type inspired by collections.namedtuple, etc.,
- Data Classes are not, and are not intended to be, a replacement mechanism for all of the above libraries. But being in the standard library will allow many of the simpler use cases to instead leverage Data Classes. Many of the libraries listed have different feature sets, and will of course continue to exist and prosper.

Motivation

`namedtuple()` Factory Function for Tuples with Named Fields

Named tuples assign meaning to each position in a tuple and allow for more readable, self-documenting code. They can be used wherever regular tuples are used, and they add the ability to access fields by name instead of position index.

`collections.`**`namedtuple`**(*typename*, *field_names*, *, *rename=False*, *defaults=None*, *module=None*)

Returns a new tuple subclass named *typename*. The new subclass is used to create tuple-like objects that have fields accessible by attribute lookup as well as being indexable and iterable. Instances of the subclass also have a helpful docstring (with typename and field_names) and a helpful `__repr__()` method which lists the tuple contents in a `name=value` format.

```
from collections import namedtuple

Person = namedtuple('Person',['person_id','name','gender'])

if __name__ == '__main__':
    person_1 = Person(100,'Guido','M')

    print(person_1) # nice __repr__ method
    print(person_1.name) # access by name
    person_id,name,gender = person_1 #unpack
    print(person_id,name,gender)
```

```
Person(person_id=100, name='Guido', gender='M')
Guido
100 Guido M
```

Python 2.6

Immutable

Motivation

class typing.**NamedTuple**

Typed version of namedtuple.

The resulting class has two extra attributes: `_field_types`, giving a dict mapping field names to types, and `_field_defaults`, a dict mapping field names to default values. (The field names are in the `_fields` attribute, which is part of the namedtuple API.)

```
from typing import NamedTuple

class Person(NamedTuple):
    person_id:int
    name:str
    gender:str

if __name__ == '__main__':
    person_1 = Person(100,'Guido','M')
    person_2 = Person(200,'Eric V Smith','M')

    print(person_1) # nice __repr__ method
    print(person_1.name) # access by name
    person_id,name,gender = person_2 #unpack
    print(person_id,name,gender)
```

```
Person = namedtuple('Person',['person_id','name','gender'])
```

Added in 3.5.2

3.6: Added support for PEP 526 variable annotation syntax.

3.6.1: Added support for default values, methods, and docstrings.

Immutable

Motivation

After declaring your attributes `attrs` gives you:

- a concise and explicit overview of the class's attributes,
- a nice human-readable `__repr__`,
- a complete set of comparison methods,
- an initializer,
- and much more,

`attrs` : Classes Without Boilerplate

`attrs` is the Python package that will bring back the **joy** of **writing classes** by relieving you from the drudgery of implementing object protocols (aka dunder methods).

Its main goal is to help you to write **concise** and **correct** software without slowing down your code.

For that, it gives you a class decorator and a way to declaratively define the attributes on that class:

```
>>> import attr

>>> @attr.s
... class SomeClass(object):
...     a_number = attr.ib(default=42)
...     list_of_numbers = attr.ib(factory=list)
...
...     def hard_math(self, another_number):
...         return self.a_number + sum(self.list_of_numbers) * another_number
```

Motivation

26k views | 3 score

Records

Python / datastructures , namedtuple , record / by George Sakkis (9 years ago)

RECORDS (PYTHON RECIPE)

This is a recipe similar in functionality and exec-style optimized implementation to the very well received namedtuple (http://code.activestate.com/recipes/500261/) that was included in Python 2.6. The main difference is that **records**, unlike named tuples, are mutable. In addition, fields can have a default value. Instead of subclassing tuple or list, the implementation create a regular class with __slots__.

dataclasses — Data Classes

Source code: Lib/dataclasses.py

This module provides a decorator and functions for automatically adding generated special methods such as `__init__()` and `__repr__()` to user-defined classes. It was originally described in **PEP 557**.

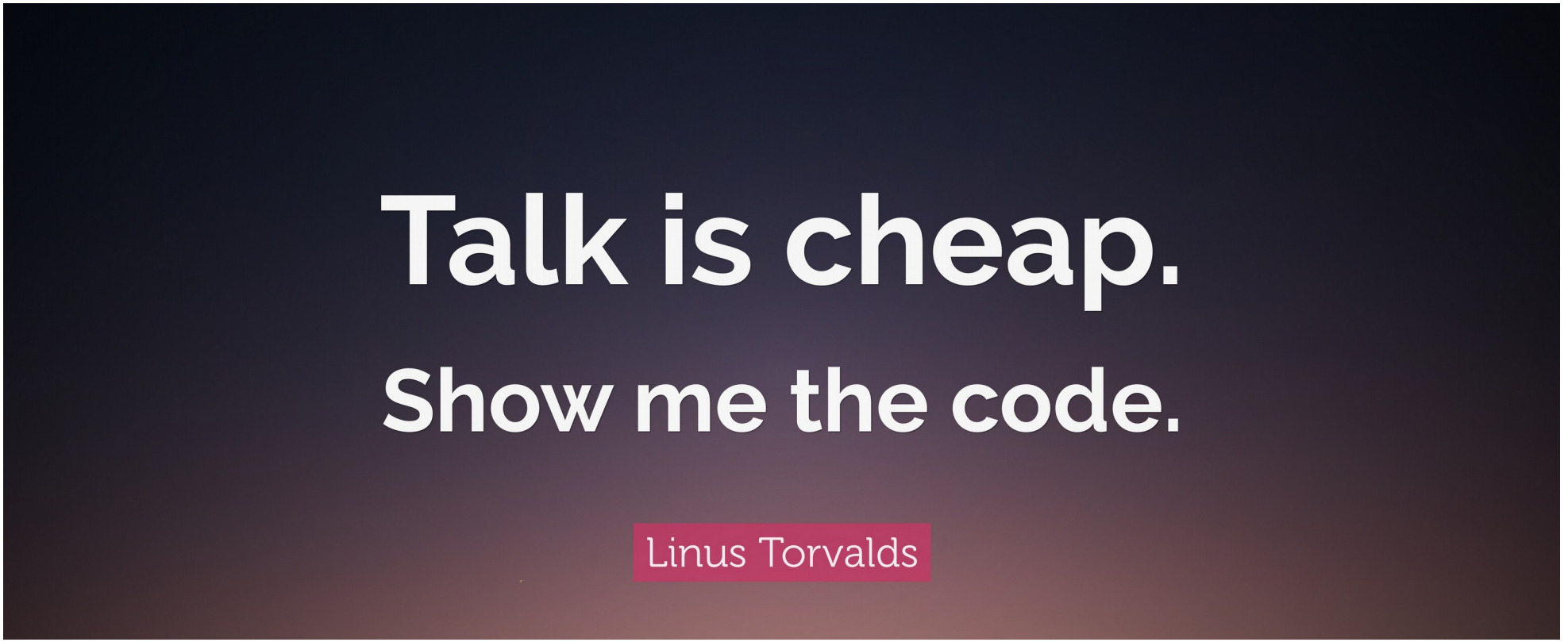
```
# PEP 557 - Dataclasses
# Python3.7

from dataclasses import dataclass

@dataclass
class Vehicle:
    registration_number: str
    make: str = 'Maruti'
    year_of_manufacture: int = 2018
```

Dataclasses

- One main design goal of Data Classes is to **support static type checkers**
- **Standard library** called Data Classes, can be thought of as "mutable namedtuples with defaults".
- A **class decorator** is provided which inspects a class definition for variables with type annotations as defined in PEP 526 (fields)
- Data Classes use normal class definition syntax, you are free to use **inheritance,metaclasses, docstrings, user-defined methods, class factories, and other Python class features**.
- Attribute type annotation is completely ignored by Data Classes, No base classes or metaclasses are used by Data Classes.

dataclass parameters

- “dataclass” function is typically used as a class decorator is provided to post-process classes and add generated methods (“dunder”)
- The dataclass decorator examines the class to find “fields.” A field is defined as any variable identified in __annotations__, is guaranteed to be an ordered mapping
- None of the Data Class machinery examines the type specified in the annotation.
- The dataclass decorator is typically used with no parameters
- def dataclass(*, init=True, repr=True, eq=True, order=False, unsafe_hash=False, frozen=False)



Let’s examine some code

Field objects

- Field objects describe each defined field
- These objects are created internally, and are returned by the fields()
- Users should never instantiate a Field object directly
- Its documented attributes are
  - name: The name of the field.
  - type: The type of the field.
  - default, default_factory, init, repr, hash, compare, and metadata

Customization

- Post-init processing
- Class variables
- Init-only variables
- Frozen instances
- Default factory functions
- Mutable default values

Conclusion

- Avoids lot of boilerplate coding
- Standard Library – Many simpler use caes can leverage
- Specified attribute type annotation is completely ignored by Data Classes.
- No base classes or metaclasses are used by Data Classes
- Not appropriate to use Data Classes
  - API compatibility with tuples or dicts is required.
  - Type validation beyond that provided by PEPs 484 and 526 is required, or value validation or conversion is required.
- __slots__ is not supported in the initial release

References

- PEP 557 Data classes
- The Ultimate Guide to Data Classes in Python 3.7
- Raymond Hettinger - Dataclasses
- Python Documentation


Thank
you


Any
Questions