
Pydantic
Giving your data classes awesome powers

Data parsing and validation using Python type hints

WHAT IS A DATACLASS?

main.py

```
from dataclasses import dataclass

@dataclass
class Tesla:
    style: str
    battery_pack: float
```

repl

```
>>> Tesla("model 3", 10)
Tesla(style='model 3', battery_size=10)
```

DATACLASSES
WHAT IF WE WANTED TO CONSTRAIN IT?

```
from enum import Enum

class Battery(str, Enum):
    standard = '380 kilograms'
    long_range = '480 kilograms'
```

```
@dataclass
class Tesla:
    style: str
    battery_pack: Tuple[Battery, ...]
```

```
>>> Tesla(2, ("petrol-gas-is-not-a-liquid"))
Tesla(style=2, battery_pack=('"petrol-gas-is-not-a-
liquid))
```

With dataclasses, types are not enforced... ouch
But in this case, we'll lean on the shoulders of a titan that is, Pydantic.

main.py

```
from pydantic.dataclasses import dataclass
from typing import Tuple
from enum import Enum

class Battery(str, Enum):
 standard = '380 kilograms'
 long_range = '480 kilograms'

@dataclass
class Tesla:
 style: str
 battery_pack: Tuple[Battery, ...]
```

Error

```
pydantic.error_wrappers.ValidationError: 1 validation error for Tesla
battery_pack
  value is not a valid tuple (type=type_error.tuple)
```

But wait!

As well as BaseModel, pydantic provides a dataclass decorator which creates (almost) vanilla python dataclasses with input data parsing and validation.

REFERENCES

Great docs
https://pydantic-docs.helpmanual.io/
Awesome presentation
http://slides.com/hultner/python-pizza-2020/