Unexpected dataclasses

```
@pa_schembri
https://netsach.com
Technology Bridging & Automation
```


Netsach

Who am I

```
Netsach founder & CEO
Full stack (hardware => cloud software)
@pa_schembri
https://netsach.com
Technology Bridging & Automation
```


Netsach

Unexpected dataclasses

- Mutable namedtuples with defaults : A tour
- Data classes and best practices and unexpected uses...

Mutable namedtuples with defaults

```
#: A basic tuple

>>> alice = ('Alice', 'Smith')
```

```
#: A named tuple
>>> from collections import namedtuple

>>> User = namedtuple('User', ('first_name', 'last_name'))

>>> alice = User('Alice', 'Smith')

>>> print(alice)

User(first_name='Alice', last_name='Smith')
```

Mutable namedtuples with defaults

- Dataclass => `class`
- Boilerplate code generated
- Behave like a named tuple

Before dataclasses : Basics

```
class User:
def __init__(self, first_name, last_name, favorites):
    self.first_name = first_name
    self.last_name = last_name
    self.favorites = favorites or []
```

A dataclass

```python
from dataclasses import dataclass

@dataclass
class User:
    first_name: str
    last_name: str
    favorites: list
```

dataclasses love typing

```python
from dataclasses import dataclass
from typing import Text, List

@dataclass
class User:
    first_name: Text
    last_name: Text
    favorites: List[Text]
```

Dangerous defaults

```
#: Don't do this !

class User:
    def __init__(self, ..., favorites=[]):
        ...
        self.favorites = favorites
```

Dangerous defaults

```
#: Don't use mutable values as defaults !

class User:
  def __init__(self, favorites=[]):
    self.favorites = favorites
    self.favorites.append('test')

>>> User()
<__main__.User object at 0x10e96e160>
>>> User()
<__main__.User object at 0x10e96e240>
>>> u = User()
>>> u.favorites
['test', 'test', 'test']
```

dataclasses defaults

```
from dataclasses import dataclass, field
from typing import Text, List

@dataclass
class User:
    first_name: Text = field(default='Jane')
    favorites: List[Text] = field(default=[])


ValueError: mutable default <class 'list'> for field
favorites is not allowed: use default_factory
```

dataclasses defaults

```python
from dataclasses import dataclass, field
from typing import Text, List

@dataclass
class User:
    first_name: Text = field(default='Jane')
    favorites: List[Text] = field(default_factory=list)
```

What you just got for free

- Sane defaults value definition
- `__init__`, `__repr__`, ...
- `asdict`, `astuple`, ...

```
from dataclasses import dataclass, field, asdict

>>> user = User()
>>> print(user)
User(first_name='Jane', favorites=[])

>>> print(asdict(user))
{'first_name': 'Jane', 'favorites': []}
```

Immutable instances

- Immutability through `frozen` keyword

```
from dataclasses import dataclass

@dataclass(frozen=True)
class User:
    first_name: str

>>> user = User(first_name='John')
>>> user.first_name = 'Roger'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 3, in __setattr__
dataclasses.FrozenInstanceError: cannot assign to field 'first_name'
```

Hashable instances

- Hashable instances...

```
@dataclass(frozen=True)
class User:
    name: Text

@dataclass(frozen=True)
class Project:
    name: Text
    manager: User

>>> p1 = Project(name="Python 3.8", manager=User('Alice'))

#: Valid code through immutability
>>> additional_info = {p1: ['info #1', 'info #2']}
>>> current_projects = set([p1])
```

Fields

- Default value
- Factories (like with `defaultdict` of the `collection` module)
- `repr`, `hash`, `compare` optional exclusion
- `metadata` parameter as a customization mechanism

Fields metadata

```
from dataclasses import dataclass, field, fields
from typing import Text

@dataclass
class User:
    first_name: Text = field(
        default='John', metadata={'key': 'value'}
    )

>>> f = fields(User)[0]
>>> print(f.name)
first_name
>>> print(f.metadata)
{'key': 'value'}
#: Magic use authorized 
```

Fields metadata

```python
from dataclasses import dataclass, field
from typing import Text

@dataclass
class User:
    first_name: Text = field(
        default='John', metadata={
            'verbose_name': 'First name'
        }
    )
```

Fields metadata

```python
from dataclasses import dataclass, field
from typing import ByteString

@dataclass
class Image:
    data: ByteString = field(
        default=b'', metadata={
            'loaders': ['module_1', 'module_2', ...]
        }
    )
    ...
```

Looking at the big picture

- Data structure definition
- Self-documented code
- Sane defaults
- Frozen / Hashable instances
- Customization hooks

More info about dataclasses

- Raymond Hettinger : Dataclasses, The code generator to end all code generators
  - https://youtu.be/T-TwcmT6Rcw

Dataclasses and best practices

- Defining and documenting interfaces

Defining and documenting interfaces

- Code written in Python is for humans
- Abstraction is required to build better software
- Trey Hunner : Readability Counts
  - https://youtu.be/knMg6G9_XCg
- Brandon Rhodes : Python and the Glories of the UNIX Tradition
  - https://youtu.be/zFMdhXYlFfY

An example : Finite State Machine

```
#: Todo #1 - Write interfaces
#: Todo #2 - Write tests
#: Todo #3 - Implement functional code
```

An example : Finite State Machine

```python
@dataclass
class State:
    pass

@dataclass
class Event:
    pass

@dataclass
class Machine:
    pass
```

```python
from dataclasses import dataclass
from typing import Text, FrozenSet, Callable

@dataclass(frozen=True)
class State:
    name: Text
    tasks: FrozenSet[Callable]
```

```python
from dataclasses import dataclass
from typing import Text

@dataclass(frozen=True)
class Event:
    name: Text
    source: State
    destination: State
```

```python
from typing import Text, Deque, Dict
from collections import deque

MachineContext = Dict

@dataclass
class Machine:
    name: Text

    current_state: State

    run_tasks: Callable[[MachineContext], None]

    context: MachineContext = field(default_factory=dict)

    events: Deque[Event] = field(
        default_factory=deque,
        metadata={
            'information': {
                'new_events': 'appendleft',
                'unprocessed_events': 'pop',
            }
        }
    )
```

Dataclasses + Typing = Specification

Netsach

Q & A

```
GET /end-of-talk/
```

```
HTTP/1.1 406 Not Acceptable
Reason: Questions ?

Important: Python 3.7.1 is out

Main-Location: https://docs.python.org/3/library/dataclasses.html
Attrs-Location: https://www.attrs.org/en/stable/
PEP-Location: https://www.python.org/dev/peps/pep-0557/
PythonPatterns-Location: https://python-patterns.guide/

@pa_schembri
https://netsach.com
Technology Bridging & Automation
```