

ХОЧУ ПОМОЧЬ

ПРОДУ

Модуль dataclasses в Python, создание типов данных

Яндекс Взгляд · Опрос

Выберите 1 или несколько ответов

Какие сервисы проверки истории автомобилей вы знаете?

☐

Avtocod

☐

Avinfobot

☐

ПроАвто/Auto.ru

☐

Автотека/Авито

☐

Ни один из вариантов

1 из 3 вопросов

Продолжить

[Стандартная библиотека Python3.](#) / Модуль dataclasses в Python, создание типов данных

Упрощенное создание пользовательских типов данных

[Модуль dataclasses](#) предоставляет декоратор и функции для автоматического добавления сгенерированных специальных методов, таких как `__init__()` и `__repr__()`, в определяемые пользователем классы.

Атрибуты класса - переменные для использования в этих сгенерированных методах определяются с использованием [аннотаций типов](#).

Пример:

```
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Класс для отслеживания товара."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0

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

Декоратор `@dataclass` помимо прочего, добавит метод `__init__()`, который будет выглядеть так:

```
def __init__(self, name: str, unit_price: float, quantity_on_hand: int=0):
    self.name = name
    self.unit_price = unit_price
    self.quantity_on_hand = quantity_on_hand
```

Обратите внимание, что этот метод автоматически добавляется в класс: он не прописывается вручную в показанном выше определении класса `InventoryItem()`. Такое поведение облегчает написание небольших классов, представляющих из себя [пользовательские типы](#), предназначенные для упорядоченного хранения нестандартных данных.

Вверх

Содержание:

- [Обработка после инициализации, метод __post_init__\(\);](#)
- [Поле dataclass как переменная ClassVar;](#)

РЕКЛАМА · 18+

ОФ

на платформе Яндекса

practicum.yandex.ru

Бесплатное занятие английским в Яндекс Практикуме

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка >

Практика английского >

Узнать больше

инициализации InitVar;

`<dataclass;`

`<dataclass;`

[занию в dataclass;](#)

[чанию в dataclass;](#)

[ключевые аргументы.](#)

инициализации, метод __post_init__().

вызовет специальный метод с именем `__post_init__()`, если `__post_init__()` определен в `lf.__post_init__()`.

`InitVar`, они также будут переданы в `__post_init__()` в том порядке, в котором они были [__init__\(\)](#) не сгенерирован, то `__post_init__()` не будет вызываться автоматически.

ализировать значения полей, которые зависят от одного или нескольких других полей.

```
с: float = field(init=False)

def __post_init__(self):
    self.c = self.a + self.b

>>> mytype = MyType(7, 9)
>>> mytype
# MyType(a=7, b=9, c=16)
```

Смотрите раздел ниже, посвященный [переменным только для инициализации](#), чтобы узнать о способах передачи параметров в специальный метод `__post_init__()`.

Также посмотрите предупреждение о том, как функция модуля [dataclasses.replace\(\)](#) обрабатывает поля `init=False`.

Поле dataclass как переменная ClassVar.

Одно из двух мест, где функция-декоратор [dataclasses.dataclass\(\)](#) фактически проверяет тип поля - это определение того, является ли поле переменной класса. В этом случае функция-декоратор проверяет, является ли тип поля типизированным [typing.ClassVar](#).

Если поле типизировано `typing.ClassVar`, то оно исключается из рассмотрения как поле и игнорируется механизмами класса данных. Такие псевдо-поля `ClassVar` не возвращаются функцией модуля [dataclasses.fields\(\)](#).

```
from dataclasses import dataclass
from typing import ClassVar

@dataclass
class MyType:
    x: int
    y: int
    n: ClassVar

>>> mytype = MyType(7, 9)
>>> mytype
# MyType(x=7, y=9)

# присвоим переменной класса данных `n` значение
>> 

Вверх

 e.n = 5
# переменная `n` игнорируется механизмами
```

класса данных и не будет видна как поле

>>> mytype

Point(x=7, y=9)

РЕКЛАМА · 18+

ОР



practicum.yandex.ru

Бесплатное занятие английским в Яндекс Практикуме

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка >

Практика английского >

Узнать больше

mytype = MyType(10, database=my_database)

работать

лько для инициализации InitVar.

атор `dataclasses.dataclass()` проверяет [аннотацию типа](#), - это определение, является ли иализации.

ли тип поля тип `dataclasses.InitVar`. Если поле является `InitVar`, то оно считается полем только для инициализации. Поскольку это не истинное поле, оно не возвращается . Поля только для инициализации добавляются в качестве параметров к сгенерированному в необязательный метод `__post_init__()`. По другому они не используются классами данных.

будет инициализировано из базы данных, если значение не указано при создании класса:

```
Type] = None

tabase):
database is not None:
lookup('j')
```

В этом случае `dataclasses.fields()` вернет объекты `Field` для `i` и `j`, но не для базы данных.

Неизменяемые классы данных dataclass.

Невозможно создать действительно неизменяемые объекты Python, но, передав `Frozen=True` декоратору `dataclasses.dataclass()`, можно эмулировать неизменяемость. В этом случае классы данных добавят к классу методы `__setattr__()` и `__delattr__()`. При вызове эти методы вызывают ошибку `dataclasses.FrozenInstanceError`.

При использовании аргумента `frozen=True` наблюдается небольшое снижение производительности: `__init__()` не может использовать простое присваивание для инициализации полей и должен использовать `object.__setattr__()`.

```
from dataclasses import dataclass

@dataclass(frozen=True)
class MyType:
    x: int
    y: int

>>> mytype = MyType(7, 9)
>>> mytype
# Point(x=7, y=9)
>>> mytype.x = 15
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
#   File "<string>", line 4, in __setattr__
# dataclasses.FrozenInstanceError: cannot assign to field 'x'
```

Наследование классов данных dataclass.

Когда класс данных создается декоратором `dataclasses.dataclass()`, он просматривает все базовые классы класса в обратном MRO (то есть начиная с объекта) и для каждого найденного класса данных добавляет поля этого базового класса в упорядоченное отображение полей.

После добавления всех полей базового класса он добавляет свои собственные поля к упорядоченному отображению. Все сг


Вверх

 данные методы будут использовать это комбинированное вычисляемое упорядоченное сопоставление полей. Поскольку поля расположены в порядке вставки, производные классы переопределяют базовые классы. Например:

```
from dataclasses import dataclass
from typing import Any
```

@dataclass

РЕКЛАМА · 18+



practicum.yandex.ru

Бесплатное занятие английским в Яндекс Практикуме

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка >

Практика английского >

Узнать больше

```
x: int
mylist: list = field(default_factory=list)
```

```
>>> mytype = MyType(9)
>>> mytype1 = MyType(9, [10, 20, 30])
>>> mytype
# MyType(x=9, mylist=[])
>>> mytype1
# MyType(x=9, mylist=[10, 20, 30])
```

Если поле исключено из `__init__()` (с использованием `init=False`) и в поле также указывается `default_factory`, то тогда фабричная функция по умолчанию всегда будет вызываться из сгенерированной функции `__init__()`. Это происходит потому, что другого способа присвоить полю начальное значение нет.

Изменяемые значения по умолчанию в dataclass.

Python сохраняет значения переменных-членов по умолчанию в атрибутах класса. Рассмотрим класс типа данных, который не использует модуль `dataclasses`:

```
class MyType:
    x = []
    def add(self, element):
        self.x.append(element)
```

```
>>> o1 = MyType()
>>> o2 = MyType()
>>> o1.add(1)
>>> o2.add(2)
>>> assert o1.x == [1, 2]
>>> assert o1.x is o2.x
```

Обратите внимание, что такой код имеет проблему - два экземпляра класса `MyType`, которые не указывают значение для `x` при создании экземпляра класса, будут использовать одну и ту же копию списка `x`.

Что бы защититься от таких ошибок, при создании классов для хранения данных нужно использовать [модуль dataclasses](#) и декоратор `@dataclass`. Если написать код, эквивалентный предыдущему примеру с использованием `@dataclass`, то при создании экземпляра класса `MyType()` просто поднимется исключение.

```
from dataclasses import dataclass

@dataclass
class MyType:
    last):

    le default {type(f.default)} for field '
    <class 'list'> for field x is not allowed: use default_factory

ValueError, если обнаруживают параметр по умолчанию list, dict или set. Это частичное
гих проблем.

рые используют в качестве полей изменяемые последовательности используйте фабричные
```

РЕКЛАМА · 18+

practicum.yandex.ru

**Бесплатное
занятие
английским
в Яндекс
Практиуме**

Полноценное занятие
с преподавателем, а не
презентация курсов

Устный тест на уровень
языка >

Практика английского >

Узнать больше

Классы данных и ключевые аргументы.

С версии Python 3.10 модуль dataclasses поддерживает поля, содержащие только ключевые слова в сгенерированном методе `__init__()`. Есть несколько способов указать поля, содержащие только ключевые слова.

Можете определить в конструкторе, что каждое поле содержит только ключевые слова:

```
from dataclasses import dataclass

@dataclass(kw_only=True)
class Birthday:
    name: str
    birthday: datetime.date
```

И `name`, и `birthday` являются только ключевыми аргументами для сгенерированного метода `__init__()`.

Можно указать только ключевые слова для каждого поля:

```
from dataclasses import dataclass

@dataclass
class Birthday:
    name: str
    birthday: datetime.date = field(kw_only=True)
```

Здесь только `birthday` - только ключевое слово. Если устанавливать `kw_only` для отдельных полей, то нужно иметь в виду, что существуют правила переупорядочения полей из-за того, что поля, содержащие только ключевые аргументы. Эти поля должны следовать за полями, не содержащими ключевые аргументы.

Также можно указать, что все поля, следующие за маркером `KW_ONLY`, содержат только ключевые аргумента. Скорее всего, это будет наиболее распространенное использование:

```
from dataclasses import dataclass, KW_ONLY

@dataclass
class Example:
    x: float
    y: float
    z: float
    w: float
    v: float
    u: float
    t: float
    s: float
    r: float
    q: float
    p: float
    o: float
    n: float
    m: float
    l: float
    k: float
    j: float
    i: float
    h: float
    g: float
    f: float
    e: float
    d: float
    c: float
    b: float
    a: float
    kw_only=True
```

```
y: float
_: KW_ONLY
z: float = 0.0
t: float = 0.0
```

РЕКЛАМА · 18+



practicum.yandex.ru

Бесплатное
занятие
английским
в Яндекс
Практикуме

Полноценное занятие
с преподавателем, а не
презентация курсов

Устный тест на уровень
языка

DOCS-Python.ru™, 2023 г.
Практика английского

Узнать больше

и только для ключевых слов, а x и y - нет.

Содержание раздела:

[dataclasses](#)

[ses](#)

[ses](#)

[sses](#)

[dataclasses](#)

[sses](#)

[ataclasses](#)

[ses](#)

[ses](#)

(Внимание! При копировании материала ссылка на источник обязательна)

[@docs_python_ru](#)

Вверх