Project description

# Dataclass CSV

Dataclass CSV permet de travailler avec des fichiers CSV plus facilement et beaucoup mieux que de travailler avec Dict. Il utilise les classes de données Python pour stocker les données de chaque ligne du fichier CSV et utilise également des annotations de type qui permettent une vérification et une validation de type correctes.

Caractéristiques principales  
  
Utilisez des classes de données au lieu de dictionnaires pour représenter les lignes du fichier CSV.  
 Tirez parti de l'annotation de type de propriétés de classe de données. DataclassReader utilise l'annotation de type pour valider les données du fichier CSV.  
     Conversion de type automatique. DataclassReader supporte str, int, float, complex, datetime et bool.

Vous aide à résoudre les problèmes liés aux données du fichier CSV. DataclassReader montrera exactement dans quelle ligne du fichier CSV contient des erreurs.  
Extrayez uniquement les données dont vous avez besoin. Il analysera uniquement les propriétés définies dans la classe de données.  
Syntaxe familière. DataclassReader est utilisé presque de la même manière que DictReader dans la bibliothèque standard.  
Il utilise des fonctionnalités de classe de données qui vous permettent de définir des propriétés de métadonnées afin que les données puissent être analysées exactement comme vous le souhaitez.  
Rendre le code plus propre. Plus de boucles supplémentaires pour convertir les données dans le type correct, effectuer une validation, définir des valeurs par défaut, DataclassReader fera tout cela à votre place.

Commencer  
  
Tout d'abord, ajoutez les importations nécessaires :

from dataclasses import dataclass

from dataclass\_csv import DataclassReader

En supposant que nous ayons un fichier CSV avec le contenu ci-dessous :

prenom,email,age

Elsa,elsa@test.com, 11

Astor,astor@test.com, 7

Edit,edit@test.com, 3

Ella,ella@test.com, 2

Créons une classe de données qui représentera une ligne dans le fichier CSV ci-dessus:  
  
*@dataclass*

class **utilisateur**() :

prenom: str

email: str

age: int

La classe de données utilisateur a 3 propriétés, prenom et email est de type str et age est de type int.  
  
Pour charger et lire le contenu du fichier CSV, nous procédons comme si nous utilisions DictReader à partir du module csv de la bibliothèque standard de Python. Après avoir ouvert le fichier, nous créons une instance de DataclassReader en passant deux arguments. Le premier est le fichier et le second est la classe de données que nous souhaitons utiliser pour représenter les données de chaque ligne du fichier CSV. Ainsi:

with open(filename) as users\_csv:

reader = DataclassReader(users\_csv, User)

for row in reader:

print(row)

DataclassReader utilise en interne le DictReader du module csv pour lire le fichier CSV, ce qui signifie que vous pouvez transmettre les mêmes arguments que vous transmettriez au DictReader. La liste complète des arguments est indiquée ci-dessous:

dataclass\_csv.DataclassReader(

f,

cls,

fieldnames=None,

restkey=None,

restval=None,

dialect=*'excel'*,

\*args,

\*\*kwds

)

User(prenom='Elsa', email='elsa@test.com', age=11)

User(prenom ='Astor', email='astor@test.com', age=7)

User(prenom ='Edit', email='edit@test.com', age=3)

User(prenom ='Ella', email='ella@test.com', age=2)

La gestion des erreurs  
  
L'un des avantages de l'utilisation de DataclassReader est qu'il permet de détecter facilement lorsque le type de données dans le fichier CSV n'est pas conforme aux attentes du modèle de votre application. De plus, DataclassReader affiche des erreurs qui aideront à identifier les lignes présentant un problème dans votre fichier CSV.  
  
Par exemple, supposons que nous modifions le contenu du fichier CSV indiqué dans la section Mise en route et que nous modifions l'âge de l'utilisateur Astor, modifions-le en une valeur de chaîne

Astor, astor@test.com, test

N'oubliez pas que dans l'utilisateur DataClass, la propriété age est annotée avec int. Si nous relançons le code, une exception sera générée avec le message ci-dessous:

**dataclass\_csv**.exceptions.CsvValueError: The field **`age`** is defined as <class *'int'*> but

received a value of type <class *'str'*>. [CSV Line number: 3]

Notez qu'en plus d'indiquer quelle était l'erreur, le DataclassReader indiquera également quelle ligne du fichier CSV contient les données avec des erreurs.

Les valeurs par défaut  
  
DataclassReader gère également les propriétés avec des valeurs par défaut. Modifions la classe de données utilisateur et ajoutons une valeur par défaut pour le champ email:

from dataclasses import dataclass

*@dataclass*

class **utilisateur**() :

prenom: str

email: str *'Not specified'*

age: int

Mappage des champs de la classe de données aux colonnes  
  
Le mappage entre une propriété dataclass et une colonne du fichier CSV sera effectué automatiquement si les noms correspondent. Toutefois, il existe des cas où le nom de l'en-tête d'une colonne est différent. Nous pouvons facilement dire à DataclassReader comment le mappage doit être effectué à l'aide de la méthode map. En supposant que nous ayons un fichier CSV avec le contenu ci-dessous:  
  
Prenom, email, age  
Elsa, elsa @ test.com, 11

Notez que maintenant, la colonne s'appelle Prénom et non prénom  
  
Et nous pouvons utiliser la méthode map, comme ceci:

reader = DataclassReader(users\_csv, User)

reader.map(*'Prenom*).to(*'prenom'*)  
  
Dorénavant, DataclassReader saura comment extraire les données de la colonne Prenom et les ajouter à la propriété to dataclass.

Dorénavant, DataclassReader saura comment extraire les données de la colonne First Name et les ajouter à la propriété to dataclass.

Annotation de type prise en charge  
  
Pour le moment, DataclassReader prend en charge int, str, float, complex, datetime et bool. Lors de la définition d'une propriété datetime, il est nécessaire d'utiliser le décorateur dateformat, par exemple:

from dataclasses import dataclass

from datetime import datetime

from dataclass\_csv import DataclassReader, dateformat

*@dataclass*

*@dateformat*(*'%Y/%m/%d'*)

class **User**:

name: str

email: str

birthday: datetime

if \_\_name\_\_ == *'\_\_main\_\_'*:

with open(*'users.csv'*) as f:

reader = DataclassReader(f, User)

for row in reader:

print(row)

En supposant que le fichier CSV ait le contenu suivant (date à l’anglaise):

name,email,birthday

Edit,edit@test.com,2018/11/23

La sortie ressemblerait à ceci:

User(name='Edit', email='edit@test.com', birthday=datetime.datetime(2018, 11, 23, 0, 0))

from dataclasses import dataclass

from datetime import datetime

from dataclass\_csv import DataclassReader, dateformat

*@dataclass*

*@dateformat*(*'%Y/%m/%d'*)

class **Utilisateur**:

prenom: str

email: str

nele: datetime

if \_\_name\_\_ == *'\_\_main\_\_'*:

with open(*'users.csv'*) as f:

reader = DataclassReader(f, Utilisateur)

for row in reader:

print(row)

En supposant que le fichier CSV ait le contenu suivant (version date française):

prenom,email,nele

jean,toto@fr.fr,1951/01/01

cool,titi@fr.fr,1875/12/25

Edit,edit@test.com,2001/05/07

La sortie ressemblerait à ceci:

Utilisateur(prenom='jean', email='toto@fr.fr', nele=datetime.datetime(1951, 1, 1, 0, 0))

Utilisateur(prenom='cool', email='titi@fr.fr', nele=datetime.datetime(1875, 12, 25, 0, 0))

Utilisateur(prenom='Edit', email='edit@test.com', nele=datetime.datetime(2001, 5, 7, 0, 0))

Changeons des informations sur le fichier csv

prenom,email,Nele

jean,toto@fr.fr, 1951/01/01

**Le debug**

KeyError: 'nele'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

File "exempledate.py", line 19, in <module>

for row in reader:

File "Python\Python37\lib\site-packages\dataclass\_csv\dataclass\_reader.py", line 191, in \_\_next\_\_

return self.\_process\_row(row)

File "Python\Python37\lib\site-packages\dataclass\_csv\dataclass\_reader.py", line 138, in \_process\_row

value = self.\_get\_value(row, field)

File "Python\Python37\lib\site-packages\dataclass\_csv\dataclass\_reader.py", line 87, in \_get\_value

f'The value `{field.name}` is missing in the CSV file.'

KeyError: 'The value `nele` is missing in the CSV file.'

Ici nom de la colonne présente dans la première ligne du fichier csv n’est pas strictement identique avec l’identifiant associé dans la classe Utilisateur. D’un côté nous avons « Nele » et dans la class ‘nele’

Autres exemples sur la transposition du format de date

prenom,email,nele

jean,toto@fr.fr, 01/01/2010

from dataclasses import dataclass

from datetime import datetime

from dataclass\_csv import DataclassReader, dateformat

*@dataclass*

*@dateformat*(*'%Y/%m/%d'*)

class **Utilisateur**:

prenom: str

email: str

nele: datetime

if \_\_name\_\_ == *'\_\_main\_\_'*:

with open(*'users.csv'*) as f:

reader = DataclassReader(f, Utilisateur)

for row in reader:

print(row)

Traceback (most recent call last):

File " TestDivers\exempledate.py", line 19, in <module>

for row in reader:

File "Python37\lib\site-packages\dataclass\_csv\dataclass\_reader.py", line 191, in \_\_next\_\_

return self.\_process\_row(row)

File "\Python\Python37\lib\site-packages\dataclass\_csv\dataclass\_reader.py", line 154, in \_process\_row

) from None

dataclass\_csv.exceptions.CsvValueError: time data '01/01/2010' does not match format '%Y/%m/%d' [CSV Line number: 2]

Le programme remonte un problème d’interprétation de la date.

Métadonnées de champs  
  
Il est important de noter que le décorateur de dateformat définira le format de date qui sera utilisé pour analyser la date de toutes les propriétés de la classe. Il existe maintenant des situations où les données d'un fichier CSV contiennent deux colonnes ou plus avec des valeurs de date dans différents formats. Il est possible de définir un format spécifique pour chaque propriété à l'aide de dataclasses.field. Disons que nous avons maintenant un fichier CSV avec le contenu suivant:

name,email,birthday, create\_date

Edit,edit@test.com,2018/11/23,2018/11/23 10:43

Comme vous pouvez le voir, la create\_date contient également des informations sur l'heure.  
  
La classe de données utilisateur peut être définie comme ceci:

from dataclasses import dataclass, field

from datetime import datetime

from dataclass\_csv import DataclassReader, dateformat

*@dataclass*

*@dateformat*(*'%Y/%m/%d'*)

class **User**:

name: str

email: str

birthday: datetime

create\_date: datetime = field(metadata={*'dateformat'*: *'%Y/%m/%d %H:%M'*})

Notez que le format du champ anniversaire n’a pas été défini à l’aide des métadonnées du champ. Dans ce cas, le format spécifié dans le décorateur dateformat sera utilisé.

Gestion des valeurs avec des espaces vides  
  
Lors de la définition d'une propriété de type str dans la classe de données, DataclassReader traitera les valeurs contenant uniquement des espaces blancs. Pour changer ce comportement, il existe un décorateur appelé @accept\_whitespaces. Lors de la décoration de la classe avec @accept\_whitespaces, toutes les propriétés de la classe accepteront les valeurs ne contenant que des espaces.  
  
Par exemple:

from dataclass\_csv import DataclassReader, accept\_whitespaces

*@accept\_whitespaces*

*@dataclass*

class **User**:

name: str

email: str

birthday: datetime

created\_at: datetime

Si vous avez besoin d'un champ spécifique pour accepter les espaces blancs, vous pouvez définir la propriété accept\_whitespaces dans les métadonnées du champ, comme suit.

*@dataclass*

class **User**:

name: str

email: str = field(metadata={*'accept\_whitespaces'*: True})

birthday: datetime

created\_at: datetime

## Copyright and License

Copyright (c) 2018 Daniel Furtado. Code released under BSD 3-clause license