array_collections Documentation

Release 2019

Yoel Rene Cortes-Pena

Jul 19, 2019

Contents

CHAPTER 1

tuple_array

class array_collections.**tuple_array**

Create an array that is immutable and hashable.

Parameters

array: [array_like] Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays.

dtype: [data-type] By default, the data-type is inferred from the input data.

order: {'C', 'F'} Whether to use row-major (C-style) or column-major (Fortran-style) memory representation. Defaults to 'C'.

Examples

Create a tuple_array object:

```
>>> arr = tuple_array([1, 18])
tuple_array([1, 18])
```

tuple_array objects are immutable:

```
>>> arr[1] = 0
TypeError: 'tuple_array' objects are immutable.
```

tuple_array objects are hashable:

```
>>> hash(arr)
3713080549427813581
```

CHAPTER 2

material_array

class array_collections.**material_array**

Create an array that issues a RuntimeWarning when a non-positive or non-finite value is encountered.

Parameters

array: [array_like] Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays.

dtype: [data-type] By default, the data-type is inferred from the input data.

order: {'C', 'F'} Whether to use row-major (C-style) or column-major (Fortran-style) memory representation. Defaults to 'C'.

Examples

Create material_array:

```
>>> arr = material_array([1, 18])
material_array([1, 18])
```

A negative value issues a RuntimeWarning:

```
>>> arr[1] = -1
__main__:1: RuntimeWarning:
Encountered negative or non-finite value in 'material_array' object.
```

classmethod enforce_valuecheck(*val*)

If *val* is True, issue warning when non-finite or negative values are encountered.

CHAPTER 3

property_array

class array_collections.**property_array**

Create an array that allows for array-like manipulation of FreeProperty objects. All entries in a property_array must be instances of FreeProperty. Setting items of a property_array sets values of objects instead.

Parameters

array: array_like[FreeProperty] Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays.

order: {'C', 'F'} Whether to use row-major (C-style) or column-major (Fortran-style) memory representation. Defaults to 'C'.

Examples

Use the PropertyFactory to create a Weight property class which calculates weight based on density and volume:

```
from array_collections import PropertyFactory

>>> @PropertyFactory
>>> def Weight(self):
...     '''Weight (kg) based on volume (m^3).'''
...     data = self.data
...     rho = data['rho'] # Density (kg/m^3)
...     vol = data['vol'] # Volume (m^3)
...     return rho * vol
>>>
>>> @Weight.setter
>>> def Weight(self, weight):
...     data = self.data
...     rho = data['rho'] # Density (kg/m^3)
...     data['vol'] = weight / rho
```

Create dictionaries of data and initialize new properties:

```
>>> water_data = {'rho': 1000, 'vol': 3}
>>> ethanol_data = {'rho': 789, 'vol': 3}
>>> weight_water = Weight('Water', water_data)
>>> weight_ethanol = Weight('Ethanol', ethanol_data)
>>> weight_water
Weight(Water) -> 3000 (kg)
>>>weight_ethanol
Weight(Ethanol) -> 2367 (kg)
```

Create a property_array from data:

```
>>> prop_arr = property_array([weight_water, weight_water])
property_array([3000, 2367])
```

Changing the values of a property_array changes the value of its properties:

```
>>> # Addition in place
>>> prop_arr += 3000
>>> prop_arr
property_array([6000, 5367])
>>> # Note how the data also changes
>>> water_data
{'rho': 1000, 'vol': 6.0}
>>> ethanol_data
{'rho': 789, 'vol': 6.802281368821292}
>>> # Setting an item changes the property value
>>> prop_arr[1] = 2367
>>> ethanol_data
{'rho': 789, 'vol': 3}
```

New arrays have no connection to the property_array:

```
>>> prop_arr - 1000 #  Returns a new array
array([5000.0, 1367.0], dtype=object)
>>> water_data #  Data remains unchanged
{'rho': 1000, 'vol': 6.0}
```

A representative DataFrame can also be made from the property_array:

```
>>> prop_arr.table()
        Weight (kg)
Water        6000.0
Ethanol      2367.0
```

Note: The DataFrame object contains the values of the properties, not the FreeProperty objects as a property_array would.

---

table(*title=''*, *with_units=True*)

Create a representative DataFrame object.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index