dataclass_dict

Leandro (Cerberus1746) Benedet Garcia

Nov 30, 2020

CONTENTS

All this package does is create a dataclass so you can straight up use it as if it was a dictionary. Here's a basic example:

```
from dataclass_dict import create_dataclass_dict

# Generate a instance
instance = create_dataclass_dict({"name": "Test", "value": 10})

# prints "Test"
print(instance.name)

# also prints "Test"
print(instance["name"])

# prints "Test" and deletes the field "name"
print(instance.pop("name")
```

Also, you can automatically generate a dataclass with a json like this:

```
from dataclass_dict import dataclass_from_json

json_code = """{
    "name": "Test",
    "value": 10
}"""

instance = dataclass_from_json(json_code)
```

If you'd like, you can load a json file straight from a url like this:

```
from dataclass_dict import dataclass_from_url

dataclass_from_url("json_url")
```

Plus, if you pass multiple parameters this way:

```
from dataclass_dict import dataclass_from_url

dataclass_from_url("json_url_1", "json_url_2")
```

They will be downloaded at the same time using threads.

Keep in mind that all parameters from the function `json.dumps()` works with the `dataclass_from_json()` and `dataclass_from_url()` so you can write special parsers.

CHAPTER

ONE

API REFERENCE

1.1 Dataclass Dict

created 2019-09-28

author Leandro (Cerberus1746) Benedet Garcia

class dataclass_dict.**DataclassDict**(*_, ***kwargs: Dict[str, Any]*)

Bases: collections.abc.MutableMapping, collections.abc.KeysView

The dataclass dict. It automatically transforms any class that inherits it into a dataclass.

__contains__(*field_name*)

Return true if the field exist inside the input dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to check
- **field_name** (str) – The name of the field to check

Return type bool

__delitem__(*field_name*, *default=None*)

Remove the field from the dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to delete the field from
- **field_name** (str) – The field name to delete
- **default** (Optional[Any]) – the value to be returned if the field doesn't exist

Raises **KeyError** – If default is *None* and the field doesn't exist

Return type Any

classmethod __init_subclass__(***kwargs*)

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

static __new__(*cls*, *_, ***kwargs*)

Create and return a new object. See help(type) for accurate signature.

Return type *DataclassDict*

__setattr__(*key*, *value*)

Implement setattr(self, name, value).

pop(*field_name*, *default=None*)
Remove the field from the dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to delete the field from
- **field_name** (str) – The field name to delete
- **default** (Optional[Any]) – the value to be returned if the field doesn't exist

Raises **KeyError** – If default is *None* and the field doesn't exist

Return type Any

update_from_json(*json_input*)
Exactly like update() but it loads data from a json string.

Parameters **json_input** (str) – The input json string.

dataclass_dict.**add_field**(*dataclass_instance*, *field_name*, *field_type*, *field_value=None*)
Create a new dataclass field

Parameters

- **dataclass_instance** (object) – The input dataclass
- **field_name** (str) – The name of the field
- **field_type** (Type[Any]) – The field type
- **field_value** (Optional[Any]) – The value of the field

dataclass_dict.**check_field**(*dataclass_instance*, *field_name*)
Return true if the field exist inside the input dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to check
- **field_name** (str) – The name of the field to check

Return type bool

dataclass_dict.**delete_field**(*dataclass_instance*, *field_name*, *default=None*)
Remove the field from the dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to delete the field from
- **field_name** (str) – The field name to delete
- **default** (Optional[Any]) – the value to be returned if the field doesn't exist

Raises **KeyError** – If default is *None* and the field doesn't exist

Return type Any

dataclass_dict.**load_json_from_url**(**urls*, ***kwargs*)
Load one or more json from the urls

Parameters **urls** (str) – one or more urls to be loaded

Return type Union[List[Union[Dict[str, Any], Any]], Dict[str, Any], List[Union[List[Union[Dict[str, Any], Any]], Dict[str, Any]]]]

1.2 Threading

dataclass_dict.threaded_request.**load_json_from_url**(**urls*, ***kwargs*)
Load one or more json from the urls

Parameters **urls** (str) – one or more urls to be loaded

Return type Union[List[Union[Dict[str, Any], Any]], Dict[str, Any], List[Union[List[Union[Dict[str, Any], Any]], Dict[str, Any]]]]

dataclass_dict.threaded_request.**load_url**(**urls*)
Load one or more urls executing them from threads

Parameters **urls** (str) – one or more urls to be loaded

Return type List[ThreadedGetData]

1.3 Utils

created 2019-07-29

author Leandro (Cerberus1746) Benedet Garcia

dataclass_dict.utils.**add_field**(*dataclass_instance*, *field_name*, *field_type*, *field_value=None*)
Create a new dataclass field

Parameters

- **dataclass_instance** (object) – The input dataclass
- **field_name** (str) – The name of the field
- **field_type** (Type[Any]) – The field type
- **field_value** (Optional[Any]) – The value of the field

dataclass_dict.utils.**check_field**(*dataclass_instance*, *field_name*)
Return true if the field exist inside the input dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to check
- **field_name** (str) – The name of the field to check

Return type bool

dataclass_dict.utils.**delete_field**(*dataclass_instance*, *field_name*, *default=None*)
Remove the field from the dataclass

Parameters

- **dataclass_instance** (object) – The dataclass to delete the field from
- **field_name** (str) – The field name to delete
- **default** (Optional[Any]) – the value to be returned if the field doesn't exist

Raises **KeyError** – If default is *None* and the field doesn't exist

Return type Any

dataclass_dict.utils.**item_zip**(**dicts_input*)

Function to iterate across multiple dictionaries

An example in how to use this function is:

```
first_dict = {"first": 1}
second_dict = {"second": 2}

for first_key, first_var, second_key, second_var in item_zip(first_dict , second_
↪dict):
    #prints first, 1
    print(first_key, first_var)
    #prints second, 2
    print(second_key, second_var)
```

dataclass_dict.utils.**valid_variable**(*name*)

Check if string is a valid keyword name

Parameters **name** – the string to be checked

CHAPTER

TWO

CHANGELOGS

2.1 0.0.7

2.1.1 Changed

- Improved packaging, coverage, tox and tests

2.2 0.0.6

2.2.1 Changed

- The software will now tell which variable has a invalid name.

2.3 0.0.5

2.3.1 Added

- If the creation dictionary have a string that is not a valid variable name, it will raise an `AssertionError`
- *`valid_variable()`* was added

2.4 0.0.4

2.4.1 Fixed

- Now, the package should be able to be installed normally. The package name was being identified as src

2.5 0.0.3

2.5.1 Added

- Any attribute, including parent ones, that starts with underscore will be ignored.
- `item_zip()` was added. It iterates between more than one `dict`.

2.5.2 Fixed

- Now, indeed, anything started with underscore will be ignored.

2.6 0.0.2

2.6.1 Changed

- Any attribute starting with a _ will not be added to the dataclass, but will be available normally

2.6.2 Fixed

- `__new__()` and `__init_subclass__()` now calls their parents with `super()`
- If a class inherited `DataclassDict` and it didn't have a field with annotations, it would raise an error. That's fixed now.

2.7 0.0.1

2.7.1 Added

- Package released
- Added Tidelif information in the readme

2.7.2 Fixed

- Fixed informations in the *setup.py* file such as descriptions and repository link.

PYTHON MODULE INDEX

d

INDEX

Symbols

A

C

D

I

L

M

P

U

V