Decent Documentation

Release 0.1.0.dev1

Veeti Paananen

September 24, 2015

Contents

Decent is a data validation library for Python 2 & 3. It comes with a schema validator and many useful validation functions for common cases.

CHAPTER 1

Sections

1.1 Introduction

Decent is a data validation library that can be used for many purposes. For example, a REST API can use it to validate and transform input data from the user.

Validation is designed around ordinary callable functions. A validator callable takes a single value as its argument. It must check it for validity and raise an error if necessary. Otherwise, it should return the result value: either a new value based on the input, or simply the input as it was.

To validate key-value data like dictionaries, the *decent.schema.Schema* class is used. A schema can be built to validate specified keys using validators.

Decent comes with many useful built-in validators to be used in schemas or on their own. Of course, you can also create your own.

1.1.1 Example

```
from decent import *

User = Schema({
    'username': All(NotEmpty(), Strip(), Length(max=32)),
    'password': All(NotEmpty(), Length(min=10)),
    Optional('bio'): Length(max=1000),
})
```

This is a schema with three fields: the username, password and bio. The bio key is optional, and can be omitted from input data without raising an error.

Every key must map to a validator callable. Here multiple validators are combined into one with the *decent.validators.All()* helper.

The resulting `User` schema is a validator callable like any other. It can be used with input data:

```
result = User({
    'username': " user",
    'password': "1234567890",
})
```

The result is a dictionary of all the result values. For example, here the username is passed through the built-in *decent.validators.Strip()* validator. This removes extra whitespace from the username. We can look at the result value and find that it works as expected:

```
result['username']
>>>> "user"
```

If any errors are encountered, an *decent.error.Invalid* exception is raised. For example:

```
User({
    'username': "",
    'password': "short",
})

>>>> decent.error.Invalid: Must not be empty, Must have a length of at least 10
```

Simple, right?

1.2 Schemas

The *decent.schema.Schema* class validates key-to-value mappings (typically `dict` instances), collecting result values and possible errors.

1.2.1 Creating and using a schema

The only required argument for constructing a schema is, well, a `schema`. It must be a `dict` mapping expected keys to validator callables. A created schema is a validator callable like any other:

```
schema = Schema({
    'number': Is(int),
})

schema(123)
>>>> 123
```

This allows for easy nesting of schemas inside schemas.

1.2.2 Default values

Keys can be wrapped with the *decent.schema.Default* marker to give them a default value:

```
Schema({
    Default('name', default='John Doe'): validator,
})
```

This value will be used if the key is not present. The `default` argument is also available in other key markers.

1.2.3 Optional keys

Keys can be wrapped with the *decent.schema.Optional* marker to make them optional:

```
Schema({
    Optional('name'): validator,
})
```

The given validator will only be called if the name key is present in the input data.

1.2.4 Extra keys

If the schema encounters unknown keys, it will silently drop them from the output by default. It can also be configured to pass them through as-is or raise an error: see the `extra_keys` constructor argument.

1.2.5 Validating the entire result

The schema can also have a so-called `entire` validator that can be used to further validate and/or change the resulting data. For example, this can be useful if a validation depends on multiple fields.

```
def entire(data):
    # data is the result dictionary after all keys have been processed
    # and validated. Extra validation errors can be raised here, and
    # the data can be modified.
    return data
schema = Schema({ ... }, entire=entire)
```

The validator is always called: even if individual fields have failed earlier. In this case, failed fields will not be included in the data.

1.3 Validators

Validators are ordinary callables that take a single argument (the input value) and return a transformed result value. They can also raise validation errors.

IValidator(*value*)
Validates the input `value` and possibly transforms it.

Returns the new value. If the value is invalid, raises a *decent.error.Error* or *decent.error.Invalid* exception.

Validator callables can be used inside schemas, but also standalone.

1.3.1 Built-in validators

Decent ships with many built-in validators that can be used and combined for usual tasks. Instances of them are built through the following constructors:

Helpers & Building blocks

decent.validators.**All**(**validators*)
Combines all the given validator callables into one, running all the validators in sequence on the given value.

decent.validators.**Any**(**validators*)
Combines all the given validator callables into one, running the given value through them in sequence until a valid result is given.

decent.validators.**Default**(*default*)
Creates a validator callable that replaces `None` with the specified default value.

decent.validators.**Maybe**(*validator*)
Wraps the given validator callable, only using it for the given value if it is not `None`.

decent.validators.**Msg**(*validator*, *message*)
Wraps the given validator callable, replacing any error messages raised.

Basics

decent.validators.**Eq**(*value*, *message='Not equal to {!s}'*)
Creates a validator that compares the equality of the given value to `value`.

A custom message can be specified with `message`. It will be formatted with `value`.

decent.validators.**Coerce**(*type*, *message='Not a valid {} value'*)
Creates a validator that attempts to coerce the given value to the specified `type`. Will raise an error if the coercion fails.

A custom message can be specified with `message`.

decent.validators.**Instance**(*expected*, *message='Not an instance of {}'*)
Creates a validator that checks if the given value is an instance of `expected`.

A custom message can be specified with `message`.

decent.validators.**Type**(*expected*, *message='Not of type {}'*)
Creates a validator that compares the type of the given value to `expected`. This is a direct type() equality check. Also see `Instance`, which is an isinstance() check.

A custom message can be specified with `message`.

Collections

decent.validators.**List**(*validator*)
Creates a validator that runs the given validator on every item in a list or other collection. The validator can mutate the values.

Any raised errors will be collected into a single `Invalid` error. Their paths will be replaced with the index of the item. Will raise an error if the input value is not iterable.

decent.validators.**Length**(*min=None*, *max=None*, *min_message='Must have a length of at least {min}'*, *max_message='Must have a length of at most {max}'*)
Creates a validator that checks if the given value's length is in the specified range, inclusive. (Returns the original value.)

See *Range()*.

Booleans

decent.validators.**Boolean**()
Creates a validator that attempts to convert the given value to a boolean or raises an error. The following rules are used:

`None` is converted to `False`.

`int` values are `True` except for `0`.

`str` values converted in lower- and uppercase:

- `y, yes, t, true`
- `n, no, f, false`

Numbers

decent.validators.**Range**(*min=None*, *max=None*, *min_message='Must be at least {min}'*, *max_message='Must be at most {max}'*)
Creates a validator that checks if the given numeric value is in the specified range, inclusive.

Accepts values specified by `numbers.Number` only, excluding booleans.

The error messages raised can be customized with `min_message` and `max_message`. The `min` and `max` arguments are formatted.

decent.validators.**Length**(*min=None*, *max=None*, *min_message='Must have a length of at least {min}'*, *max_message='Must have a length of at most {max}'*)
Creates a validator that checks if the given value's length is in the specified range, inclusive. (Returns the original value.)

See *Range()*.

Strings

decent.validators.**Lower**()
Creates a validator that converts the input string to lowercase. Will raise an error for non-string types.

decent.validators.**Upper**()
Creates a validator that converts the input string to UPPERCASE. Will raise an error for non-string types.

decent.validators.**Strip**()
Creates a validator that strips the input string of whitespace. Will raise an error for non-string types.

decent.validators.**Length**(*min=None*, *max=None*, *min_message='Must have a length of at least {min}'*, *max_message='Must have a length of at most {max}'*)
Creates a validator that checks if the given value's length is in the specified range, inclusive. (Returns the original value.)

See *Range()*.

String conversions

decent.validators.**Uuid**(*to_uuid=True*)
Creates a UUID validator. Will raise an error for non-string types and non-UUID values.

The given value will be converted to an instance of `uuid.UUID` unless `to_uuid` is `False`.

1.4 Errors

The main error type in Decent is *decent.error.Error*. It contains an error `message` as a description of the validation failure and a `path` list pointing to the erroneous field.

A subclass of `Error` called *decent.error.Invalid* can contain multiple errors. If you're working with schemas, you'll probably want to catch this error.

1.4.1 Error paths

The `path` field of an error is a list of nodes leading to the erroneous field. For example, a validation error on a field called `password` inside a `user` schema would have a path of `['user', 'field']`.

Error paths are typically populated automatically. For example, *decent.schema.Schema* automatically sets error paths for raised errors. Likewise, the built-in *decent.validators.List()* validator will prepend list indexes to all raised errors.

1.4.2 Overriding messages

You can use the built-in *decent.validators.Msg()* helper to override the error messages raised by any validator. For example:

```
password_validator = Msg(
    Length(min=10),
    "Pick a passwords with 10 characters or more to be safe."
)
```

Many built-in validators also have granular arguments for overriding their messages.

CHAPTER 2

API

2.1 API listing

All submodule contents are available on the top-level `decent` module.

2.1.1 decent.schema

class decent.schema.**Schema**(*schema*, *entire=None*, *extra_keys='IGNORE'*, *required_error=None*)
A schema that validates data given to it using the specified rules.

The `schema` must be a dictionary of key-value mappings. Values must be callable validators. See XX.

The `entire` argument allows specifying a callable validator that runs on the entire input after every field is validated. If provided, the validator will always run, even if validation errors are raised beforehand. Failed keys will not be included in the given data.

The `extra_keys` argument must be one of *ACCEPT*, *IGNORE* or *REJECT*.

The `required_error` argument specifies the error message used when a key is missing. *REQUIRED_ERROR* is the default.

ACCEPT = 'ACCEPT'

IGNORE = 'IGNORE'

REJECT = 'REJECT'

REJECT_ERROR = 'This field is unknown.'

REQUIRED_ERROR = 'This field is required.'

__call__(*data*)
Validates the given `data` dictionary and returns transformed values.

Will raise *decent.error.Invalid* if any validation errors are encountered.

class decent.schema.**Marker**(*key*, *default=None*)
A base class for key markers that wrap a key.

class decent.schema.**Default**(*key*, *default=None*)
A marker for specifying a default value for a key.

class decent.schema.**Optional**(*key*, *default=None*)
A marker for specifying a key as optional. The schema will validate data without the key present.

2.1.2 decent.validators

decent.validators.**All**(**validators*)
Combines all the given validator callables into one, running all the validators in sequence on the given value.

decent.validators.**Any**(**validators*)
Combines all the given validator callables into one, running the given value through them in sequence until a valid result is given.

decent.validators.**Boolean**()
Creates a validator that attempts to convert the given value to a boolean or raises an error. The following rules are used:

`None` is converted to `False`.

`int` values are `True` except for `0`.

`str` values converted in lower- and uppercase:

- `y, yes, t, true`
- `n, no, f, false`

decent.validators.**Coerce**(*type*, *message='Not a valid {} value'*)
Creates a validator that attempts to coerce the given value to the specified `type`. Will raise an error if the coercion fails.

A custom message can be specified with `message`.

decent.validators.**Default**(*default*)
Creates a validator callable that replaces `None` with the specified default value.

decent.validators.**Eq**(*value*, *message='Not equal to {!s}'*)
Creates a validator that compares the equality of the given value to `value`.

A custom message can be specified with `message`. It will be formatted with `value`.

decent.validators.**Instance**(*expected*, *message='Not an instance of {}'*)
Creates a validator that checks if the given value is an instance of `expected`.

A custom message can be specified with `message`.

decent.validators.**Length**(*min=None*, *max=None*, *min_message='Must have a length of at least {min}'*, *max_message='Must have a length of at most {max}'*)
Creates a validator that checks if the given value's length is in the specified range, inclusive. (Returns the original value.)

See *Range()*.

decent.validators.**List**(*validator*)
Creates a validator that runs the given validator on every item in a list or other collection. The validator can mutate the values.

Any raised errors will be collected into a single `Invalid` error. Their paths will be replaced with the index of the item. Will raise an error if the input value is not iterable.

decent.validators.**Lower**()
Creates a validator that converts the input string to lowercase. Will raise an error for non-string types.

decent.validators.**Maybe**(*validator*)
Wraps the given validator callable, only using it for the given value if it is not `None`.

decent.validators.**Msg**(*validator*, *message*)
Wraps the given validator callable, replacing any error messages raised.

decent.validators.**NotEmpty**()
Creates a validator that validates the given string is not empty. Will raise an error for non-string types.

decent.validators.**Range**(*min=None*, *max=None*, *min_message='Must be at least {min}'*, *max_message='Must be at most {max}'*)
Creates a validator that checks if the given numeric value is in the specified range, inclusive.

Accepts values specified by `numbers.Number` only, excluding booleans.

The error messages raised can be customized with `min_message` and `max_message`. The `min` and `max` arguments are formatted.

decent.validators.**Strip**()
Creates a validator that strips the input string of whitespace. Will raise an error for non-string types.

decent.validators.**Type**(*expected*, *message='Not of type {}'*)
Creates a validator that compares the type of the given value to `expected`. This is a direct type() equality check. Also see `Instance`, which is an isinstance() check.

A custom message can be specified with `message`.

decent.validators.**Upper**()
Creates a validator that converts the input string to UPPERCASE. Will raise an error for non-string types.

decent.validators.**Uuid**(*to_uuid=True*)
Creates a UUID validator. Will raise an error for non-string types and non-UUID values.

The given value will be converted to an instance of `uuid.UUID` unless `to_uuid` is `False`.

2.1.3 decent.error

exception decent.error.**DecentError**
Bases: `Exception`

exception decent.error.**Error**(*message*, *path=None*)
Bases: *decent.error.DecentError*

A single validation error.

The `message` contains an explanation for the error: for example, "this value must be at least 10 characters long".

The `path` is a list of keys to the field this error is for. This is usually automatically set by the *decent.schema.Schema* and/or validator callable being used.

as_dict(*join='.'*)
Returns the error as a path to message dictionary. Paths are joined with the `join` string.

messages

paths

exception decent.error.**Invalid**(*errors=None*)
Bases: *decent.error.Error*

A collection of one or more validation errors for a schema.

append(*error*)

as_dict(*join='.'*)
Returns all the errors in this collection as a path to message dictionary. Paths are joined with the `join` string.

message
The first error message in this collection.

messages
The list of error messages in this collection.

path
The first error path in this collection.

paths
The list of error paths in this collection.

exception decent.error.**SchemaError**
Bases: *decent.error.DecentError*

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d

Symbols

A

B

C

D

E

I

L

M

N

O

P

R

S

T

U