Welcome to voluptuous's documentation!

pypi v0.15.2 | license BSD-3-Clause | python 3.9 | 3.10 | 3.11 | 3.12 | Tests passing

Voluptuous, despite the name, is a Python data validation library. It is primarily intended for validating data coming into Python as JSON, YAML, etc.

It has three goals:

1. Simplicity.
2. Support for complex data structures.
3. Provide useful error messages.

Content

- voluptuous
  - voluptuous package

Document version: 0.14.2

Indices and tables

- Index
- Module Index
- Search Page

voluptuous

- voluptuous package
  - Submodules
  - voluptuous.error module
    - `AllInvalid`
    - `AnyInvalid`
    - `BooleanInvalid`
    - `CoerceInvalid`
    - `ContainsInvalid`
    - `DateInvalid`
    - `DatetimeInvalid`
    - `DictInvalid`
    - `DirInvalid`
    - `EmailInvalid`
    - `Error`
    - `ExactSequenceInvalid`
    - `ExclusiveInvalid`
    - `FalseInvalid`
    - `FileInvalid`
    - `InInvalid`
    - `InclusiveInvalid`
    - `Invalid`
      - `Invalid.error_message`
      - `Invalid.msg`
      - `Invalid.path`
      - `Invalid.prepend()`
    - `LengthInvalid`
    - `LiteralInvalid`
    - `MatchInvalid`
    - `MultipleInvalid`
      - `MultipleInvalid.add()`
      - `MultipleInvalid.error_message`
      - `MultipleInvalid.msg`
      - `MultipleInvalid.path`
      - `MultipleInvalid.prepend()`
    - `NotEnoughValid`
    - `NotInInvalid`
    - `ObjectInvalid`
    - `PathInvalid`
    - `RangeInvalid`
    - `RequiredFieldInvalid`
    - `ScalarInvalid`
    - `SchemaError`
    - `SequenceTypeInvalid`
    - `TooManyValid`
    - `TrueInvalid`
    - `TypeInvalid`
    - `UrlInvalid`
    - `ValueInvalid`

- voluptuous.humanize module
  - humanize_error()
  - validate_with_humanized_errors()
- voluptuous.schema_builder module
  - Exclusive
  - Extra()
  - Inclusive
  - Marker
  - Msg
  - Object
  - Optional
  - Remove
  - Required
  - Schema
    - Schema.extend()
    - Schema.infer()
  - Self()
  - Undefined
  - VirtualPathComponent
  - default_factory()
  - extra()
  - message()
  - raises()
  - validate()
- voluptuous.util module
  - Capitalize()
  - DefaultTo
  - Literal
  - Lower()
  - Set
  - SetTo
  - Strip()
  - Title()
  - Upper()
- voluptuous.validators module
  - All
  - And
  - Any
  - Boolean()
  - Clamp
  - Coerce
  - Contains
  - Date
    - Date.DEFAULT_FORMAT
  - Datetime
    - Datetime.DEFAULT_FORMAT
  - Email()
  - Equal

- `ExactSequence`
- `FqdnUrl()`
- `In`
- `IsDir()`
- `IsFalse()`
- `IsFile()`
- `IsTrue()`
- `Length`
- `Match`
- `Maybe()`
- `NotIn`
- `Number`
- `Or`
- `PathExists()`
- `Range`
- `Replace`
- `SomeOf`
- `Switch`
- `Union`
- `Unique`
- `Unordered`
- `Url()`
- `truth()`

voluptuous package

Submodules

voluptuous.error module

exception voluptuous.error.**AllInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value did not pass all validators.

exception voluptuous.error.**AnyInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value did not pass any validator.

exception voluptuous.error.**BooleanInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a boolean.

exception voluptuous.error.**CoerceInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

Impossible to coerce value to type.

exception voluptuous.error.**ContainsInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

List does not contain item

exception voluptuous.error.**DateInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a formatted date string.

exception voluptuous.error.**DatetimeInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a formatted datetime string.

exception voluptuous.error.**DictInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value found was not a dict.

exception voluptuous.error.**DirInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a directory.

exception voluptuous.error.**EmailInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not an email address.

exception voluptuous.error.**Error** [source]

Bases: **Exception**

Base validation exception.

exception voluptuous.error.**ExactSequenceInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

exception voluptuous.error.**ExclusiveInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

More than one value found in exclusion group.

exception voluptuous.error.**FalseInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not False.

exception voluptuous.error.**FileInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a file.

exception voluptuous.error.**InInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

exception voluptuous.error.**InclusiveInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

Not all values found in inclusion group.

exception voluptuous.error.**Invalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Error**

The data was invalid.

Attr msg:	The error message.
Attr path:	The path to the error, as a list of keys in the source data.
Attr error_message:	The actual error message that was raised, as a string.

property **error_message**: *str*

property **msg**: *str*

property **path**: *List[Hashable]*

prepend(*path: List[Hashable]*) → None [source]

exception voluptuous.error.**LengthInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

exception voluptuous.error.**LiteralInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The literal values do not match.

exception voluptuous.error.**MatchInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value does not match the given regular expression.

exception voluptuous.error.**MultipleInvalid**(*errors: List[Invalid] | None = None*) [source]

Bases: **Invalid**

add(*error: Invalid*) → None [source]

property **error_message**: *str*

property **msg**: *str*

property **path**: *List[Hashable]*

prepend(*path: List[Hashable]*) → None [source]

exception voluptuous.error.**NotEnoughValid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value did not pass enough validations.

exception voluptuous.error.**NotInInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

exception voluptuous.error.**ObjectInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value we found was not an object.

exception voluptuous.error.**PathInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a path.

exception voluptuous.error.**RangeInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not in given range.

exception voluptuous.error.**RequiredFieldInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

Required field was missing.

exception voluptuous.error.**ScalarInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

Scalars did not match.

exception voluptuous.error.**SchemaError** [source]

Bases: **Error**

An error was encountered in the schema.

exception voluptuous.error.**SequenceTypeInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The type found is not a sequence type.

exception voluptuous.error.**TooManyValid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value passed more than expected validations.

exception voluptuous.error.**TrueInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not True.

exception voluptuous.error.**TypeInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value was not of required type.

exception voluptuous.error.**UrlInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value is not a URL.

exception voluptuous.error.**ValueInvalid**(*message: str, path: List[Hashable] | None = None, error_message: str | None = None, error_type: str | None = None*) [source]

Bases: **Invalid**

The value was found invalid by evaluation function.

voluptuous.humanize module

voluptuous.humanize.**humanize_error**(*data, validation_error: Invalid, max_sub_error_length: int = 500*) → str [source]

Provide a more helpful + complete validation error message than that provided automatically Invalid and MultipleInvalid do not include the offending value in error messages, and MultipleInvalid.__str__ only provides the first error.

voluptuous.humanize.**validate_with_humanized_errors**(*data, schema: Schema, max_sub_error_length: int = 500*) → Any [source]

voluptuous.schema_builder module

class voluptuous.schema_builder.**Exclusive**(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, group_of_exclusion: str, msg: str | None = None, description: str | None = None*) [source]

Bases: **Optional**

Mark a node in the schema as exclusive.

Exclusive keys inherited from Optional:

```
>>> schema = Schema({Exclusive('alpha', 'angles'): int, Exclusive('beta', 'angles
>>> schema({'alpha': 30})
{'alpha': 30}
```

Keys inside a same group of exclusion cannot be together, it only makes sense for dictionaries:

```
>>> with raises(er.MultipleInvalid, "two or more values in the same group of excl
...   schema({'alpha': 30, 'beta': 45})
```

For example, API can provides multiple types of authentication, but only one works in the same time:

```
>>> msg = 'Please, use only one type of authentication at the same time.'
>>> schema = Schema({
```

```
... Exclusive('classic', 'auth', msg=msg):{
...     Required('email'): str,
...     Required('password'): str
...     },
... Exclusive('internal', 'auth', msg=msg):{
...     Required('secret_key'): str
...     },
... Exclusive('social', 'auth', msg=msg):{
...     Required('social_network'): str,
...     Required('token'): str
...     }
... })
```

```
>>> with raises(er.MultipleInvalid, "Please, use only one type of authentication
...     schema({'classic': {'email': 'foo@example.com', 'password': 'bar'},
...             'social': {'social_network': 'barfoo', 'token': 'tEMp'}})
```

`voluptuous.schema_builder.`**`Extra`**`(_) → None` [source]

Allow keys in the data that are not present in the schema.

class `voluptuous.schema_builder.`**`Inclusive`**`(`*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, group_of_inclusion: str, msg: str | None = None, description: str | None = None, default=...*`)` [source]

Bases: **Optional**

Mark a node in the schema as inclusive.

Inclusive keys inherited from Optional:

```
>>> schema = Schema({
...     Inclusive('filename', 'file'): str,
...     Inclusive('mimetype', 'file'): str
... })
>>> data = {'filename': 'dog.jpg', 'mimetype': 'image/jpeg'}
>>> data == schema(data)
True
```

Keys inside a same group of inclusive must exist together, it only makes sense for dictionaries:

```
>>> with raises(er.MultipleInvalid, "some but not all values in the same group o
...     schema({'filename': 'dog.jpg'})
```

If none of the keys in the group are present, it is accepted:

```
>>> schema({})
{}
```

For example, API can return ‘height’ and ‘width’ together, but not separately.

```
>>> msg = "Height and width must exist together"
>>> schema = Schema({
...     Inclusive('height', 'size', msg=msg): int,
...     Inclusive('width', 'size', msg=msg): int
... })
```

```
>>> with raises(er.MultipleInvalid, msg + " @ data[<size>]"):
...     schema({'height': 100})
```

```
>>> with raises(er.MultipleInvalid, msg + " @ data[<size>]"):
...     schema({'width': 100})
```

```
>>> data = {'height': 100, 'width': 100}
>>> data == schema(data)
True
```

class voluptuous.schema_builder.**Marker**(*schema_: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, msg: str | None = None, description: str | None = None*) [source]

Bases: **object**

Mark nodes for special treatment.

description is an optional field, unused by Voluptuous itself, but can be introspected by any external tool, for example to generate schema documentation.

class voluptuous.schema_builder.**Msg**(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, msg: str, cls: Type[Error] | None = None*) [source]

Bases: **object**

Report a user-friendly message if a schema fails to validate.

```
>>> validate = Schema(
...   Msg(['one', 'two', int],
...       'should be one of "one", "two" or an integer'))
>>> with raises(er.MultipleInvalid, 'should be one of "one", "two" or an integer
...   validate(['three'])
```

Messages are only applied to invalid direct descendants of the schema:

```
>>> validate = Schema(Msg([['one', 'two', int]], 'not okay!'))
>>> with raises(er.MultipleInvalid, 'expected int @ data[0][0]'):
...   validate([['three']])
```

The type which is thrown can be overridden but needs to be a subclass of Invalid

```
>>> with raises(er.SchemaError, 'Msg can only use subclases of Invalid as custom
...   validate = Schema(Msg([int], 'should be int', cls=KeyError))
```

If you do use a subclass of Invalid, that error will be thrown (wrapped in a MultipleInvalid)

```
>>> validate = Schema(Msg([['one', 'two', int]], 'not okay!', cls=er.RangeInvalid
>>> try:
...  validate(['three'])
... except er.MultipleInvalid as e:
...   assert isinstance(e.errors[0], er.RangeInvalid)
```

class voluptuous.schema_builder.**Object**(*schema: Any, cls: object = ...*) [source]

Bases: **dict**

Indicate that we should work with attributes, not keys.

class voluptuous.schema_builder.**Optional**(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, msg: str | None = None, default=..., description: str | None = None*) [source]

Bases: **Marker**

Mark a node in the schema as optional, and optionally provide a default

```
>>> schema = Schema({Optional('key'): str})
>>> schema({})
{}
>>> schema = Schema({Optional('key', default='value'): str})
>>> schema({})
{'key': 'value'}
>>> schema = Schema({Optional('key', default=list): list})
>>> schema({})
{'key': []}
```

If 'required' flag is set for an entire schema, optional keys aren't required

```
>>> schema = Schema({
...     Optional('key'): str,
...     'key2': str
... }, required=True)
>>> schema({'key2':'value'})
{'key2': 'value'}
```

class voluptuous.schema_builder.**Remove**(*schema_: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, msg: str | None = None, description: str | None = None*) [source]

Bases: **Marker**

Mark a node in the schema to be removed and excluded from the validated output. Keys that fail validation will not raise `Invalid`. Instead, these keys will be treated as extras.

```
>>> schema = Schema({str: int, Remove(int): str})
>>> with raises(er.MultipleInvalid, "extra keys not allowed @ data[1]"):
...     schema({'keep': 1, 1: 1.0})
>>> schema({1: 'red', 'red': 1, 2: 'green'})
{'red': 1}
>>> schema = Schema([int, Remove(float), Extra])
>>> schema([1, 2, 3, 4.0, 5, 6.0, '7'])
[1, 2, 3, 5, '7']
```

class voluptuous.schema_builder.**Required**(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, msg: str | None = None, default=..., description: str | None = None*) [source]

Bases: **Marker**

Mark a node in the schema as being required, and optionally provide a default value.

```
>>> schema = Schema({Required('key'): str})
>>> with raises(er.MultipleInvalid, "required key not provided @ data['key']"):
...   schema({})
```

```
>>> schema = Schema({Required('key', default='value'): str})
>>> schema({})
{'key': 'value'}
>>> schema = Schema({Required('key', default=list): list})
>>> schema({})
{'key': []}
```

class voluptuous.schema_builder.**Schema**(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, required: bool = False, extra: int = 0*) [source]

Bases: **object**

A validation schema.

The schema is a Python tree-like structure where nodes are pattern matched against corresponding trees of values.

Nodes can be values, in which case a direct comparison is used, types, in which case an isinstance() check is performed, or callables, which will validate and optionally convert the value.

We can equate schemas also.

For Example:

```
>>> v = Schema({Required('a'): str})
>>> v1 = Schema({Required('a'): str})
>>> v2 = Schema({Required('b'): str})
>>> assert v == v1
>>> assert v != v2
```

extend(*schema: Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable, required: bool | None = None, extra: int | None = None*) → Schema [source]

Create a new *Schema* by merging this and the provided *schema*.

Neither this *Schema* nor the provided *schema* are modified. The resulting *Schema* inherits the *required* and *extra* parameters of this, unless overridden.

Both schemas must be dictionary-based.

Parameters:
- **schema** – dictionary to extend this *Schema* with
- **required** – if set, overrides *required* of this *Schema*
- **extra** – if set, overrides *extra* of this *Schema*

classmethod **infer**(*data, **kwargs*) → Schema [source]

Create a Schema from concrete data (e.g. an API response).

For example, this will take a dict like:

{

‘foo’: 1, ‘bar’: {

‘a’: True, ‘b’: False

}, ‘baz’: [‘purple’, ‘monkey’, ‘dishwasher’]

}

And return a Schema:

{

‘foo’: int, ‘bar’: {

‘a’: bool, ‘b’: bool

}, ‘baz’: [str]

}

Note: only very basic inference is supported.

voluptuous.schema_builder.**Self**() → None [source]

class voluptuous.schema_builder.**Undefined** [source]

Bases: **object**

class voluptuous.schema_builder.**VirtualPathComponent** [source]

Bases: **str**

voluptuous.schema_builder.**default_factory**(*value*) → Undefined | Callable[[], Any] [source]

voluptuous.schema_builder.**extra**(_) → None

Allow keys in the data that are not present in the schema.

voluptuous.schema_builder.**message**(*default: str | None = None, cls: Type[Error] | None = None*) → Callable [source]

Convenience decorator to allow functions to provide a message.

Set a default message:

```
>>> @message('not an integer')
... def isint(v):
...   return int(v)
```

```
>>> validate = Schema(isint())
>>> with raises(er.MultipleInvalid, 'not an integer'):
...   validate('a')
```

The message can be overridden on a per validator basis:

```
>>> validate = Schema(isint('bad'))
>>> with raises(er.MultipleInvalid, 'bad'):
...   validate('a')
```

The class thrown too:

```
>>> class IntegerInvalid(er.Invalid): pass
>>> validate = Schema(isint('bad', clsoverride=IntegerInvalid))
>>> try:
...  validate('a')
... except er.MultipleInvalid as e:
...   assert isinstance(e.errors[0], IntegerInvalid)
```

voluptuous.schema_builder.**raises**(*exc, msg: str | None = None, regex: Pattern | None = None*) → Generator[None, None, None] [source]

voluptuous.schema_builder.**validate**(**a, **kw*) → Callable [source]

Decorator for validating arguments of a function against a given schema.

Set restrictions for arguments:

```
>>> @validate(arg1=int, arg2=int)
... def foo(arg1, arg2):
...   return arg1 * arg2
```

Set restriction for returned value:

```
>>> @validate(arg=int, __return__=int)
... def bar(arg1):
...   return arg1 * 2
```

voluptuous.util module

`voluptuous.util.`**`Capitalize`**`(v: str) → str` [source]

Capitalise a string.

```
>>> s = Schema(Capitalize)
>>> s('hello world')
'Hello world'
```

class `voluptuous.util.`**`DefaultTo`**`(default_value, msg: str | None = None)` [source]

Bases: **object**

Sets a value to default_value if none provided.

```
>>> s = Schema(DefaultTo(42))
>>> s(None)
42
>>> s = Schema(DefaultTo(list))
>>> s(None)
[]
```

class `voluptuous.util.`**`Literal`**`(lit)` [source]

Bases: **object**

`voluptuous.util.`**`Lower`**`(v: str) → str` [source]

Transform a string to lower case.

```
>>> s = Schema(Lower)
>>> s('HI')
'hi'
```

class `voluptuous.util.`**`Set`**`(msg: str | None = None)` [source]

Bases: **object**

Convert a list into a set.

```
>>> s = Schema(Set())
>>> s([]) == set([])
True
>>> s([1, 2]) == set([1, 2])
True
>>> with raises(Invalid, regex="^cannot be presented as set: "):
...   s([set([1, 2]), set([3, 4])])
```

class `voluptuous.util.`**`SetTo`**`(value)` [source]

Bases: **object**

Set a value, ignoring any previous value.

```
>>> s = Schema(validators.Any(int, SetTo(42)))
>>> s(2)
2
>>> s("foo")
42
```

`voluptuous.util.`**`Strip`**`(v: str) → str` [source]

Strip whitespace from a string.

```
>>> s = Schema(Strip)
>>> s('  hello world  ')
'hello world'
```

`voluptuous.util.Title(v: str) → str` [source]

Title case a string.

```
>>> s = Schema(Title)
>>> s('hello world')
'Hello World'
```

`voluptuous.util.Upper(v: str) → str` [source]

Transform a string to upper case.

```
>>> s = Schema(Upper)
>>> s('hi')
'HI'
```

voluptuous.validators module

class `voluptuous.validators.All(*validators, msg=None, required=False, discriminant=None, **kwargs)` [source]

Bases: **_WithSubValidators**

Value must pass all validators.

The output of each validator is passed as input to the next.

Parameters:
- **msg** – Message to deliver to user if validation fails.
- **kwargs** – All other keyword arguments are passed to the sub-schema constructors.

```
>>> validate = Schema(All('10', Coerce(int)))
>>> validate('10')
10
```

`voluptuous.validators.And`

alias of **All**

class `voluptuous.validators.Any(*validators, msg=None, required=False, discriminant=None, **kwargs)` [source]

Bases: **_WithSubValidators**

Use the first validated value.

Parameters:
- **msg** – Message to deliver to user if validation fails.
- **kwargs** – All other keyword arguments are passed to the sub-schema constructors.

Returns: Return value of the first validator that passes.

```
>>> validate = Schema(Any('true', 'false',
...                       All(Any(int, bool), Coerce(bool))))
>>> validate('true')
'true'
>>> validate(1)
True
>>> with raises(MultipleInvalid, "not a valid value"):
...   validate('moo')
```

msg argument is used

```
>>> validate = Schema(Any(1, 2, 3, msg="Expected 1 2 or 3"))
>>> validate(1)
1
```

```
>>> with raises(MultipleInvalid, "Expected 1 2 or 3"):
...   validate(4)
```

voluptuous.validators.**Boolean**(*v*) [source]

Convert human-readable boolean values to a bool.

Accepted values are 1, true, yes, on, enable, and their negatives. Non-string values are cast to bool.

```
>>> validate = Schema(Boolean())
>>> validate(True)
True
>>> validate("1")
True
>>> validate("0")
False
>>> with raises(MultipleInvalid, "expected boolean"):
...   validate('moo')
>>> try:
...  validate('moo')
... except MultipleInvalid as e:
...   assert isinstance(e.errors[0], BooleanInvalid)
```

class voluptuous.validators.**Clamp**(*min: int | float | None = None, max: int | float | None = None, msg: str | None = None*) [source]

Bases: **object**

Clamp a value to a range.

Either min or max may be omitted.

```
>>> s = Schema(Clamp(min=0, max=1))
>>> s(0.5)
0.5
>>> s(5)
1
>>> s(-1)
0
```

class voluptuous.validators.**Coerce**(*type: type | Callable, msg: str | None = None*) [source]

Bases: **object**

Coerce a value to a type.

If the type constructor throws a ValueError or TypeError, the value will be marked as Invalid.

Default behavior:

```
>>> validate = Schema(Coerce(int))
>>> with raises(MultipleInvalid, 'expected int'):
...   validate(None)
>>> with raises(MultipleInvalid, 'expected int'):
...   validate('foo')
```

With custom message:

```
>>> validate = Schema(Coerce(int, "moo"))
>>> with raises(MultipleInvalid, 'moo'):
...   validate('foo')
```

class voluptuous.validators.**Contains**(*item, msg: str | None = None*) [source]

Bases: **object**

Validate that the given schema element is in the sequence being validated.

```
>>> s = Contains(1)
>>> s([3, 2, 1])
[3, 2, 1]
>>> with raises(ContainsInvalid, 'value is not allowed'):
...   s([3, 2])
```

class voluptuous.validators.**Date**(*format: str | None = None, msg: str | None = None*) [source]

Bases: **Datetime**

Validate that the value matches the date format.

DEFAULT_FORMAT = *'%Y-%m-%d'*

class voluptuous.validators.**Datetime**(*format: str | None = None, msg: str | None = None*) [source]

Bases: **object**

Validate that the value matches the datetime format.

DEFAULT_FORMAT = *'%Y-%m-%dT%H:%M:%S.%fZ'*

voluptuous.validators.**Email**(*v*) [source]

Verify that the value is an email address or not.

```
>>> s = Schema(Email())
>>> with raises(MultipleInvalid, 'expected an email address'):
...   s("a.com")
>>> with raises(MultipleInvalid, 'expected an email address'):
...   s("a@.com")
>>> with raises(MultipleInvalid, 'expected an email address'):
...   s("a@.com")
>>> s('t@x.com')
't@x.com'
```

class voluptuous.validators.**Equal**(*target, msg: str | None = None*) [source]

Bases: **object**

Ensure that value matches target.

```
>>> s = Schema(Equal(1))
>>> s(1)
1
>>> with raises(Invalid):
...    s(2)
```

Validators are not supported, match must be exact:

```
>>> s = Schema(Equal(str))
>>> with raises(Invalid):
...     s('foo')
```

class voluptuous.validators.**ExactSequence**(*validators: Iterable[Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable], msg: str | None = None, **kwargs*) [source]

Bases: **object**

Matches each element in a sequence against the corresponding element in the validators.

Parameters:
- **msg** – Message to deliver to user if validation fails.
- **kwargs** – All other keyword arguments are passed to the sub-schema constructors.

```
>>> from voluptuous import Schema, ExactSequence
>>> validate = Schema(ExactSequence([str, int, list, list]))
>>> validate(['hourly_report', 10, [], []])
['hourly_report', 10, [], []]
>>> validate(('hourly_report', 10, [], []))
('hourly_report', 10, [], [])
```

`voluptuous.validators.`**`FqdnUrl`**`(v)` [source]

Verify that the value is a fully qualified domain name URL.

```
>>> s = Schema(FqdnUrl())
>>> with raises(MultipleInvalid, 'expected a fully qualified domain name URL'):
...   s("http://localhost/")
>>> s('http://w3.org')
'http://w3.org'
```

class `voluptuous.validators.`**`In`**`(container: Iterable, msg: str | None = None)` [source]

Bases: **`object`**

Validate that a value is in a collection.

`voluptuous.validators.`**`IsDir`**`(v)` [source]

Verify the directory exists.

```
>>> IsDir()('/')
'/'
>>> with raises(DirInvalid, 'Not a directory'):
...   IsDir()(None)
```

`voluptuous.validators.`**`IsFalse`**`(v)` [source]

Assert that a value is false, in the Python sense.

(see **`IsTrue()`** for more detail)

```
>>> validate = Schema(IsFalse())
>>> validate([])
[]
>>> with raises(MultipleInvalid, "value was not false"):
...   validate(True)
```

```
>>> try:
...  validate(True)
... except MultipleInvalid as e:
...   assert isinstance(e.errors[0], FalseInvalid)
```

`voluptuous.validators.`**`IsFile`**`(v)` [source]

Verify the file exists.

```
>>> os.path.basename(IsFile()(__file__)).startswith('validators.py')
True
>>> with raises(FileInvalid, 'Not a file'):
...   IsFile()("random_filename_goes_here.py")
>>> with raises(FileInvalid, 'Not a file'):
...   IsFile()(None)
```

`voluptuous.validators.`**`IsTrue`**`(v)` [source]

Assert that a value is true, in the Python sense.

```
>>> validate = Schema(IsTrue())
```

"In the Python sense" means that implicitly false values, such as empty lists, dictionaries, etc. are treated as "false":

```
>>> with raises(MultipleInvalid, "value was not true"):
...   validate([])
>>> validate([1])
[1]
>>> with raises(MultipleInvalid, "value was not true"):
...   validate(False)
```

…and so on.

```
>>> try:
...  validate([])
... except MultipleInvalid as e:
...   assert isinstance(e.errors[0], TrueInvalid)
```

class voluptuous.validators.**Length**(*min: int | float | None = None, max: int | float | None = None, msg: str | None = None*) [source]

Bases: **object**

The length of a value must be in a certain range.

class voluptuous.validators.**Match**(*pattern: Pattern | str, msg: str | None = None*) [source]

Bases: **object**

Value must be a string that matches the regular expression.

```
>>> validate = Schema(Match(r'^0x[A-F0-9]+$'))
>>> validate('0x123EF4')
'0x123EF4'
>>> with raises(MultipleInvalid, 'does not match regular expression ^0x[A-F0-9]+$
...   validate('123EF4')
```

```
>>> with raises(MultipleInvalid, 'expected string or buffer'):
...   validate(123)
```

Pattern may also be a compiled regular expression:

```
>>> validate = Schema(Match(re.compile(r'0x[A-F0-9]+', re.I)))
>>> validate('0x123ef4')
'0x123ef4'
```

voluptuous.validators.**Maybe**(*validator: Callable, msg: str | None = None*) [source]

Validate that the object matches given validator or is None.

Raises: **Invalid** – If the value does not match the given validator and is not None.

```
>>> s = Schema(Maybe(int))
>>> s(10)
10
>>> with raises(Invalid):
...  s("string")
```

class voluptuous.validators.**NotIn**(*container: Iterable, msg: str | None = None*) [source]

Bases: **object**

Validate that a value is not in a collection.

class voluptuous.validators.**Number**(*precision: int | None = None, scale: int | None = None, msg: str | None = None, yield_decimal: bool = False*) [source]

Bases: **object**

Verify the number of digits that are present in the number(Precision), and the decimal places(Scale).

Raises: **Invalid** – If the value does not match the provided Precision and Scale.

```
>>> schema = Schema(Number(precision=6, scale=2))
>>> schema('1234.01')
'1234.01'
>>> schema = Schema(Number(precision=6, scale=2, yield_decimal=True))
>>> schema('1234.01')
Decimal('1234.01')
```

voluptuous.validators.**Or**

alias of **Any**

voluptuous.validators.**PathExists**(*v*) [source]

Verify the path exists, regardless of its type.

```
>>> os.path.basename(PathExists()(__file__)).startswith('validators.py')
True
>>> with raises(Invalid, 'path does not exist'):
...   PathExists()("random_filename_goes_here.py")
>>> with raises(PathInvalid, 'Not a Path'):
...   PathExists()(None)
```

class voluptuous.validators.**Range**(*min: int | float | None = None, max: int | float | None = None, min_included: bool = True, max_included: bool = True, msg: str | None = None*) [source]

Bases: **object**

Limit a value to a range.

Either min or max may be omitted. Either min or max can be excluded from the range of accepted values.

Raises: **Invalid** – If the value is outside the range.

```
>>> s = Schema(Range(min=1, max=10, min_included=False))
>>> s(5)
5
>>> s(10)
10
>>> with raises(MultipleInvalid, 'value must be at most 10'):
...   s(20)
>>> with raises(MultipleInvalid, 'value must be higher than 1'):
...   s(1)
>>> with raises(MultipleInvalid, 'value must be lower than 10'):
...   Schema(Range(max=10, max_included=False))(20)
```

class voluptuous.validators.**Replace**(*pattern: Pattern | str, substitution: str, msg: str | None = None*) [source]

Bases: **object**

Regex substitution.

```
>>> validate = Schema(All(Replace('you', 'I'),
...                       Replace('hello', 'goodbye')))
```

```
>>> validate('you say hello')
'I say goodbye'
```

class voluptuous.validators.**SomeOf**(*validators: List[Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable], min_valid: int | None = None, max_valid: int | None = None, **kwargs*) [source]

Bases: **_WithSubValidators**

Value must pass at least some validations, determined by the given parameter. Optionally, number of passed validations can be capped.

The output of each validator is passed as input to the next.

Parameters:
- **min_valid** – Minimum number of valid schemas.
- **validators** – List of schemas or validators to match input against.
- **max_valid** – Maximum number of valid schemas.
- **msg** – Message to deliver to user if validation fails.
- **kwargs** – All other keyword arguments are passed to the sub-schema constructors.

Raises:
- **NotEnoughValid** – If the minimum number of validations isn't met.
- **TooManyValid** – If the maximum number of validations is exceeded.

```
>>> validate = Schema(SomeOf(min_valid=2, validators=[Range(1, 5), Any(float, int
>>> validate(6.6)
6.6
>>> validate(3)
3
>>> with raises(MultipleInvalid, 'value must be at most 5, not a valid value'):
...     validate(6.2)
```

voluptuous.validators.**Switch**

alias of **Union**

class voluptuous.validators.**Union**(**validators, msg=None, required=False, discriminant=None, **kwargs*) [source]

Bases: **_WithSubValidators**

Use the first validated value among those selected by discriminant.

Parameters:
- **msg** – Message to deliver to user if validation fails.
- **validators)** (*discriminant(value,*) – Returns the filtered list of validators based on the value.
- **kwargs** – All other keyword arguments are passed to the sub-schema constructors.

Returns: Return value of the first validator that passes.

```
>>> validate = Schema(Union({'type':'a', 'a_val':'1'},{'type':'b', 'b_val':'2'},
...                         discriminant=lambda val, alt: filter(
...                         lambda v : v['type'] == val['type'] , alt)))
>>> validate({'type':'a', 'a_val':'1'}) == {'type':'a', 'a_val':'1'}
True
>>> with raises(MultipleInvalid, "not a valid value for dictionary value @ data[
...   validate({'type':'b', 'b_val':'5'})
```

`discriminant({'type':'b', 'a_val':'5'}, [{'type':'a', 'a_val':'1'},{'type':'b', 'b_val':'2'}])` is invoked

Without the discriminant, the exception would be "extra keys not allowed @ data['b_val']"

class voluptuous.validators.**Unique**(*msg: str | None = None*) [source]

Bases: **object**

Ensure an iterable does not contain duplicate items.

Only iterables convertible to a set are supported (native types and objects with correct __eq__).

JSON does not support set, so they need to be presented as arrays. Unique allows ensuring that such array does not contain dupes.

```
>>> s = Schema(Unique())
>>> s([])
[]
>>> s([1, 2])
[1, 2]
>>> with raises(Invalid, 'contains duplicate items: [1]'):
...   s([1, 1, 2])
>>> with raises(Invalid, "contains duplicate items: ['one']"):
...   s(['one', 'two', 'one'])
>>> with raises(Invalid, regex="^contains unhashable elements: "):
...   s([set([1, 2]), set([3, 4])])
>>> s('abc')
'abc'
>>> with raises(Invalid, regex="^contains duplicate items: "):
...   s('aabbc')
```

class voluptuous.validators.**Unordered**(*validators: Iterable[Schema | Object | Mapping | list | tuple | frozenset | set | bool | bytes | int | str | float | complex | type | object | dict | None | Callable], msg: str | None = None, **kwargs*) [source]

Bases: **object**

Ensures sequence contains values in unspecified order.

```
>>> s = Schema(Unordered([2, 1]))
>>> s([2, 1])
[2, 1]
>>> s([1, 2])
[1, 2]
>>> s = Schema(Unordered([str, int]))
>>> s(['foo', 1])
['foo', 1]
>>> s([1, 'foo'])
[1, 'foo']
```

voluptuous.validators.**Url**(*v*) [source]

Verify that the value is a URL.

```
>>> s = Schema(Url())
>>> with raises(MultipleInvalid, 'expected a URL'):
...   s(1)
>>> s('http://w3.org')
'http://w3.org'
```

voluptuous.validators.**truth**(*f: Callable*) → Callable [source]

Convenience decorator to convert truth functions into validators.

```
>>> @truth
... def isdir(v):
...   return os.path.isdir(v)
>>> validate = Schema(isdir)
>>> validate('/')
'/'
>>> with raises(MultipleInvalid, 'not a valid value'):
...   validate('/notavaliddir')
```

Index

A | B | C | D | E | F | H | I | L | M | N | O | P | R | S | T | U | V

A

add() (voluptuous.error.MultipleInvalid method)
All (class in voluptuous.validators)
AllInvalid
And (in module voluptuous.validators)
Any (class in voluptuous.validators)
AnyInvalid

B

Boolean() (in module voluptuous.validators)
BooleanInvalid

C

Capitalize() (in module voluptuous.util)
Clamp (class in voluptuous.validators)
Coerce (class in voluptuous.validators)
CoerceInvalid
Contains (class in voluptuous.validators)
ContainsInvalid

D

Date (class in voluptuous.validators)
DateInvalid
Datetime (class in voluptuous.validators)
DatetimeInvalid
default_factory() (in module voluptuous.schema_builder)
DEFAULT_FORMAT (voluptuous.validators.Date attribute)
(voluptuous.validators.Datetime attribute)
DefaultTo (class in voluptuous.util)
DictInvalid
DirInvalid

E

Email() (in module voluptuous.validators)
EmailInvalid
Equal (class in voluptuous.validators)
Error
error_message (voluptuous.error.Invalid property)
(voluptuous.error.MultipleInvalid property)
ExactSequence (class in voluptuous.validators)
ExactSequenceInvalid
Exclusive (class in voluptuous.schema_builder)
ExclusiveInvalid
extend() (voluptuous.schema_builder.Schema method)
Extra() (in module voluptuous.schema_builder)
extra() (in module voluptuous.schema_builder)

F

FalseInvalid
FileInvalid
FqdnUrl() (in module voluptuous.validators)

H

humanize_error() (in module voluptuous.humanize)

I

In (class in voluptuous.validators)
Inclusive (class in voluptuous.schema_builder)
InclusiveInvalid
infer() (voluptuous.schema_builder.Schema class method)
InInvalid
Invalid
IsDir() (in module voluptuous.validators)
IsFalse() (in module voluptuous.validators)
IsFile() (in module voluptuous.validators)
IsTrue() (in module voluptuous.validators)

L

Length (class in voluptuous.validators)
LengthInvalid
Literal (class in voluptuous.util)
LiteralInvalid
Lower() (in module voluptuous.util)

M

Marker (class in voluptuous.schema_builder)
Match (class in voluptuous.validators)
MatchInvalid
Maybe() (in module voluptuous.validators)
message() (in module voluptuous.schema_builder)
module
- voluptuous
- voluptuous.error
- voluptuous.humanize
- voluptuous.schema_builder
- voluptuous.util
- voluptuous.validators

Msg (class in voluptuous.schema_builder)
msg (voluptuous.error.Invalid property)
(voluptuous.error.MultipleInvalid property)
MultipleInvalid

N

NotEnoughValid
NotIn (class in voluptuous.validators)
NotInInvalid
Number (class in voluptuous.validators)

O

Object (class in voluptuous.schema_builder)
ObjectInvalid
Optional (class in voluptuous.schema_builder)
Or (in module voluptuous.validators)

P

path (voluptuous.error.Invalid property)
(voluptuous.error.MultipleInvalid property)
PathExists() (in module voluptuous.validators)
PathInvalid
prepend() (voluptuous.error.Invalid method)
(voluptuous.error.MultipleInvalid method)

R

raises() (in module voluptuous.schema_builder)
Range (class in voluptuous.validators)
RangeInvalid
Remove (class in voluptuous.schema_builder)
Replace (class in voluptuous.validators)
Required (class in voluptuous.schema_builder)
RequiredFieldInvalid

S

ScalarInvalid
Schema (class in voluptuous.schema_builder)
SchemaError
Self() (in module voluptuous.schema_builder)
SequenceTypeInvalid
Set (class in voluptuous.util)
SetTo (class in voluptuous.util)
SomeOf (class in voluptuous.validators)
Strip() (in module voluptuous.util)
Switch (in module voluptuous.validators)

T

Title() (in module voluptuous.util)
TooManyValid
TrueInvalid
truth() (in module voluptuous.validators)
TypeInvalid

U

Undefined (class in voluptuous.schema_builder)
Union (class in voluptuous.validators)
Unique (class in voluptuous.validators)
Unordered (class in voluptuous.validators)
Upper() (in module voluptuous.util)
Url() (in module voluptuous.validators)
UrlInvalid

V

validate() (in module voluptuous.schema_builder)
validate_with_humanized_errors() (in module voluptuous.humanize)
ValueInvalid
VirtualPathComponent (class in voluptuous.schema_builder)
voluptuous
    module
voluptuous.error
    module
voluptuous.humanize
    module
voluptuous.schema_builder
    module
voluptuous.util
    module
voluptuous.validators
    module

Python Module Index

v

v

voluptuous

voluptuous.error

voluptuous.humanize

voluptuous.schema_builder

voluptuous.util

voluptuous.validators

Search

Searching for multiple words only shows matches that contain all words.

search

```
.. voluptuous documentation master file, created by
   sphinx-quickstart on Tue Jun 13 21:29:23 2023.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to voluptuous's documentation!
======================================

.. image:: https://img.shields.io/pypi/v/voluptuous.svg
   :target: https://python.org/pypi/voluptuous
   :alt: Package on PyPI

.. image:: https://img.shields.io/pypi/l/voluptuous.svg
   :target: https://python.org/pypi/voluptuous
   :alt: BSD

.. image:: https://img.shields.io/pypi/pyversions/voluptuous.svg
   :target: https://python.org/pypi/voluptuous
   :alt: Python Versions

.. image:: https://github.com/alecthomas/voluptuous/actions/workflows/tests.yml/badge.svg
    :target: https://github.com/alecthomas/voluptuous/actions/workflows/tests.yml
    :alt: Test Status


Voluptuous, despite the name, is a Python data validation library. It is primarily intended for
validating data coming into Python as JSON, YAML, etc.

It has three goals:

#. Simplicity.
#. Support for complex data structures.
#. Provide useful error messages.

Content
==================

.. toctree::
   :maxdepth: 2

   modules

Document version:
|version|
|release|

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Welcome to voluptuous's documentation!

pypi v0.15.2 | license BSD-3-Clause | python 3.9 | 3.10 | 3.11 | 3.12 | Tests passing

Voluptuous, despite the name, is a Python data validation library. It is primarily intended for validating data coming into Python as JSON, YAML, etc.

It has three goals:

1. Simplicity.
2. Support for complex data structures.
3. Provide useful error messages.

Content

- voluptuous
  - voluptuous package

Document version: 0.14.2

Indices and tables

- Index
- Module Index
- Search Page

```
voluptuous
==========

.. toctree::
   :maxdepth: 4

   voluptuous
```

```
voluptuous package
==================

.. automodule:: voluptuous
   :members:
   :undoc-members:
   :show-inheritance:

Submodules
----------

voluptuous.error module
-----------------------

.. automodule:: voluptuous.error
   :members:
   :undoc-members:
   :show-inheritance:

voluptuous.humanize module
--------------------------

.. automodule:: voluptuous.humanize
   :members:
   :undoc-members:
   :show-inheritance:

voluptuous.schema\_builder module
---------------------------------

.. automodule:: voluptuous.schema_builder
   :members:
   :undoc-members:
   :show-inheritance:

voluptuous.util module
----------------------

.. automodule:: voluptuous.util
   :members:
   :undoc-members:
   :show-inheritance:

voluptuous.validators module
----------------------------

.. automodule:: voluptuous.validators
   :members:
   :undoc-members:
   :show-inheritance:
```