Natch

Release 1.0.4

Jul 04, 2020

Contents

Pattern matching library.

CHAPTER 1

Guide

1.1 Installation

```
pip install natch
```

1.2 Examples

1.2.1 Example: Authorization

```
import natch


@natch.pattern(
    natch.Any(),
    natch.Partial(
        user=natch.Partial(
            is_authenticated=True,
        ),
    ),
)
def get_user_profile(user_id, request):
    profile = dict()
    return profile


@natch.any()
def get_user_profile(user_id, request):
    raise Exception('Not authenticated.')
```

1.2.2 Example: Factorial

```
import natch


@natch.lt(0)
def factorial(x):
    x = abs(x)
    x = factorial(x)
    x = -1 * x
    return x


@natch.eq(0)
def factorial(x):
    return 1


@natch.eq(1)
def factorial(x):
    return 1


@natch.gt(1)
def factorial(x):
    x = x * factorial(x - 1)
    return x


for i in range(-5, 6):
    result = factorial(i)
    print(i, result)
```

1.2.3 Example: JSON Encoder

```
import natch


@natch.condition(lambda x: isinstance(x, bool))
def encode(data):
    return 'true' if data is True else 'false'


@natch.condition(lambda x: isinstance(x, int))
def encode(data):
    return str(data)


@natch.condition(lambda x: isinstance(x, str))
def encode(data):
    return f'"{data}"'


@natch.condition(lambda x: isinstance(x, list))
def encode(data):
```

(continues on next page)

(continued from previous page)

```
    encoded_segments = list()
    for segment in data:
        encoded_segment = encode(segment)
        encoded_segments.append(encoded_segment)
    inner = ', '.join(encoded_segments)
    return f'[{inner}]'


@natch.condition(lambda x: isinstance(x, dict))
def encode(data):
    inner_split = []
    for key, value in data.items():
        encoded_segment = encode(value)
        inner_split.append(f'"{key}": {encoded_segment}')
    inner = ', '.join(inner_split)
    return f'{{{inner}}}'


data = {'id': -1, 'username': 'ertgl', 'follows': ['neuro-sys'], 'is_active': True}

encoded_data = encode(data)
print(encoded_data)
```

1.2.4 Example: Perfect Square (Guard)

Natch is extendable. This is a how-to guide that shows extending Natch by writing custom rules.

```
import math
import natch


class IsPerfectSquare(natch.Rule):

    def __init__(self, *args, **kwargs):
        super(IsPerfectSquare, self).__init__()

    def does_match(self, *args, **kwargs):
        x = args[0]
        root = math.sqrt(x)
        does_match = int(root + 0.5) ** 2 == x
        return does_match


is_perfect_square = natch.make_rule_decorator(IsPerfectSquare)


@is_perfect_square()
def is_perfect(x):
    """

    Or:
    @natch.all_of(
        natch.Condition(
```

(continues on next page)

(continued from previous page)

```
            lambda x: isinstance(x, int),
        ),
        IsPerfectSquare(),
    )

    Or:
    @natch.pattern(IsPerfectSquare())

    """
    return True


@natch.any()
def is_perfect(x):
    return False


for i in range(10):
    result = is_perfect(i)
    print(i, result)
```

1.2.5 Example: Traversal

```
import natch


class Node(object):

    def __init__(self, **kwargs):
        self.value = kwargs.get('value')
        self.next = kwargs.get('next')

    @natch.partial(
        next=natch.Neq(None),
    )
    def get_next_node(self):
        return self.next

    @natch.partial(
        next=natch.Eq(None),
    )
    def get_next_node(self):
        return None

    def __str__(self):
        return str(self.value)


node = Node(
    value=0,
    next=Node(
        value=1,
        next=Node(
            value=2,
        ),
```

(continues on next page)

(continued from previous page)

```
    ),
)

current_node = node

for i in range(3):
    print(current_node)
    current_node = current_node.get_next_node()
```

1.2.6 Example: Traversal (Functional)

```
import natch


class Node(object):

    def __init__(self, **kwargs):
        self.value = kwargs.get('value')
        self.next = kwargs.get('next')

    def __str__(self):
        return str(self.value)

    @classmethod
    def new(cls, **kwargs):
        node = Node(**kwargs)
        return node

    @classmethod
    @natch.pattern(
        natch.Any(),
        natch.Partial(
            next=natch.Neq(None),
        ),
    )
    def get_next_node(cls, node):
        next_node = node.next
        return next_node

    @classmethod
    @natch.pattern(
        natch.Any(),
        natch.Partial(
            next=natch.Eq(None),
        ),
    )
    def get_next_node(cls, node):
        return None

    @classmethod
    @natch.pattern(
        natch.Any(),
        natch.Eq(None),
    )
    def get_next_node(cls, node):
```

(continues on next page)

(continued from previous page)

```
        return None


node = Node.new(
    value=0,
    next=Node.new(
        value=1,
        next=Node.new(
            value=2,
        ),
    ),
)

current_node = node

for i in range(5):
    print(current_node)
    current_node = Node.get_next_node(current_node)
```

CHAPTER 2

Abstract

class natch.abstract.**Hasher**(**args*, ***kwargs*)

hash(*obj*)

class natch.abstract.**Registry**(**args*, ***kwargs*)

del_hasher()

del_index()

get_hasher()

get_index()

hasher

index

lookup(*func*, **args*, ***kwargs*)

register(*func*, *rule*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

set_hasher(*hasher*)

set_index(*index*)

unregister(*func*, *rule*)

class natch.abstract.**Rule**(**args*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

CHAPTER 3

Core

class natch.core.**Decoration**

classmethod make_rule_decorator(*rule_cls*)

class natch.core.**Registry**(**args*, ***kwargs*)

del_hasher()

del_index()

get_hasher()

get_index()

hasher

index

lookup(*func*, **args*, ***kwargs*)

register(*func*, *rule*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

set_hasher(*hasher*)

set_index(*index*)

unregister(*func*, *rule*)

CHAPTER 4

Decorators

natch.decorators.**any**(**rule_args*, ***rule_kwargs*)

natch.decorators.**any_of**(**rule_args*, ***rule_kwargs*)

natch.decorators.**all_of**(**rule_args*, ***rule_kwargs*)

natch.decorators.**eq**(**rule_args*, ***rule_kwargs*)

natch.decorators.**neq**(**rule_args*, ***rule_kwargs*)

natch.decorators.**gt**(**rule_args*, ***rule_kwargs*)

natch.decorators.**gte**(**rule_args*, ***rule_kwargs*)

natch.decorators.**lt**(**rule_args*, ***rule_kwargs*)

natch.decorators.**lte**(**rule_args*, ***rule_kwargs*)

natch.decorators.**contains**(**rule_args*, ***rule_kwargs*)

natch.decorators.**not_contains**(**rule_args*, ***rule_kwargs*)

natch.decorators.**condition**(**rule_args*, ***rule_kwargs*)

natch.decorators.**partial**(**rule_args*, ***rule_kwargs*)

natch.decorators.**pattern**(**rule_args*, ***rule_kwargs*)

CHAPTER 5

Exceptions

exception natch.exceptions.**NeverMatchesError**(**args*, ***kwargs*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

CHAPTER 6

Hashers

class natch.hashers.**QualnameHasher**(**args*, ***kwargs*)

hash(*obj*)

CHAPTER 7

Rules

class natch.rules.**Any**(**args*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**AnyOf**(**args*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**AllOf**(**args*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Eq**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Neq**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Gt**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Gte**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Lt**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Lte**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

**kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Contains**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**NotContains**(*value*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Condition**(*func*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Partial**(**args*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

class natch.rules.**Pattern**(**rules*, ***kwargs*)

args

del_args()

del_kwargs()

does_match(**args*, ***kwargs*)

get_args()

get_kwargs()

kwargs

set_args(*args*)

set_kwargs(*kwargs*)

- modindex

Python Module Index

n

Index

R

S

U

W