reiter

Release 0.8.0

Andrei Lapets

May 27, 2023

CONTENTS

Wrapper for Python iterators and iterables that implements a list-like random-access interface by caching retrieved items for later reuse.

CHAPTER

ONE

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install reiter
```

The library can be imported in the usual way:

```
import reiter
from reiter import reiter
```

1.1 Examples

This library makes it possible to wrap any iterator or iterable object within an interface that enables repeated iteration over – and random access by index of – the items contained within that object. A `reiter` instance yields the same sequence of items as the wrapped iterator or iterable:

```
>>> from reiter import reiter
>>> xs = iter([1, 2, 3])
>>> ys = reiter(xs)
>>> list(ys)
[1, 2, 3]
```

Unlike iterators and some iterable objects (including those that are built-in and those that are user-defined), an instance of the `reiter` class *always* allows iteration over its items any number of times. More specifically, every invocation of `iter` (explicit or implicit) returns an iterator that begins iteration from the first item found in the originally wrapped iterator or iterable:

```
>>> list(iter(ys)), list(iter(ys))
([1, 2, 3], [1, 2, 3])
>>> list(ys), list(ys)
([1, 2, 3], [1, 2, 3])
```

Furthermore, it is also possible to access elements by their index:

```
>>> xs = iter([1, 2, 3])
>>> ys = reiter(xs)
>>> ys[0], ys[1], ys[2]
(1, 2, 3)
```

The built-in Python `next` function is also supported, and any attempt to retrieve an item once the sequence of items is exhausted raises the `StopIteration` exception in the usual manner:

```
>>> xs = reiter(iter([1, 2, 3]))
>>> next(xs), next(xs), next(xs)
(1, 2, 3)
>>> next(xs)
Traceback (most recent call last):
  ...
StopIteration
```

However, all items yielded during iteration can be accessed by their index, and it is also possible to iterate over those items again:

```
>>> xs[0], xs[1], xs[2]
(1, 2, 3)
>>> [x for x in xs]
[1, 2, 3]
```

Retrieval of yielded items using slice notation is also supported via the `__getitem__` method:

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[0:2]
[1, 2]
```

Instances of `reiter` support additional inspection methods, as well. For example, the `has` method returns a boolean value indicating whether a next item is available and the `length` method returns the length of the sequence of items emitted by the instance (once no more items can be emitted):

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs.has(), xs.has(), xs.has(), xs.has()
(True, True, True, False)
>>> xs.length()
3
```

CHAPTER

TWO

DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

2.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

2.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Alternatively, all unit tests are included in the module itself and can be executed using doctest:

```
python src/reiter/reiter.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/reiter
```

2.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

2.4 Versioning

The version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

2.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

2.5.1 reiter module

Wrapper for Python iterators and iterables that implements a list-like random-access interface by caching retrieved items for later reuse.

class reiter.reiter.**reiter**(*iterable: Iterable*)

Bases: `collections.abc.Iterator`, `collections.abc.Iterable`

Wrapper class for iterators and iterables that provides an interface enabling repeated iteration and random access by index of the sequence of items contained within.

static __new__(*cls*, *iterable: Iterable*)

Constructor that wraps an iterator or iterable. An instance of this class yields the same sequence of items as the wrapped object.

```
>>> xs = iter([1, 2, 3])
>>> ys = reiter(xs)
>>> list(ys)
[1, 2, 3]
```

Unlike iterators and some iterable objects (including those that are built-in and those that are user-defined), an instance of this class *always* allows iteration over its items any number of times.

```
>>> list(ys), list(ys)
([1, 2, 3], [1, 2, 3])
```

Furthermore, it is also possible to access elements using their index.

```
>>> xs = iter([1, 2, 3])
>>> ys = reiter(xs)
>>> ys[0], ys[1], ys[2]
(1, 2, 3)
```

An instance of this class can be constructed from another instance of this class.

```
>>> list(reiter(reiter(iter([1, 2, 3]))))
[1, 2, 3]
```

The type of an instance of this class can be checked in the usual manner, and an instance of this class cannot be constructed from a value or object that is not an iterator or iterable.

```
>>> isinstance(reiter(xs), reiter)
True
>>> reiter(123)
Traceback (most recent call last):
  ...
TypeError: supplied object is not iterable
```

__next__() → Any

Substitute definition of the corresponding method for iterators that also caches the retrieved item before returning it.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> next(xs), next(xs), next(xs)
(1, 2, 3)
```

Any attempt to retrieve items once the sequence of items is exhausted raises an exception in the usual manner.

```
>>> next(xs)
Traceback (most recent call last):
  ...
StopIteration
```

However, all items yielded during iteration can be accessed by their index, and it is also possible to iterate over them again.

```
>>> xs[0], xs[1], xs[2]
(1, 2, 3)
```

(continues on next page)

(continued from previous page)

```
>>> [x for x in xs]
[1, 2, 3]
>>> [x for x in xs], [x for x in xs]
([1, 2, 3], [1, 2, 3])
```

__getitem__(*index: Union[int, slice]*) → Any

Returns the item at the supplied index or the items within the range of the supplied slice, retrieving additional items from the iterator (and caching them) as necessary.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[2]
3
>>> xs[1]
2
>>> xs = reiter(range(10))
>>> xs[0]
0
>>> xs = reiter(range(10))
>>> xs[10]
Traceback (most recent call last):
  ...
IndexError: index out of range
>>> xs['abc']
Traceback (most recent call last):
  ...
ValueError: index must be integer or slice
```

Use of slice notation is supported, but it should be used carefully. Omitting a lower or upper bound may require retrieving (and caching) all items.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[0:2]
[1, 2]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[:2]
[1, 2]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[0:]
[1, 2, 3]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[:]
[1, 2, 3]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[2:0:-1]
[3, 2]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[2::-1]
[3, 2, 1]
>>> xs = reiter(iter([1, 2, 3]))
>>> xs[::-1]
[3, 2, 1]
>>> xs = reiter(iter([1, 2, 3]))
```

(continues on next page)

(continued from previous page)

```
>>> xs[:0:-1]
[3, 2]
```

__iter__() → Iterable

Builds a new iterator that begins at the first cached element and continues from there. This method is an effective way to "reset" the instance of this class so that the built-in `next` function can be used again.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> next(xs)
1
>>> next(xs)
2
>>> next(xs)
3
>>> next(xs)
Traceback (most recent call last):
  ...
StopIteration
>>> xs = iter(xs)
>>> next(xs), next(xs), next(xs)
(1, 2, 3)
```

has(*index: Optional[int] = None*) → bool

Returns a boolean indicating whether a next item is available, or if an item exists at the specified index.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs.has(), xs.has(), xs.has(), xs.has()
(True, True, True, False)
```

If an explicit index is supplied, a boolean value is returned indicating whether an item exists at that position in the sequence within the wrapped iterator or iterable.

```
>>> xs.has(2)
True
>>> xs = reiter(iter([1, 2, 3]))
>>> xs.has(2)
True
>>> xs.has(3)
False
```

length() → Optional[int]

Returns the length of this instance, if *all* items have been retrieved. If not all items have been retrieved, `None` is returned.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs.length() is None
True
>>> next(xs)
1
>>> xs.length() is None
True
>>> next(xs), next(xs)
(2, 3)
```

(continues on next page)

(continued from previous page)

```
>>> next(xs)
Traceback (most recent call last):
  ...
StopIteration
>>> xs.length()
3
```

Invoking the *has* method until the instance is exhausted is sufficient to ensure that all items have been retrieved.

```
>>> xs = reiter(iter([1, 2, 3]))
>>> xs.has(), xs.has(), xs.has(), xs.has()
(True, True, True, False)
>>> xs.length()
3
```

PYTHON MODULE INDEX

r

INDEX

Symbols

H

L

M

R