amzlist Documentation

Release 0.1

John Keyes

April 26, 2012

CONTENTS

A linked list implementation by John Keyes.

CHAPTER

ONE

AMZLIST

1.1 amzlist Package

1.1.1 amzlist Package

class amzlist.__init__.**LinkedList**(*strict=None*)
Bases: object

A LinkedList implementation.

append(*node*)
Inserts *node* at the tail of the LinkedList.

as_list()
Returns this LinkedList as a *list* of Nodes.

data
Returns the *data* for the first Node.

find(*node*, *inc_prev=None*)
Find the specified Node.

If the *node* parameter is a Node, and it has *data* and a *next* Node then the first Node with encountered that has the same *data* and *next* attribute values will match.

If the *node* parameter is a value other than a Node or a Node with just a *data* attribute value, then the first node encountered with the same *data* attribute value will match.

If *inc_prev* is *True*, this method returns the node and it's previous node in a tuple, otherwise it returns the node.

This method returns *None* if the node cannot be found.

insert(*node*, *after*)
Inserts *node* and makes *after.next* refer to it.

last_node
Returns the last Node.

next
Returns the *next* Node.

pop()
Returns the Node from the head, and removes it.

prepend(*node*)
Inserts *node* at the head of the LinkedList.

push(*node*)
Prepends a Node to the head.

remove(*node*)
Remove the specified *node*.

If the *node* parameter is a Node, and it has *data* and a *next* Node then the first Node with encountered that has the same *data* and *next* attribute values will be removed.

If the *node* parameter is a value other than a Node or a Node with just a *data* attribute value, then the first node encountered with the same *data* attribute is removed.

reverse_iterative()
Returns a new LinkedList with the Nodes in reverse order.

This method uses an iterative approach.

reverse_recursive(*node=None*, *new_list=None*)
Returns a new LinkedList with the Nodes in reverse order.

This method uses a recursive approach.

class amzlist.__init__.**Node**(*data*)
Bases: object

A Node is a simple object with two attributes, *next* and *data*.

data stores a value, and *next* holds a reference to another Node.

data = None
Data text

next = None
Next text

strict = False

CHAPTER

TWO

GETTING THE CODE

The simplest way is to clone the repository from GitHub:

```
git clone https://github.com/jkeyes/amzlist.git
```

Or you can download the repository in a ZIP file.

CHAPTER

THREE

BASIC USAGE

You can prepend items to a linked list, which adds each new Node as the head of the list:

```
from amzlist import LinkedList

lnkd_list = LinkedList()
lnkd_list.prepend("amazon.com")
lnkd_list.prepend("@")
lnkd_list.prepend("jkeyes")
```

You can also append. Note this is much slower as we must traverse the entire list to find where to insert the node:

```
lnkd_list.append("Why would you use append?")
```

It's also possible to insert a node after another node:

```
from amzlist import Node
n_color = Node("Red")
n_answer = Node(42)

lnkd_list.prepend(n_color)
lnkd_list.insert(n_answer, n_color) # insert n_answer after n_color
```

To remove a node the *remove* method can be used:

```
lnkd_list.remove("Red")
```

Alternatively you can use *push* and *pop* to add and remove nodes from the list:

```
lnkd_list.push("Item 1")
lnkd_list.push("Item 2")
node = lnkd_list.pop()
node.data == "Item 2"
```

CHAPTER

FOUR

REVERSING

There are two methods to reverse the list, one uses an iterative approach and the other a recursive one:

```
rvsd_list = lnkd_list.reverse_iterative()
rvsd_list = lnkd_list.reverse_recursive()
```

CHAPTER

FIVE

CYCLE DETECTION

If the *LinkedList* methods are used no cycles can be introduced. However, it is possible to introduct a cycle by directly manipulating the nodes:

```
lnkd_list = LinkedList()
node_john = Node('John')
lnkd_list.prepend(node_john)
node_james = Node('James')
lnkd_list.prepend(node_james)
node_joe = Node('Joe')
lnkd_list.prepend(node_joe)

# introduce a cycle
# Joe->James->John->Jules->John
node_jules = Node('Jules')
node_john.next = node_jules
node_jules.next = node_john
```

To prevent this you can create a *strict* LinkedList.

```
lnkd_list = LinkedList(strict=True)
...
node_jules.next = node_john
# Raises ValueError
```

WARNING: this is an EXTREMELY costly feature, as it requires traversal of the list for each Node that is added to the List (if the node being added has a value for *next*).

CHAPTER

SIX

RUNNING THE TESTS

To run the testsuite you'll need to setup the environment first:

```
cd amzlist
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
```

Then you can run the testsuite:

```
nosetests tests
```

You can also get a coverage report:

```
nosetests tests --with-coverage --cover-package amzlist
```

CHAPTER

SEVEN

CONTINUOUS INTEGRATION

The testsuite has been run on Python 2.5, 2.6, 2.7 and 3.2 on Travis CI.

CHAPTER

EIGHT

API DOCUMENTATION

Browse the API Documentation.

If you want to generate the documentation use the following commands:

```
cd docs
make html # docs will be generated in _build/html
```

PYTHON MODULE INDEX

a