pyLINQ Documentation

Release 1.0.0

Joshua M. Fernandes

Sep 18, 2019

Contents

This library extends regular python lists to make them easier to filter and sort. Typical list comprehensions such as [x for x in list] can quickly grow out of control for complicated behaviors. pyLINQ solves this by replacing all features of list comprehensions with a series of methods that act on a list.

Here is how easy it is to use:

```
>>> my_list = List([1,-9,5,2,1])
>>> my_list.select("abs(x)").where("x>3") # Get elements where the absolute value is
↪greater than 3.
[9,5]
>>> my_list.first("x % 2 == 0") # Get the first even number
2
>>> nameList = List(['harry','tina','jeff','hank','john','tom','steve'])
>>> nameList.groupby("x[0]") # Group list elements by first letter in name.
[{'h': ['harry', 'hank']}, {'t': ['tina', 'tom']}, {'j': ['jeff', 'john']}, {'s': [
↪'steve']}]
>>> nameList.orderby("x[0]").takewhile("x[0] < 'm'") # Get all names that come before
↪letter 'm'.
['harry', 'hank', 'jeff', 'john']
```

List of Methods for List Object		
Method Name	Description	Output
any	Returns true if any elements matches expression	bool
all	Returns true if all elements matches expression	bool
concat	Combines two lists into a single list	List
count	Returns the number of elements that match expression	int
distinct	Returns all elements that are not duplicates	List
duplicate	Returns all elements that are duplicates	List
element	Returns the element at a specific index	element
except_set	Returns all elements except if they are in a second list	List
first	Returns the first element that matches an expression	element
groupby	Groups all value by a certain key	List
intersect	Returns all elements common between two lists	List
last	Returns the last element that matches an expression	element
max	Returns the max element of the list	element
min	Returns the min element of the list	element
oftype	Returns all elements that match a certain type	List
orderby	Sort the elements	List
select	Perform an operation/function on list elements	List
skipwhile	Return all elements after an expression is true	List
sum	Return the sum of all elements	float
takewhile	Return all elements before an expression is false	List
union	Return the combination of two lists	List
where	Return all elements that meet an expression	List

CHAPTER 1

Example Usage

```
>>> int_list = List([2,4,6,5])
>>> other_int_list = List([1,2,2,3,4,4,5])
>>> # Sum examples
>>> other_int_list.sum("x > 3") # Sum using conditional statement x > 3
(4 + 4 + 5) = 13
>>> other_int_list.sum("x - 2") # Sum using operation x - 2
(-1 + 0 + 0 + 1 + 2 + 2 + 3) = 7
>>> # chaining expressions example
>>> other_int_list.where("x % 2 != 0").select("x**2").max() # finding the maximum of␣
↪the square of the odds
max(1,9,25) = 25
>>> # counting
>>> int_list.concat(other_int_list).count("x % 2 == 0") # counting the number of␣
↪evens in both lists
count(2,4,6,2,2,4,4) = 7
```

CHAPTER 2

List Object

class List(*data=None*)

Object which extends python list object to include LINQ style queries.

all(*expression*)

Checks to see if all items in a list meet a conditional criteria. E.g. List.all("x > 0").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object, which must be a conditional statement. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a bool that is true if all elements satisfy the applied expression.

any(*expression*)

Checks to see if any items in a list meet a conditional criteria. E.g. List.any("x > 0").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object, which must be a conditional statement. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a bool that is true if any elements satisfy the applied expression.

append(*val*)

S.append(value) – append value to the end of the sequence

clear() → None – remove all items from S

concat(*other*)

Adds all the elements of another List object onto the calling List object. E.g. List1.concat(List2)

Parameters **other** (List) – Another List instance to concatinate with the calling List.

Returns a new List object that is the concatination of self and other.

count(*expression=None*)

Counts the number of items in a list that meet a conditional criteria. E.g. List.count("x > 0").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an integer representing the number of elements that satisfy the applied expression.

distinct(*expression=None*)
Checks to see if any items in a list meet a conditional criteria, and also removes duplicates. E.g. List.distinct("x > 0").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a new List object that results from the applied expression.

duplicate(*expression=None*)
Checks to see if any items in a list meet a conditional criteria, but only if they are duplicate items. E.g. List.duplicate("x > 0").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a new List object that results from the applied expression.

element(*number*)
Returns the item at a specified index. E.g. List.element(0)

Parameters **number** (*int*) – The index to get the item from.

Returns an item from the list or None if index is out of range.

except_set(*other*)
Removes all instances from a list that exist in another list exept for duplicates. E.g. List1.except_set(List2)

Parameters **other** (List) – Another List instance to compare entries with the calling List.

Returns a new List object that contains all items in self, except any item that exist in other.

first(*expression=None*)
Finds the first item in a list that meets a conditional criteria. E.g. List.first("x > 0").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an item from the list or None if nothing is found.

groupby(*expression*)
Uses a parameter of that data to group all entries according to key values. E.g. List.groupby("x['key']").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object. Can either take a string ("x[0]") or a lambda expression (lambda x : x[0]).

Returns a new List object that results from the applied expression.

insert(*ii*, *val*)
S.insert(index, value) – insert value before index

intersect(*other*)
Returns all entries that are common between two lists without duplicates. E.g. List1.intersect(List2)

Parameters **other** (List) – Another List instance to compare entries with the calling List.

Returns a new List object that contains all items that exist in both self and other.

last (*expression=None*)
Finds the last item in a list that meets a conditional criteria. E.g. List.last("x > 0").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an item from the list or None if nothing is found.

max (*expression=None*)
Finds the maximum value item in a list. Optionally, the item can be applied to a conditional criteria. E.g. List.max("x > 0") or List.max("x**2").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object. Can also be an operation or function to apply to elements. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an item from the list or None if nothing is found.

min (*expression=None*)
Finds the minimum value item in a list. Optionally, the item can be applied to a conditional criteria. E.g. List.min("x > 0") or List.min("x**2").

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object. Can also be an operation or function to apply to elements. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an item from the list or None if nothing is found.

oftype (*elementType*)
Checks to see if any items in a list are of a certain type. E.g. List.oftype(int)

Parameters **elementType** (*str or type*) – The expression to apply to the List object. Can either take a string "int","dict","List", etc. or a type.

Returns a new List object that contains only items of the specified type.

orderby (*expression=None*, *reverse=False*)
Orders the items in a list by a certain value. Optionally, the item can be applied to a conditional criteria. E.g. List.orderby("x > 0") or List.orderby("len(x)").

Parameters

- **expression** (*str or LambdaType or None*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).
- **reverse** (*bool*) – True if the list should be descending. False if the list should be ascending.

Returns a new List object that results from the applied expression.

pop ([*index*]) → item – remove and return item at index (default last).
Raise IndexError if list is empty or index is out of range.

remove (*val*)
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

select(*expression*)
Applies an operator or function to each item in the list. E.g. List.select("len(x)").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object. Can either take a string ("x**2") or a lambda expression (lambda x : x**2).

Returns a new List object that results from the applied expression.

skipwhile(*expression*)
Returns all elements after the first item in the list that fails to meet a conditional criteria. E.g. List.skipwhile("x > 0").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object, which must be a conditional statement. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a new List object that results from the applied expression.

sum(*expression=None*)
Finds the sum of all values in a list. Optionally, the item can be applied to a conditional criteria. E.g. List.sum("x > 0") or List.sum("x**2")

Parameters **expression** (*str or LambdaType or None*) – The expression to apply to the List object. Can also be an operation or function to apply to elements. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns an item from the list or None if nothing is found.

takewhile(*expression*)
Returns all elements before the first item in the list that fails to meet a conditional criteria. E.g. List.takewhile("x > 0").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object, which must be a conditional statement. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a new List object that results from the applied expression.

union(*other*)
Returns all entries for two Lists except for any duplicates. E.g. List1.union(List2)

Parameters **other** (List) – Another List instance to compare entries with the calling List.

Returns a new List object that contains all items in self and other, except any duplicates.

where(*expression*)
Checks to see if any items in a list meet a conditional criteria. E.g. List.where("x > 0").

Parameters **expression** (*str or LambdaType*) – The expression to apply to the List object, which must be a conditional statement. If no expression is supplied, then function acts on the whole list. Can either take a string ("x > 0") or a lambda expression (lambda x : x > 0).

Returns a new List object that results from the applied expression.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index