tributary Documentation

Release 0.1.4

Tim Paine

Sep 06, 2020

Contents

Python Data Streams

Build Status GitHub issues Coverage PyPI PyPI Docs

Tributary is a library for constructing dataflow graphs in python. Unlike many other DAG libraries in python (airflow, luigi, prefect, dagster, dask, kedro, etc), tributary is not designed with data/etl pipelines or scheduling in mind. Instead, tributary is more similar to libraries like mdf, pyungo, streamz, or pyfunctional, in that it is designed to be used as the implementation for a data model. One such example is the greeks library, which leverages tributary to build data models for options pricing.

CHAPTER 1

Installation

Install with pip:

```
pip install tributary
```

or with conda:

```
conda install -c conda-forge tributary
```

or from source:

```
python setup.py install
```

Note: If installing from source or with pip, you'll also need Graphviz itself if you want to visualize the graph using the `.graphviz()` method.

CHAPTER 2

Stream Types

Tributary offers several kinds of streams:

2.1 Streaming

These are synchronous, reactive data streams, built using asynchronous python generators. They are designed to mimic complex event processors in terms of event ordering.

2.2 Functional

These are functional streams, built by currying python functions (callbacks).

2.3 Lazy

These are lazily-evaluated python streams, where outputs are propogated only as inputs change. They are implemented as directed acyclic graphs.

CHAPTER 3

Examples

- Streaming: In this example, we construct a variety of forward propogating reactive graphs.
- Lazy: In this example, we construct a variety of lazily-evaluated directed acyclic computation graphs.
- Automatic Differentiation: In this example, we use `tributary` to perform automatic differentiation on both lazy and streaming graphs.

CHAPTER 4

Graph Visualization

You can visualize the graph with Graphviz. All streaming and lazy nodes support a `graphviz` method.

Streaming and lazy nodes also support ipydagred3 for live update monitoring.

4.1 Streaming

Here green indicates executing, yellow indicates stalled for backpressure, and red indicates that `StreamEnd` has been propogated (e.g. stream has ended).

4.2 Lazy

Here green indicates executing, and red indicates that the node is dirty. Note the the determination if a node is dirty is also done lazily (we can check with `isDirty` whcih will update the node's graph state.

CHAPTER 5

Sources and Sinks

5.1 Sources

- Python Function/Generator/Async Function/Async Generator
- Random - generates a random dictionary of values
- File - streams data from a file, optionally loading each line as a json
- Kafka - streams data from kafka
- Websocket - strams data from a websocket
- Http - polls a url with GET requests, streams data out
- SocketIO - streams data from a socketIO connection

5.2 Sinks

- File - data to a file
- Kafka - streams data to kafka
- Http - POSTs data to an url
- Websocket - streams data to a websocket
- SocketIO - streams data to a socketIO connection

CHAPTER 6

Transforms

6.1 Modulate

- Delay - Streaming wrapper to delay a stream
- Apply - Streaming wrapper to apply a function to an input stream
- Window - Streaming wrapper to collect a window of values
- Unroll - Streaming wrapper to unroll an iterable stream
- UnrollDataFrame - Streaming wrapper to unroll a dataframe into a stream
- Merge - Streaming wrapper to merge 2 inputs into a single output
- ListMerge - Streaming wrapper to merge 2 input lists into a single output list
- DictMerge - Streaming wrapper to merge 2 input dicts into a single output dict. Preference is given to the second input (e.g. if keys overlap)
- Reduce - Streaming wrapper to merge any number of inputs

6.2 Calculations

6.2.1 Arithmetic Operators

- Noop (unary) - Pass input to output
- Negate (unary) - -1 * input
- Invert (unary) - 1/input
- Add (binary) - add 2 inputs
- Sub (binary) - subtract second input from first
- Mult (binary) - multiple inputs

- Div (binary) - divide first input by second
- RDiv (binary) - divide second input by first
- Mod (binary) - first input % second input
- Pow (binary) - first input^second input
- Sum (n-ary) - sum all inputs
- Average (n-ary) - average of all inputs

6.2.2 Boolean Operators

- Not (unary) - `Not` input
- And (binary) - `And` inputs
- Or (binary) - `Or` inputs

6.2.3 Comparators

- Equal (binary) - inputs are equal
- NotEqual (binary) - inputs are not equal
- Less (binary) - first input is less than second input
- LessOrEqual (binary) - first input is less than or equal to second input
- Greater (binary) - first input is greater than second input
- GreaterOrEqual (binary) - first input is greater than or equal to second input

6.2.4 Math

- Log (unary)
- Sin (unary)
- Cos (unary)
- Tan (unary)
- Arcsin (unary)
- Arccos (unary)
- Arctan (unary)
- Sqrt (unary)
- Abs (unary)
- Exp (unary)
- Erf (unary)

6.3 Converters

- Int (unary)
- Float (unary)
- Bool (unary)
- Str (unary)

6.4 Python Builtins

- Len (unary)

6.5 Rolling

- RollingCount - Node to count inputs
- RollingMin - Node to take rolling min of inputs
- RollingMax - Node to take rolling max of inputs
- RollingSum - Node to take rolling sum inputs
- RollingAverage - Node to take the running average
- SMA - Node to take the simple moving average over a window
- EMA - Node to take an exponential moving average over a window

6.6 Node Type Converters

- Lazy->Streaming

CHAPTER 7

API Documentation

CHAPTER 8

API

8.1 Streaming

tributary.streaming.**run**(*node*)

class tributary.streaming.base.**StreamingGraph**(*output_node*)

Bases: object

dagre()

graph()

graphviz()

run()

tributary.streaming.utils.**Apply**(*node*, *foo*, *foo_kwargs=None*)

Streaming wrapper to apply a function to an input stream

Parameters

- **node** (*node*) – input stream
- **foo** (*callable*) – function to apply
- **foo_kwargs** (*dict*) – kwargs for function

tributary.streaming.utils.**Delay**(*node*, *delay=1*)

Streaming wrapper to delay a stream

Parameters

- **node** (*node*) – input stream
- **delay** (*float*) – time to delay input stream

tributary.streaming.utils.**DictMerge**(*node1*, *node2*)

Streaming wrapper to merge 2 input dicts into a single output dict. Preference is given to the second input (e.g. if keys overlap)

Parameters

- **node1** (*node*) – input stream
- **node2** (*node*) – input stream

tributary.streaming.utils.**FixedMap**(*node*, *count*, *mapper=None*)
Streaming wrapper to split stream into a fixed number of outputs

Parameters

- **node** (*Node*) – input stream
- **count** (*int*) – number of output nodes to generate
- **mapper** (*function*) – how to map the inputs into *count* streams

tributary.streaming.utils.**ListMerge**(*node1*, *node2*)
Streaming wrapper to merge 2 input lists into a single output list

Parameters

- **node1** (*node*) – input stream
- **node2** (*node*) – input stream

tributary.streaming.utils.**Merge**(*node1*, *node2*)
Streaming wrapper to merge 2 inputs into a single output

Parameters

- **node1** (*node*) – input stream
- **node2** (*node*) – input stream

tributary.streaming.utils.**Reduce**(**nodes*, *reducer=None*)
Streaming wrapper to merge any number of inputs

Parameters

- **nodes** (*tuple*) – input streams
- **reducer** (*function*) – how to map the outputs into one stream

tributary.streaming.utils.**Unroll**(*node*)
Streaming wrapper to unroll an iterable stream

Parameters **node** (*node*) – input stream

tributary.streaming.utils.**UnrollDataFrame**(*node*, *json=False*, *wrap=False*)
Streaming wrapper to unroll a dataframe into a stream

Parameters **node** (*node*) – input stream

tributary.streaming.utils.**Window**(*node*, *size=-1*, *full_only=False*)
Streaming wrapper to collect a window of values

Parameters

- **node** (*node*) – input stream
- **size** (*int*) – size of windows to use
- **full_only** (*bool*) – only return if list is full

8.1.1 Input

class tributary.streaming.input.file.**File**(*filename*, *json=True*)
Bases: tributary.streaming.input.input.Foo

Open up a file and yield back lines in the file

Parameters

- **filename** (*str*) – filename to read
- **json** (*bool*) – load file line as json

class tributary.streaming.input.http.**HTTP**(*url*, *interval=1*, *repeat=1*, *json=False*, *wrap=False*, *field=None*, *proxies=None*, *cookies=None*)
Bases: tributary.streaming.input.input.Foo

Connect to url and yield results

Parameters

- **url** (*str*) – url to connect to
- **interval** (*int*) – interval to re-query
- **repeat** (*int*) – number of times to request
- **json** (*bool*) – load http content data as json
- **wrap** (*bool*) – wrap result in a list
- **field** (*str*) – field to index result by
- **proxies** (*list*) – list of URL proxies to pass to requests.get
- **cookies** (*list*) – list of cookies to pass to requests.get

class tributary.streaming.input.kafka.**Kafka**(*servers*, *group*, *topics*, *json=False*, *wrap=False*, *interval=1*, ***consumer_kwargs*)
Bases: tributary.streaming.input.input.Foo

Connect to kafka server and yield back results

Parameters

- **servers** (*list*) – kafka bootstrap servers
- **group** (*str*) – kafka group id
- **topics** (*list*) – list of kafka topics to connect to
- **json** (*bool*) – load input data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – kafka poll interval

class tributary.streaming.input.socketio.**SocketIO**(*url*, *channel=''*, *field=''*, *sendinit=None*, *json=False*, *wrap=False*, *interval=1*)
Bases: tributary.streaming.input.input.Foo

Connect to socketIO server and yield back results

Parameters

- **url** (*str*) – url to connect to

- **channel** (*str*) – socketio channel to connect through
- **field** (*str*) – field to index result by
- **sendinit** (*list*) – data to send on socketio connection open
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – socketio wai interval

class tributary.streaming.input.ws.**WebSocket**(*url*, *json=False*, *wrap=False*)

Bases: tributary.streaming.input.input.Foo

Connect to websocket and yield back results

Parameters

- **url** (*str*) – websocket url to connect to
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list

8.1.2 Output

tributary.streaming.output.http.**HTTP**(*node*, *url=''*, *json=False*, *wrap=False*, *field=None*, *proxies=None*, *cookies=None*)

Connect to url and post results to it

Parameters

- **node** (*Node*) – input tributary
- **url** (*str*) – url to post to
- **json** (*bool*) – dump data as json
- **wrap** (*bool*) – wrap input in a list
- **field** (*str*) – field to index result by
- **proxies** (*list*) – list of URL proxies to pass to requests.post
- **cookies** (*list*) – list of cookies to pass to requests.post

tributary.streaming.output.kafka.**Kafka**(*node*, *servers=''*, *topic=''*, *json=False*, *wrap=False*, ***producer_kwargs*)

Connect to kafka server and send data

Parameters

- **node** (*Node*) – input tributary
- **servers** (*list*) – kafka bootstrap servers
- **topic** (*str*) – kafka topic to connect to
- **json** (*bool*) – load input data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – kafka poll interval

tributary.streaming.output.socketio.**SocketIO**(*node*, *url*, *channel=''*, *field=''*, *sendinit=None*, *json=False*, *wrap=False*, *interval=1*)

Connect to socketIO server and send updates

Parameters

- **node** (*Node*) – input stream
- **url** (*str*) – url to connect to
- **channel** (*str*) – socketio channel to connect through
- **field** (*str*) – field to index result by
- **sendinit** (*list*) – data to send on socketio connection open
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – socketio wai interval

tributary.streaming.output.ws.**WebSocket**(*node*, *url=''*, *json=False*, *wrap=False*, *field=None*, *response=False*, *response_timeout=1*)

Connect to websocket and send data

Parameters

- **node** (*Node*) – input tributary
- **url** (*str*) – websocket url to connect to
- **json** (*bool*) – dump data as json
- **wrap** (*bool*) – wrap result in a list

8.1.3 Calculations

tributary.streaming.calculations.ops.**Abs**(*self*)

tributary.streaming.calculations.ops.**Add**(*self*, *other*)

tributary.streaming.calculations.ops.**And**(*self*, *other*)

tributary.streaming.calculations.ops.**Arccos**(*self*)

tributary.streaming.calculations.ops.**Arcsin**(*self*)

tributary.streaming.calculations.ops.**Arctan**(*self*)

tributary.streaming.calculations.ops.**Average**(**others*)

tributary.streaming.calculations.ops.**Bool**(*self*)

tributary.streaming.calculations.ops.**Ceil**(*self*)

tributary.streaming.calculations.ops.**Cos**(*self*)

tributary.streaming.calculations.ops.**Div**(*self*, *other*)

tributary.streaming.calculations.ops.**Equal**(*self*, *other*)

tributary.streaming.calculations.ops.**Erf**(*self*)

tributary.streaming.calculations.ops.**Exp**(*self*)

tributary.streaming.calculations.ops.**Float**(*self*)

tributary.streaming.calculations.ops.**Floor**(*self*)

tributary.streaming.calculations.ops.**Ge**(*self*, *other*)

tributary.streaming.calculations.ops.**Gt**(*self*, *other*)

tributary.streaming.calculations.ops.**Int**(*self*)

tributary.streaming.calculations.ops.**Invert**(*self*)

tributary.streaming.calculations.ops.**Le**(*self*, *other*)

tributary.streaming.calculations.ops.**Len**(*self*)

tributary.streaming.calculations.ops.**Log**(*self*)

tributary.streaming.calculations.ops.**Lt**(*self*, *other*)

tributary.streaming.calculations.ops.**Mod**(*self*, *other*)

tributary.streaming.calculations.ops.**Mult**(*self*, *other*)

tributary.streaming.calculations.ops.**Negate**(*self*)

tributary.streaming.calculations.ops.**Noop**(*self*)

tributary.streaming.calculations.ops.**Not**(*self*)

tributary.streaming.calculations.ops.**NotEqual**(*self*, *other*)

tributary.streaming.calculations.ops.**Or**(*self*, *other*)

tributary.streaming.calculations.ops.**Pow**(*self*, *other*)

tributary.streaming.calculations.ops.**RDiv**(*self*, *other*)

tributary.streaming.calculations.ops.**Round**(*self*, *ndigits=0*)

tributary.streaming.calculations.ops.**Sin**(*self*)

tributary.streaming.calculations.ops.**Sqrt**(*self*)

tributary.streaming.calculations.ops.**Str**(*self*)

tributary.streaming.calculations.ops.**Sub**(*self*, *other*)

tributary.streaming.calculations.ops.**Sum**(**others*)

tributary.streaming.calculations.ops.**Tan**(*self*)

tributary.streaming.calculations.ops.**binary**(*foos*, *name*)

tributary.streaming.calculations.ops.**n_ary**(*foos*, *name*)

tributary.streaming.calculations.ops.**unary**(*foos*, *name*)

tributary.streaming.calculations.rolling.**Average**(*node*)

Node to take the running average

If stream type is iterable, will do (average + sum(input))/(count+len(input)). If input stream type is not iterable, will do (average + input)/count

tributary.streaming.calculations.rolling.**Count**(*node*)

Node to count inputs

Parameters **node** (*Node*) – input stream

tributary.streaming.calculations.rolling.**EMA**(*node*, *window_width=10*, *full_only=False*)

Node to take the exponential moving average over a window of ticks

Parameters

- **node** (*node*) – input stream
- **window_width** (*int*) – size of window to use
- **full_only** (*bool*) – only return if list is full

tributary.streaming.calculations.rolling.**First**(*node*)
Node to return the first value encountered

tributary.streaming.calculations.rolling.**Last**(*node*)
Node to return the last value encountered

tributary.streaming.calculations.rolling.**Max**(*node*)
Node to take rolling max of inputs

Parameters **node** (*Node*) – input stream

tributary.streaming.calculations.rolling.**Min**(*node*)
Node to take rolling min of inputs

Parameters **node** (*Node*) – input stream

tributary.streaming.calculations.rolling.**SMA**(*node*, *window_width=10*, *full_only=False*)
Node to take the simple moving average over a window of ticks

Parameters

- **node** (*node*) – input stream
- **window_width** (*int*) – size of window to use
- **full_only** (*bool*) – only return if list is full

tributary.streaming.calculations.rolling.**Sum**(*node*)
Node to take rolling sum inputs

If stream type is iterable, will do += sum(input). If input stream type is not iterable, will do += input.

Parameters **node** (*Node*) – input stream

8.2 Functional

tributary.functional.**pipeline**(*foos*, *foo_callbacks*, *foo_kwargs=None*, *on_data=<built-in function print>*, *on_data_kwargs=None*)
Pipeline a sequence of functions together via callbacks

Parameters

- **foos** (*list of callables*) – list of functions to pipeline
- **foo_callbacks** (*List[str]*) – list of strings indicating the callback names (kwargs of the foos)
- **foo_kwargs** (*List[dict]*) –
- **on_data** (*callable*) – callable to call at the end of the pipeline
- **on_data_kwargs** (*dict*) – kwargs to pass to the on_data function>?

tributary.functional.**run_submit**(*fn*, *function_to_call*, **args*, ***kwargs*)

tributary.functional.**stop**()
Stop the executor for the pipeline runtime

tributary.functional.**submit**(*fn*, **args*, ***kwargs*)
Submit a function to be run on the executor (internal)

Parameters

- **fn** (*callable*) – function to call
- **args** (*tuple*) – args to pass to function
- **kwargs** (*dict*) – kwargs to pass to function

tributary.functional.**wrap**(*function*, **args*, ***kwargs*)
wrap a function in a partial

tributary.functional.utils.**map**(*data*, **callbacks*)
Pass data to multiple callbacks

Parameters

- **data** (*any*) – data to pass to all callbacks
- **callbacks** (*tuple*) – callbacks to pass data to

tributary.functional.utils.**merge**(*data1*, *data2*, *callback*)
merge two data sources into one callback

Parameters

- **data1** (*any*) – first data to pass to callback
- **data2** (*any*) – second data to pass to callback
- **callback** (*callable*) – callback to pass data to

tributary.functional.utils.**reduce**(*callback*, **datas*)
merge multiple data sources into one callback

Parameters

- **callback** (*callable*) – callback to pass data to
- **datas** (*tuple*) – data to pass to callback

tributary.functional.utils.**split**(*data*, *callback1*, *callback2*)
Pass data to 2 callbacks

Parameters

- **data** (*any*) – data to pass to both callbacks
- **callback1** (*callable*) – first function to call
- **callback2** (*callable*) – second function to call

8.2.1 Input

tributary.functional.input.**http**(*url*, *callback*, *interval=1*, *repeat=1*, *json=False*, *wrap=False*, *field=None*, *proxies=None*, *cookies=None*)
Connect to url and pipe results through the callback

Parameters

- **url** (*str*) – url to connect to
- **callback** (*callable*) – function to call on websocket data
- **interval** (*int*) – interval to re-query

- **repeat** (*int*) – number of times to request
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list
- **field** (*str*) – field to index result by
- **proxies** (*list*) – list of URL proxies to pass to requests.get
- **cookies** (*list*) – list of cookies to pass to requests.get

tributary.functional.input.**kafka**(*callback*, *servers*, *group*, *topics*, *json=False*, *wrap=False*, *interval=1*)

Connect to kafka server and pipe results through the callback

Parameters

- **callback** (*callable*) – function to call on websocket data
- **servers** (*list*) – kafka bootstrap servers
- **group** (*str*) – kafka group id
- **topics** (*list*) – list of kafka topics to connect to
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – socketio wai interval

tributary.functional.input.**socketio**(*url*, *callback*, *channel=''*, *field=''*, *sendinit=None*, *json=False*, *wrap=False*, *interval=1*)

Connect to socketIO server and pipe results through the callback

Parameters

- **url** (*str*) – url to connect to
- **callback** (*callable*) – function to call on websocket data
- **channel** (*str*) – socketio channel to connect through
- **field** (*str*) – field to index result by
- **sendinit** (*list*) – data to send on socketio connection open
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list
- **interval** (*int*) – socketio wai interval

tributary.functional.input.**ws**(*url*, *callback*, *json=False*, *wrap=False*)

Connect to websocket and pipe results through the callback

Parameters

- **url** (*str*) – websocket url to connect to
- **callback** (*callable*) – function to call on websocket data
- **json** (*bool*) – load websocket data as json
- **wrap** (*bool*) – wrap result in a list

8.2.2 Output

8.3 Lazy

8.3.1 Input

8.3.2 Output

8.3.3 Calculations

tributary.lazy.calculations.ops.**Abs**(*self*)

tributary.lazy.calculations.ops.**Add**(*self*, *other*)

tributary.lazy.calculations.ops.**And**(*self*, *other*)

tributary.lazy.calculations.ops.**Arccos**(*self*)

tributary.lazy.calculations.ops.**Arcsin**(*self*)

tributary.lazy.calculations.ops.**Arctan**(*self*)

tributary.lazy.calculations.ops.**Average**(*self*, **others*)

tributary.lazy.calculations.ops.**Bool**(*self*)

tributary.lazy.calculations.ops.**Ceil**(*self*)

tributary.lazy.calculations.ops.**Cos**(*self*)

tributary.lazy.calculations.ops.**Div**(*self*, *other*)

tributary.lazy.calculations.ops.**Equal**(*self*, *other*)

tributary.lazy.calculations.ops.**Erf**(*self*)

tributary.lazy.calculations.ops.**Exp**(*self*)

tributary.lazy.calculations.ops.**Float**(*self*)

tributary.lazy.calculations.ops.**Floor**(*self*)

tributary.lazy.calculations.ops.**Ge**(*self*, *other*)

tributary.lazy.calculations.ops.**Gt**(*self*, *other*)

tributary.lazy.calculations.ops.**Int**(*self*)

tributary.lazy.calculations.ops.**Invert**(*self*)

tributary.lazy.calculations.ops.**Le**(*self*, *other*)

tributary.lazy.calculations.ops.**Len**(*self*)

tributary.lazy.calculations.ops.**Log**(*self*)

tributary.lazy.calculations.ops.**Lt**(*self*, *other*)

tributary.lazy.calculations.ops.**Mod**(*self*, *other*)

tributary.lazy.calculations.ops.**Mult**(*self*, *other*)

tributary.lazy.calculations.ops.**Negate**(*self*)

tributary.lazy.calculations.ops.**Not**(*self*)

tributary.lazy.calculations.ops.**NotEqual**(*self*, *other*)

tributary.lazy.calculations.ops.**Or**(*self*, *other*)

tributary.lazy.calculations.ops.**Pow**(*self*, *other*)

tributary.lazy.calculations.ops.**Round**(*self*, *ndigits=0*)

tributary.lazy.calculations.ops.**Sin**(*self*)

tributary.lazy.calculations.ops.**Sqrt**(*self*)

tributary.lazy.calculations.ops.**Str**(*self*)

tributary.lazy.calculations.ops.**Sub**(*self*, *other*)

tributary.lazy.calculations.ops.**Sum**(*self*, **others*)

tributary.lazy.calculations.ops.**Tan**(*self*)

tributary.lazy.calculations.ops.**binary**(*node1*, *other*, *name*, *lam*)

tributary.lazy.calculations.ops.**n_nary**(*node*, *others*, *name*, *lam*)

tributary.lazy.calculations.ops.**unary**(*node*, *name*, *lam*)

8.4 Symbolic

tributary.symbolic.**construct_lazy**(*expr*, *modules=None*)
Construct Lazy tributary class from sympy expression

Parameters

- **expr** (*sympy expression*) – A Sympy expression
- **modules** (*list*) – a list of modules to use for sympy's lambdify function

Returns tributary.lazy.LazyGraph

tributary.symbolic.**construct_streaming**(*expr*, *modules=None*)
Construct streaming tributary class from sympy expression

Parameters

- **expr** (*sympy expression*) – A Sympy expression
- **modules** (*list*) – a list of modules to use for sympy's lambdify function

Returns:

tributary.symbolic.**graphviz**(*expr*)
Plot sympy expression tree using graphviz :param expr: :type expr: sympy expression

tributary.symbolic.**parse_expression**(*expr*)
Parse string as sympy expression :param expr: string to convert to sympy expression :type expr: string

tributary.symbolic.**symbols**(*expr*)
Get symbols used in sympy expression :param expr: :type expr: sympy expression

tributary.symbolic.**traversal**(*expr*)
Traverse sympy expression tree :param expr: :type expr: sympy expression

8.5 Common

class tributary.base.**StreamEnd**
Bases: object

Indicates that a stream has nothing left in it

instance = None

class tributary.base.**StreamNone**
Bases: object

indicates that a stream does not have a value

all_bin_ops(*other*)

all_un_ops()

float()

instance = None

int()

class tributary.base.**StreamRepeat**
Bases: object

Indicates that a stream has a gap, this object should be ignored and the previous action repeated

instance = None

tributary.thread.**run**(*target*, *timeout=1*)
Helper for running a thread

Parameters

- **target** (*function*) – function to run on a thread
- **timeout** (*int*) – how long to wait for target to return

Returns result of the function

Return type data

tributary.utils.**LazyToStreaming**(*lazy_node*)

Python Module Index

t

Index

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W