IoTPy Documentation

Release 1.0.0

Author

May 01, 2017

Contents

Contents:

CHAPTER 1

IoTPy package

Subpackages

IoTPy.code package

Subpackages

IoTPy.code.agents package

Submodules

IoTPy.code.agents.element_agent module

This module consists of functions that operate on a single input stream to produce a single output stream. These stream-functions encapsulate functions that operate on standard data types such as integers. In this module, the encapsulated function operates on a single element of the input stream to produce a single element of the output stream.

Functions in the module:

1. element_map_agent
2. map_stream
3. element_filter_agent
4. filter_stream

Functions from stream to stream:

1. map_stream is similar to map in Python except that it operates on streams rather than lists.
2. filter_stream is similar to filter in Python except that it operates on streams rather than lists.

Agents:

1. element_map_agent is the agent used by map_stream.

2. element_filter_agent is the agent used by filter_stream

IoTPy.code.agents.element_agent.**element_filter_agent**(*func*, *in_stream*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

This agent uses the boolean function func to filter its single input stream to produce a single output stream.

Parameters **func: function**

function from an element of the in_stream to Boolean.

in_stream: Stream The single input stream of this agent

out_stream: Stream The single output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

Uses

—-

- Agent
- check_map_agent_arguments
- check_num_args_in_func

IoTPy.code.agents.element_agent.**element_map_agent**(*func*, *in_stream*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

This agent maps the function func from its single input stream to its single output stream.

Parameters **func: function**

function from an element of the in_stream to an element of the out_stream.

in_stream: Stream The single input stream of this agent

out_stream: Stream The single output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

Uses

—-

- Agent
- check_map_agent_arguments
- check_num_args_in_func

Used by

———-

map_stream

IoTPy.code.agents.element_agent.**filter_stream**(*function*, *in_stream*, *state=None*, **args*, ***kwargs*)

filter_stream returns out_stream, a stream obtained by applying the filter function to the input stream, in_stream.

Parameters **function: function**

function from an element of the in_stream to Boolean.

in_stream: Stream The single input stream of this agent

state: object function operates on a state, args, and kwargs

Returns

———-

out_stream: Stream The output stream generated by map_stream

Uses

—-

element_filter_agent

Used by

———-

map_stream

IoTPy.code.agents.element_agent.**map_stream**(*function*, *in_stream*, *state=None*, **args*, ***kwargs*)

map_stream returns out_stream, a stream obtained by applying function to each element of the input stream, in_stream.

Parameters **function: function**

function from an element of the in_stream to an element of the out_stream.

in_stream: Stream The single input stream of this agent

state: object function operates on a state, args, and kwargs

Returns

———-

out_stream: Stream The output stream generated by map_stream

Uses

———-

- element_map_agent

IoTPy.code.agents.element_agent.**test_element_agent**()

IoTPy.code.agents.gate_agent module

IoTPy.code.agents.gate_agent.**gate_agent**(*in_stream*, *out_stream*, *call_streams*, *name=None*)

Parameters **in_stream: Stream**

The single input stream of the agent

out_stream: Stream The single output stream of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns

——-

Agent. The agent created by this function.

IoTPy.code.agents.gate_agent.**test_gate_agents**()

IoTPy.code.agents.list_agent module

The list agent is one of a collection of agents, each of which has a different type of transition function, func.

The list-agent func operates on a list input and returns a list. func may also operate on a state which can be of any type.

This file has the following agents: list_map_agent: Has a single input stream and a single output stream. list_sink_agent: Has a single input stream and no outputs. list_merge_agent: Has a list of input streams and a single output stream. list_split_agent: Has a single input stream and a list of output streams. list_many_agent: Has a list of input streams and a list of output streams.

A single input stream is in_stream and a single output stream is out_stream. A list of input streams is in_streams and a list of output streams is out_streams. In general, plurals refer to lists and singular to single elements.

IoTPy.code.agents.list_agent.**list_many_agent**(*func*, *in_streams*, *out_streams*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from an input list to an output list

in_streams: list of Stream The input streams of the agent

out_streams: list of Stream The output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_stream. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.list_agent.**list_map_agent**(*func*, *in_stream*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

This agent maps the function func from its single input stream to its single output stream.

Parameters **func: function**

function from a list (a slice of the in_stream) to a list (a slice of the out_stream).

in_stream: Stream The single input stream of this agent

out_stream: Stream The single output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns Agent.

The agent created by this function.

IoTPy.code.agents.list_agent.**list_merge_agent**(*func*, *in_streams*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from a list of lists (one list per input stream) to an output list

in_streams: list of Stream The list of input streams of the agent

out_stream: Stream The single output stream of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.list_agent.**list_sink_agent**(*func*, *in_stream*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

This agent applies func to its single input stream. It has no output streams.

Parameters **func: function**

function from a list (a slice of the in_stream) to None (no output).

in_stream: Stream The single input stream of this agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.list_agent.**list_split_agent**(*func*, *in_stream*, *out_streams*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from an input list to a list of lists (one per output stream).

in_stream: Stream The single input stream of the agent

out_streams: list of Stream The list of output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.list_agent.**test_list_agents**()

IoTPy.code.agents.merge module

This module consists of functions that merge multiple input streams into a single output stream.

Functions in the module:

1. element_merge_agent
2. zip_stream
3. zip_map
4. asynch_merge_agent
5. mix
6. blend_agent
7. blend

Merge functions:

1. zip_stream is similar to zip in Python except that it operates on streams rather than lists.
2. zip_map is map_stream(zip_stream()), i.e., first zip then map the result.

3. mix is an asynchronous merge of the input streams. The elements of the output stream identify the input streams that generated the elements.
4. blend is a merge followed by a map.

Agents:

1. element_merge_agent is the agent used by zip_stream and zip_map.
2. asynch_merge_agent is the agent used by mix.
3. blend_agent is the agent used by blend.

IoTPy.code.agents.merge.**asynch_merge**(*in_streams*, *out_stream*)

Parameters **in_streams: list of input streams**

out_stream: single output stream

IoTPy.code.agents.merge.**blend**(*function*, *in_streams*, *state=None*, **args*, ***kwargs*)

IoTPy.code.agents.merge.**blend_agent**(*func*, *in_streams*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

Parameters **func: function**

function from an input list and args and kwargs to an output list

in_streams: list of Stream The list of input streams of the agent

out_stream: Stream The single output stream of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.merge.**element_many_agent**(*func*, *in_streams*, *out_streams*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

Parameters **func: function**

function from an input list and args and kwargs to an output list

in_streams: list of Stream The input streams of the agent

out_streams: list of Stream The output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_stream. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.merge.**element_merge_agent**(*func*, *in_streams*, *out_stream*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

Parameters **func: function**

function from an input list and args and kwargs to an output list

in_streams: list of Stream The list of input streams of the agent

out_stream: Stream The single output stream of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.merge.**merge_split**(*function*, *in_streams*, *num_out_streams*, *state=None*, **args*, ***kwargs*)

IoTPy.code.agents.merge.**mix**(*in_streams*)

IoTPy.code.agents.merge.**tests**()

IoTPy.code.agents.merge.**zip_agent**(*in_streams*, *out_stream*)

zip_agent zips the input streams and returns values in out_stream

Parameters **in_streams: list of Stream**

The list of input streams that are zipped

out_stream: Stream The Stream to add values to

Uses

———-

- element_merge_agent

IoTPy.code.agents.merge.**zip_map**(*function*, *in_streams*, *state=None*, **args*, ***kwargs*)

zip_map returns out_stream, a stream obtained by applying function, with the specified state, args and kwargs, to the elements obtained by zipping the input streams.

Parameters **in_streams: list of Stream**

The list of input streams that are zipped

state: object function operates on a state, args, and kwargs

Returns

———-

out_stream: Stream The output stream generated by zip_map

Uses

———-

- element_merge_agent

IoTPy.code.agents.merge.**zip_stream**(*in_streams*)
zip_stream returns out_stream, a stream obtained by zipping the input streams. zip_stream is similar to zip.

Parameters **in_streams: list of Stream**

The list of input streams that are zipped

state: object function operates on a state, args, and kwargs

Returns

———-

out_stream: Stream The output stream generated by zip_stream

IoTPy.code.agents.sink module

IoTPy.code.agents.sink.**element_sink_agent**(*func*, *in_stream*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)
This agent applies func to its single input stream. It has no output streams.

Parameters **func: function**

function from an element of the in_stream and args and kwargs to None (no return).

in_stream: Stream The single input stream of this agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.sink.**element_stream_to_queue_agent**(*stream*, *queue*, *name=None*, *descriptor=None*)

IoTPy.code.agents.sink.**sink**(*function*, *in_stream*, *state=None*, **args*, ***kwargs*)

IoTPy.code.agents.sink.**test_element_agents**()

IoTPy.code.agents.source module

This module consists of sources. A source is a function that generates a single stream. These functions are executed in separate threads. A function returns the thread and the output stream that are generated.

Functions in the module:

1. make_outstream_and_thread is a helper function used by the other functions in this module.

2. function_to_stream generates a stream whose values are specified by successive call to a function.
3. file_to_stream generates a stream from a file specified by a file name.
4. list_to_stream generates a stream from a list
5. queue_to_stream generates a stream by waiting for messages to arrive on a queue and then appending the messages to a stream.

IoTPy.code.agents.source.**file_to_stream**(*function*, *filename*, *time_interval=None*, *stream_length=None*, *state=None*, **args*, ***kwargs*)

Places lines in a file on a stream.

Parameters function: function

This function is applied to each line read from the file and the result of this function is appended to the output stream.

filename: str The name of the file that is read.

time_interval: float or int (optional) The next line of the file is read every time_interval seconds.

stream_length: int (optional) file_to_stream terminates after stream_length values are placed on the output stream. If stream_length is None then the file_to_stream terminates when the entire file is read.

state: object (optional) The state of the function; an argument of function.

args: list Positional arguments of function

kwargs: dict Keyword arguments of function

Returns: thread, out_stream

——-

thread: Thread.thread The thread created by this function. The thread must be started and thread.join() may have to be called to ensure that the thread terminates execution.

out_stream: Stream The stream created by this function.

IoTPy.code.agents.source.**function_to_stream**(*function*, *time_interval=0*, *stream_length=None*, *state=None*, *out_stream=None*, *push=False*, **args*, ***kwargs*)

Calls a function and places returned values on a stream called out_stream.

Parameters function: function

This function is called and the result of this function is appended to the output stream.

time_interval: float or int (optional), time in seconds An element is placed on the output stream every time_interval seconds.

stream_length: int (optional) function_to_stream terminates after stream_length values are placed on the output stream. If stream_length is None then function_to_stream terminates only when an exception is raised.

state: object (optional) The state of the function; an argument of function.

out_stream: Stream (optional) The output stream

push: boolean (optional) Whether the source will push values to the stream

args: list Positional arguments of function

kwargs: dict Keyword arguments of function

Returns: thread, out_stream

———-

thread: Thread.thread The thread created by this function. The thread must be started and thread.join() may have to be called to ensure that the thread terminates execution.

out_stream: Stream The stream created by this function.

IoTPy.code.agents.source.**list_to_stream**(*function*, *in_list*, *time_interval=None*, *stream_length=None*, *state=None*, **args*, ***kwargs*)

Places elements in a list on a stream.

Parameters function: function

This function is applied to each element in the list and the result of this function is appended to the output stream.

in_list: list The list that is read.

out_stream: Stream The output stream on which messages are placed.

time_interval: float or int (optional) The next line of the file is read every time_interval seconds.

stream_length: int (optional) file_to_stream terminates after stream_length values are placed on the output stream. If stream_length is None then the file_to_stream terminates when the entire file is read.

state: object (optional) The state of the function; an argument of function.

args: list Positional arguments of function

kwargs: dict Keyword arguments of function

Returns: thread, out_stream

———-

thread: Thread.thread The thread created by this function. The thread must be started and thread.join() may have to be called to ensure that the thread terminates execution.

out_stream: Stream The stream created by this function.

IoTPy.code.agents.source.**make_outstream_and_thread**(*function*, *time_interval*, *stream_length*, *state*, *args*, *kwargs*, *run_infinite_stateless*, *run_infinite_stateful*, *run_finite_stateless*, *run_finite_stateful*, *run_push_stateless*, *run_push_stateful*, *out_stream=None*, *push=False*)

Helper function called by the other functions. Parameters ———-

function: function This function is applied to each line read from the source and the result of this function is appended to the output stream.

time_interval: float or int, time in seconds An element is placed on an output stream every time_interval seconds. (See the functions that call this one for out_stream.)

stream_length: int (optional) The source terminates after stream_length values are placed on the output stream. If stream_length is None then the source function terminates when an exception is raised or the source has no more data.

state: object (optional) The state of the function. This is a parameter of function

args: list list of positional arguments of function

kwargs: dict dict of keywork arguments of function

run_infinite_stateless, run_infinite_stateful: functions run_finite_stateless, run_finite_stateful: functions run_push_stateless, run_push_stateful: functions

Target functions of threads

out_stream: Stream The output stream (default is None)

push: boolean Whether the source will push values to the stream (default is False)

This function structures creates threads and varies the target function for the threads for the following 6 cases:
1. stream_length and state are None and push is False:

target is run_infinite_stateless

2.stream_length is None and state is not None and push is False: target is run_infinite_stateful

3.stream_length is not None and state is None and push is False: target is run_finite_stateless

4.stream_length is not None and state is not None and push is False: target is run_finite_stateful

5.push is True and state is None: target is run_push_stateless

6.push is True and state is not None: target is run_push_stateful

IoTPy.code.agents.source.**queue_to_stream**(*function*, *queue*, *stop_message=None*, *state=None*, **args*, ***kwargs*)

Reads queue and places messages it receives from the queue on a stream called out_stream.

Parameters function: function

This function is applied to each element read from the queue and the result of this function is appended to the output stream.

queue: multiprocessing.Queue or Queue.Queue The queue from which messages are obtained.

stop_message: object When a stop_message is received from the queue the thread terminates execution.

state: object (optional) The state of the function; an argument of function.

args: list Positional arguments of function

kwargs: dict Keyword arguments of function

Returns: thread, out_stream

——-

thread: Thread.thread The thread created by this function. The thread must be started and thread.join() may have to be called to ensure that the thread terminates execution.

out_stream: Stream The stream created by this function.

IoTPy.code.agents.source.**test**()

IoTPy.code.agents.split module

This module consists of functions that split a single input stream into multiple output streams.

Functions in the module:

1. element_split_agent
2. split_stream
3. separate_agent
4. separate
5. unzip_agent
6. unzip_stream

Split functions:

1. split_stream: a function returns a list with each element placed in a different stream.
2. separate: separate is the inverse of mix (see merge.py). The elements of the input stream are pairs (i, v) and the value v is placed on the i-th output stream.
3. unzip_stream: unzip_stream is the inverse of zip_stream (see merge.py). The elements of the input stream are lists and the i-th element of the list is placed on the i-th output stream.

Agents:

1. element_split_agent: agent used by split_stream
2. separate_agent: agent used by separate
3. unzip_agent: not used by any function in this module. It is retained only for backward compatibility.

IoTPy.code.agents.split.**element_split_agent**(*func*, *in_stream*, *out_streams*, *state=None*, *call_streams=None*, *name=None*, **args*, ***kwargs*)

Parameters **func: function**

function from an input list and args and kwargs to an output list

in_stream: Stream The single input stream of the agent

out_streams: list of Stream The list of output streams of the agent

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.split.**separate**(*in_stream*, *num_out_streams*)

separate returns out_streams, a list of num_out_streams streams. separate is the inverse of mix (see merge.py). The elements of the input stream are pairs (i, v) and the value v is placed on the i-th output stream.

Parameters **in_stream: Stream**

The stream that will be split

num_out_streams: int The number of output streams.

Returns out_streams: List of Stream

The output streams generated by split_stream

Uses

- separate_agent

IoTPy.code.agents.split.**separate_agent**(*in_stream*, *out_streams*, *name=None*)

Parameters **in_stream: Stream**

The single input stream of the agent

out_streams: list of Stream The list of output streams of the agent

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.split.**split_stream**(*function*, *in_stream*, *num_out_streams*, *state=None*, **args*, ***kwargs*)

split_stream returns out_streams, a list of num_out_streams streams. The function, with the specified state, args and kwargs, is applied to the elements of the input stream. The return value of the function must be a list of length num_out_streams. The i-th value of the returned list is placed in the i-th output stream.

Parameters **in_stream: Stream**

The stream that will be split

num_out_streams: int The number of output streams.

state: object function operates on a state, args, and kwargs

Returns

———-

out_streams: List of Stream The output streams generated by split_stream

Uses

———-

- element_split_agent

IoTPy.code.agents.split.**test_element_agents**()

IoTPy.code.agents.split.**timed_unzip**(*in_stream*, *num_out_streams*)

timed_unzip returns out_streams which is a list of num_out_streams streams. timed_unzip is the inverse of timed_zip (see merge.py). The elements of the input stream are pairs (t, v) where v is a list of length num_out_streams. The i-th element of the list, with the timestamp t is placed on the i-th output stream if and only if v is not None.

Parameters **in_stream: Stream**

The stream that will be split

num_out_streams: int The number of output streams.

Returns out_streams: List of Stream

The output streams generated by split_stream

Uses

- split_stream

IoTPy.code.agents.split.**unzip_agent**(*in_stream*, *out_streams*, *name=None*)

Parameters **in_stream: Stream**

The single input stream of the agent

out_streams: list of Stream The list of output streams of the agent

name: str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

Note

—-

Not used by any function in this module. Used by external modules

IoTPy.code.agents.split.**unzip_stream**(*input_stream*, *num_out_streams*)

unzip_stream returns out_streams, a list of num_out_streams streams. unzip_stream is the inverse of zip_stream (see merge.py). The elements of the input stream are lists of length num_out_streams and the i-th element of the list is placed on the i-th output stream.

Parameters **in_stream: Stream**

The stream that will be split

num_out_streams: int The number of output streams.

Returns out_streams: List of Stream

The output streams generated by split_stream

Uses

- element_split_agent

IoTPy.code.agents.timed_agent module

This module has timed_zip and timed_window which are described in the manual documentation.

IoTPy.code.agents.timed_agent.**test_timed_zip_agents**()

IoTPy.code.agents.timed_agent.**timed_window**(*function*, *in_stream*, *window_duration*, *step_time*, *state=None*, *args=[]*, *kwargs={}*)

IoTPy.code.agents.timed_agent.**timed_window_agent**(*func*, *in_stream*, *out_stream*, *window_duration*, *step_time*, *window_start_time=0*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

IoTPy.code.agents.timed_agent.**timed_zip**(*list_of_streams*)

IoTPy.code.agents.timed_agent.**timed_zip_agent**(*in_streams*, *out_stream*, *call_streams=None*, *name=None*)

Parameters **in_streams: list of Stream**

The list of input streams of the agent

out_stream: Stream The single output stream of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

Notes

——

Each stream in in_streams must be a stream of tuples or lists or NumPy arrays where element[0] is a time and where time is a total order. Each stream in in_stream must be strictly monotonically increasing in time.

out_stream merges the in_streams in order of time. An element of out_stream is a list where element[0] is a time T and element[1] is a list consisting of all elements of in in_streams that have time T.

IoTPy.code.agents.timed_merge_agent module

IoTPy.code.agents.timed_merge_agent.**test_timed_merge_agents**()

IoTPy.code.agents.timed_merge_agent.**timed_merge_agent**(*in_streams*, *out_stream*, *call_streams=None*, *name=None*)

Parameters **in_streams: list of Stream**

The list of input streams of the agent

out_stream: Stream The single output stream of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

——-

Agent. The agent created by this function.

Notes

——

Each stream in in_streams must be a stream of tuples or lists or NumPy arrays where element[0] is a time and where time is a total order. Each stream in in_stream must be strictly monotonically increasing in time.

out_stream merges the in_streams in order of time. An element of out_stream is a list where element[0] is a time T and element[1] is a list consisting of all elements of in in_streams that have time T.

IoTPy.code.agents.window_agent module

IoTPy.code.agents.window_agent.**dynamic_window_agent**(*func*, *in_stream*, *out_stream*, *state*, *min_window_size*, *max_window_size*, *step_size*, *args={}*, *kwargs={}*)

IoTPy.code.agents.window_agent.**test_window_agents**()

IoTPy.code.agents.window_agent.**window**(*function*, *in_stream*, *window_size*, *step_size*, *state=None*, *args=[]*, *kwargs={}*)

IoTPy.code.agents.window_agent.**window_many_agent**(*func*, *in_streams*, *out_streams*, *window_size*, *step_size*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from a list of windows with one window for each input stream to an output list containing a single element for each output stream.

in_streams: list of Stream The list of input streams of the agent

out_streams: list of Stream The list of output streams of the agent

window_size: int Positive integer. The size of the moving window

step_size: int Positive integer. The step size of the moving window

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

——-

Agent. The agent created by this function.

IoTPy.code.agents.window_agent.**window_map_agent**(*func*, *in_stream*, *out_stream*, *window_size*, *step_size*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from a single element of the input stream to a single element of the output stream

in_stream: Stream The single input stream of this agent

out_stream: Stream The single output streams of the agent

window_size: int Positive integer. The size of the moving window

step_size: int Positive integer. The step size of the moving window

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.window_agent.**window_merge_agent**(*func*, *in_streams*, *out_stream*, *window_size*, *step_size*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from a list of windows with one window per input stream to a single element of the output stream.

in_streams: list of Stream The list of input streams of the agent

out_stream: Stream The single output streams of the agent

window_size: int Positive integer. The size of the moving window

step_size: int Positive integer. The step size of the moving window

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

———-

Agent. The agent created by this function.

IoTPy.code.agents.window_agent.**window_split_agent**(*func*, *in_stream*, *out_streams*, *window_size*, *step_size*, *state=None*, *call_streams=None*, *name=None*, *args=[]*, *kwargs={}*)

Parameters **func: function**

function from a window to a list containing a single element for each output stream.

in_stream: Stream The single input stream of the agent

out_streams: list of Stream The list of output streams of the agent

window_size: int Positive integer. The size of the moving window

step_size: int Positive integer. The step size of the moving window

state: object The state of the agent

call_streams: list of Stream The list of call_streams. A new value in any stream in this list causes a state transition of this agent.

name: Str Name of the agent created by this function.

Returns

——-

Agent. The agent created by this function.

Module contents

IoTPy.code.helper_functions package

Submodules

IoTPy.code.helper_functions.check_agent_parameter_types module

IoTPy.code.helper_functions.check_agent_parameter_types.**check_func_output_for_multiple_stre**

IoTPy.code.helper_functions.check_agent_parameter_types.**check_function_type**(*name*, *func*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_in_lists_type**(*name*, *in_lists*, *num_in_streams*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_list_of_streams_type**(*list_of_stre*, *agent_nam*, *pa-*, *ram-*, *e-*, *ter_name*)

Helper function to check the types of streams used by an agent. Used by: check_agent_arguments()

IoTPy.code.helper_functions.check_agent_parameter_types.**check_many_agent_arguments**(*func*, *in_streams*, *out_stream*, *call_stream*, *name*)

Checks the types of arguments used by an agent.

IoTPy.code.helper_functions.check_agent_parameter_types.**check_map_agent_arguments**(*func*, *in_stream*, *out_stream*, *call_streams*, *name*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_merge_agent_arguments**(*func*, *in_stream*, *out_stream*, *call_stream*, *name*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_num_args_in_func**(*state*, *name*, *func*, *func_args*, *func_kwargs*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_num_args_in_func_no_state**(*name*, *func*, *func*, *func*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_num_args_in_func_with_state**(*n*, *f*, *f*, *f*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_sink_agent_arguments**(*func*, *in_stream*, *call_stream*, *name*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_split_agent_arguments**(*func*, *in_stream*, *out_stream*, *call_stream*, *name*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_stream_type**(*name*, *in_or_out_stream_str*, *stream*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_window_and_step_sizes**(*name*, *window_size*, *step_size*)

IoTPy.code.helper_functions.check_agent_parameter_types.**check_zip_agent_arguments**(*in_streams*, *out_stream*, *name*)

IoTPy.code.helper_functions.recent_values module

IoTPy.code.helper_functions.recent_values.**recent_values**(*stream*)

Module contents

Submodules

IoTPy.code.agent module

This module contains the Agent class. The Agent and Stream classes are the building blocks of PythonStreams.

class IoTPy.code.agent.**Agent**(*in_streams*, *out_streams*, *transition*, *state=None*, *call_streams=None*, *name=None*, *stream_manager=None*)

Bases: *IoTPy.code.agent.Blackbox*

An agent is an automaton: a state-transition machine. An agent has only one important method: the method next() that makes the agent execute a state transition.

An agent has lists of: (1) input streams, (2) output streams and (3) call streams. Streams are described in Stream.py.

During a state transition an agent: (1) May read values from its input streams. (Note that

reading values in a stream does not change the stream.)

2.Append values to the tails of its output streams.

3.Change the agent's own state.

When a call stream is modified the agent's next() method is called which causes the agent to execute a state transition.

The default is that every input stream is also a call stream, i.e., the agent executes a state transition when any of its input streams is modified. For performance reasons, we may not want the agent to execute state transitions each time any of its input streams is modified; we may want the agent to execute state transitions periodically — for example, every second. In this case, the call streams will be different from the input streams. A call stream that has a value appended to it every second will cause the agent to execute a state transition every second.

Parameters **in_streams** : list of streams

The list of the agent's input streams. This list may be empty.

out_streams : list of streams

The list of the agent's output streams. This list may be empty.

call_streams : list of streams

When a new value is added to a stream in this list a state transition is invoked. This the usual way (but not the only way) in which state transitions occur. A state transiton for an agent ag can also be executed by calling ag.next()

state: object

The state of the agent. The state is updated after a transition.

transition: function

This function is called by next() which is the state-transition operation for this agent. An agent's state transition is specified by its transition function.

stream_manager : function

Each stream has management variables, such as whether the stream is open or closed. After a state-transition the agent executes the stream_manager function to modify the management variables of the agent's output and call streams.

name : str, optional

name of this agent

Attributes

_in_lists: list of InList	InList defines the slice of a list. The j-th element of _in_lists is an InList that defines the slice of the j-th input stream that can be read by this agent in a state transition. For example, if listj = _in_lists[j].lists startj = _in_lists[j].start stopj = _in_lists[j].stop Then this agent can read the slice: listj[startj:stopj] of the jth input stream. This slice is a slice of the most recent values of the stream.
_out_lists: list	The j-th element of _out_lists is the list of values to be appended to the j-th output stream after the state transition.

Methods

next()	Execute a state transition. The method has 3 parts: (i) set up the data structures to execute a state transition, (ii) call the transition function to: (a) get the values to be appended to output streams, (b) get the next state, and (c) update 'start' indices for each input stream. The agent no longer accesses elements of its input streams with indices earlier (i.e. smaller) than 'start'. (iii) update data structures after the transition.

next(*stream_name=None*)

Execute the next state transition.

This function does the following: Part 1: set up data structures for the state transition. Part 2: execute the state transition by calling self.transition Part 3: update data structures after the transition.

This method can be called by any agent and is called whenever a value is appended to any stream in call_streams

Parameters **stream_name** : str, optional

A new value was appended to the stream with name stream_name as a result of which this agent executes a state transition.

class IoTPy.code.agent.**Blackbox**(*in_streams*, *out_streams*, *name*)

Bases: object

IoTPy.code.agent.**EPSILON = 1e-12**

class IoTPy.code.agent.**InList**

Bases: tuple

Attributes

list	Alias for field number 0
start	Alias for field number 1
stop	Alias for field number 2

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

list
Alias for field number 0

start
Alias for field number 1

stop
Alias for field number 2

IoTPy.code.agent.**main**()

IoTPy.code.stream module

This module contains the Stream class. The Stream and Agent classes are the building blocks of PythonStreams. (Version 1.0 June 16, 2016. Created by: Mani Chandy)

class IoTPy.code.stream.**Stream**(*name='NoName'*, *proc_name='UnknownProcess'*, *initial_value=[]*, *num_in_memory=16*, *min_history=4*)

Bases: object

A stream is a sequence of values. Agents can: (1) Append values to the tail of stream and close a stream. (2) Read a stream. (3) Subscribe to be notified when a value is added to a stream. (See Agent.py for details of agents.)

The ONLY way in which a stream can be modified is that values can be appended to its tail. The length of a stream (number of elements in its sequence) can stay the same or increase, but never decrease. If at some point, the length of a stream is k, then from that point onwards, the first k elements of the stream remain unchanged.

A stream is written by only one agent. Any number of agents can read a stream, and any number of agents can subscribe to a stream. An agent can be a reader and a subscriber and a writer of the same stream. An agent may subscribe to a stream without reading the stream's values; for example the agent may subscribe to a clock stream and the agent executes a state transition when the the clock stream has a new value, regardless of the value.

Parameters **name** : str, optional

name of the stream. Though the name is optional a named stream helps with debugging. default : 'NoName'

proc_name : str, optional

The name of the process in which this agent executes. default: 'UnknownProcess'

initial_value : list or array, optional

The list (or array) of initial values in the stream. If a stream starts in a known state, i.e., with a known sequence of messages, then set the initial_value to this sequence. default : []

num_in_memory: positive int, optional

It is the initial value of the number of elements in the stream that are stored in main memory. If the stream has 9000 elements and num_in_memory is 100 then the most recent 100 elements of the stream are stored in main memory and the earlier 8900 elements are stored in a file or discarded. num_in_memory may change. It increases if a

reader is reading the i-th value of the stream and if j is the index of the most recent value in the stream and **|j - i|** exceeds num_in_memory. It may decrease if the gap between the indices of the most recent value in the stream and the earliest value being read by any agent is less than num_in_memory default : DEFAULT_NUM_IN_MEMORY

specified in SystemParameters.py

min_history: non-negative int, optional

The minimum number of elements of the stream that are stored in main memory. If min_history is 3 and the stream has 9000 elements then the elements 8997, 8998, 8999 will be stored in main memory even if no agent is reading the stream. min_history is used primarily for debugging. A debugger may need to read early values of the stream, and reading from main memory is faster than reading from a file. default : DEFAULT_MIN_HISTORY

specified in SystemParameters.py.

Notes

1.AGENTS SUBSCRIBING TO A STREAM

An agent is a state-transition automaton and the only action that an agent executes is a state transition. If agent x is a subscriber to a stream s then x.next() — a state transition of x — is invoked whenever messages are appended to s.

The only point at which an agent executes a state transition is when a stream to which the agent subscribes is modified.

An agent x subscribes to a stream s by executing s.call(x).

An agent x unsubscribes from a stream s by executing:

s.delete_caller(x)

2.AGENTS READING A STREAM

2.1 Agent registers for reading

An agent can read a stream only after it registers with the stream as a reader. An agents r registers with a stream s by executing:

s.reader(r)

An agent r deletes its registration for reading s by executing:

s.delete_reader(r)

An agent that reads a stream is also a subscriber to that stream unless the agent has a call-stream. If an agent has no call-stream and stream s is an input stream of that agent, then whenever s is modified, the agent is told to execute a state transition.

2.2 Slice of a stream that can be read by an agent

At any given point, an agent r that has registered to read a stream s can only read some of the most recent values in the stream. The number of values that an agent can read may vary from agent to agent. A reader r can only read a slice:

s[s.start[r]+s.offset: s.stop+s.offset]

of stream s where start[r], stop and offset are defined later.

3.WRITING A STREAM

3.1 Extending a stream

When an agent is created it is passed a list of streams that it can write.

An agent adds a single element v to a stream s by executing:

s.append(v)

An agent adds the sequence of values in a list l to a stream s by executing:

s.extend(l)

The operations append and extend of streams are analogous to operations with the same names on lists.

3.2 Closing a Stream

A stream is either closed or open. Initially a stream is open. An agent that writes a stream s can close s by executing:

s.close()

A closed stream cannot be modified.

4.MEMORY

4.1 The most recent values of a stream

The most recent elements of a stream are stored in main memory. In addition, the user can specify whether all or part of the stream is saved to a file.

Associated with each stream s is a list (or array) s.recent that includes the most recent elements of s. If the value of s is a sequence:

s[0], ..., s[n-1],

at a point in a computation then at that point, s.recent is a list

s[m], .., s[n-1]

for some m, followed by some padding (usually a sequence of zeroes, as described later).

The system ensures that all readers of stream s only read elements of s that are in s.recent.

4.2 Slice of a stream that can be read

Associated with a reader r of stream s is an integer s.start[r]. Reader r can only read the slice:

s.recent[s.start[r] :]

of s.recent.

For readers r1 and r2 of a stream s the values s.start[r1] and s.start[r2] may be different.

4.3 When a reader finishes reading part of a stream

Reader r informs stream s that it will only read values with indexes greater than or equal to j in the list, recent, by executing

s.set_start(r, j)

which causes s.start[r] to be set to j.

5.OPERATION

5.1 Memory structure

Associated with a stream is: (1) a list or NumPy darray, recent. (2) a nonnegative integer stop where:

1.recent[: stop] contains the most recent values of the stream,

2.the slice recent[stop:] is padded with padding values (either 0 or 0.0 or default value specified by the numpy data type).

3.**a nonnegative integer s.offset where**

recent[i] = stream[i + offset] for 0 <= i < s.stop

Example: if the sequence of values in a stream is:

0, 1, .., 949

and s.offset is 900, then s.recent[i] = s[900+i] for i in 0, 1, ..., 49.

and s.recent[i] is the default value for i > 49.

Invariant: len(s) = s.offset + s.stop

where len(s) is the number of values in stream s.

The size of s.recent increases and decreases so that the length of slice that any reader may read is less than the length of s.recent.

The entire stream, or the stream up to offset, can be saved in a file for later processing. You can also specify that no part of the stream is saved to a file.

In the current implementation old values of the stream are not saved.

5.2 Memory Management

We illustrate memory management with the following example with num_in_memory=4 and buffer_size=1

Assume that a point in time, for a stream s, the list of values in the stream is [1, 2, 3, 10, 20, 30]; num_in_memory=4; s.offset=3; s.stop=3; and s.recent = [10, 20, 30, 0]. The length of s.recent is num_in_memory (i.e. 4). The s.stop (i.e. 3) most recent values in the stream are 10, 20, 30. s[3] == 10 == s.recent[0] s[4] == 20 == s.recent[1] s[5] == 30 == s.recent[2] The values in s.recent[s.stop:] are padded values (zeroes).

A reader r of stream s has access to the list: s.recent[s.start[r] : s.stop]

Assume that s has two readers r and q where: s.start[r] = 1 and s.start[q] = 2. So agent r can read the slice [1:3] of recent which is the list [20, 30], and agent q can read the slice [2:3] of recent which is the list [30]. An invariant is:

0 <= s.start[r] <= s.stop

for any reader r.

When a value v is appended to stream s, v is inserted in s.recent[s.stop], replacing a default value, and s.stop is incremented. If s.stop >= len(s.recent) then a new s.recent is created and the values that may be read by any reader are copied into the new s.recent, and s.start, s.stop, and s._begin are modified.

Example: Start with the previous example. (Assume min_history is 0. This parameter is discussed in the next paragraph.) When a new value, 40 is appended to the stream, the list of values in s becomes. [1, 2, 3, 10, 20, 30, 40]. s.recent becomes [10, 20, 30, 40], and s.stop becomes 4. Since s.stop >= len(recent), a new copy of recent is made and the elements that are being read in s are copied into the new copy. So, recent becomes [20, 30, 40, 0] because no agent is reading s[3] = 10. Then s.stop becomes 3 and s.offset becomes 4. s.start is modified with s.start[r] becoming 0 and s.start[q] becoming 1, so that r continues to have access to values of the stream after 20; thus r can now read the list [20, 30, 40] and q can read the list [30, 40].

At a later point, agent r informs the stream that it no longer needs to access elements 20, 30, 40 and so s.start[r] becomes 3. Later agent q informs the stream that it no longer needs to access element 30 and s.start[q] becomes 2. At this point r has access to the list [] and q to the list [40].

Now suppose the agent writing stream s extends the stream by the list [50, 60, 70, 80]. At this point, agent q needs to access the list [40, 50, 60, 70, 80] which is longer than len(recent). In this case the size of recent is doubled, and the new recent becomes: [40, 50, 60, 70, 80, 0, 0, 0], with s.start[r] = 1 and s.start[q] = 0. s.stop becomes 5.

Example of min_history = 4. Now suppose the stream is extended by 90, 100 so that s.recent becomes [40, 50, 60, 70, 80, 90, 100, 0] with s.stop = 7. Suppose r and q inform the stream that they no longer need to access the elements currently in the stream, and so s.start[r] and s.start[q] become 7. (In this case the size of recent may be made smaller (halved); but, this is not done in the current implementation and will be done later.) Next suppose the stream is extended by [110]. Since r and q only need to read this value, all the earlier values could be deleted from recent; however, min_history elements must remain in recent and so recent becomes: [80, 90, 100, 110, 0, 0, 0, 0]

Attributes

recent	(list or NumPy array.) A list or array that includes the most recent values of the stream. This list or array is padded with default values (see stop).
stop	(int) index into the list recent. s.recent[:s.stop] contains the s.stop most recent values of stream s. s.recent[s.stop:] contains padded values.
offset: int	index into the stream used to map the location of an element in the entire stream with the location of the same element in s.recent, which only contains the most recent elements of the stream. For a stream s: s.recent[i] = s[i + s.offset] for i in range(s.stop) Note: In later versions, offset will be implemented as a list of ints.
start	(dict of readers.) key = reader value = start index of the reader Reader r can read the slice: s.recent[s.start[r] : s.stop] in s.recent which is equivalent to the following slice in the entire stream: s[s.start[r]+s.offset: s.stop+s.offset] Invariant: For all readers r: stop - start[r] <= len(recent) This invariant is maintained by increasing the size of recent when necessary.
subscribers_set: set	the set of subscribers for this stream. Subscribers are agents to be notified when an element is added to the stream.
closed: boolean	True if and only if the stream is closed. An exception is thrown if a value is appended to a closed stream.
close_message: _close or np.NaN	This message is appended to a stream to indicate that when this message is received the stream should be closed. If the stream is implemented as a list then close_message is _close, and for StreamArray the close_message is np.NaN (not a number).
_begin	(int) index into the list, recent recent[:_begin] is not being accessed by any reader; therefore recent[:_begin] can be deleted from main memory. Invariant: for all readers r: _begin <= min(start[r])

Methods

append(value)	Append a single value to the end of the stream.
call(agent)	Register a subscriber for this stream.
close()	Close this stream.”
delete_caller(agent)	Delete a subscriber for this stream.
delete_reader(reader)	Delete this reader from this stream.
extend(value_list)	Extend the stream by value_list.

Continued on next page

Table 1.3 – continued from previous page

get_contents_after_column_value(...)	Assumes that the stream consists of rows where the number of elements in each row exceeds column_number.
get_index_for_column_value(column_number, value)	Similar to get_contents_after_column_value except that the value returned is an index into recent rather than the sequence of rows.
get_last_n(n)	**Parameters**
get_latest()	Returns the latest element in the stream.
get_latest_n(n)	Same as get_last_n()
is_empty()	Returns: boolean ——- True if and only if this stream is empty.
print_recent()	
reader(r[, start_index])	A newly registered reader starts reading recent from index start, i.e., reads recent[start_index:s.stop] If reader has already been registered with this stream its start value is updated to start_index.
set_name(name)	
set_start(reader, starting_value)	The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

append(*value*)
Append a single value to the end of the stream.

call(*agent*)
Register a subscriber for this stream.

close()
Close this stream."

delete_caller(*agent*)
Delete a subscriber for this stream.

delete_reader(*reader*)
Delete this reader from this stream.

extend(*value_list*)
Extend the stream by value_list.

Parameters **value_list: list**

get_contents_after_column_value(*column_number*, *value*)
Assumes that the stream consists of rows where the number of elements in each row exceeds column_number. Also assumes that values in the column with index column_number are in increasing order.

Returns the rows in the stream for which: row[column_number] >= value

get_index_for_column_value(*column_number*, *value*)
Similar to get_contents_after_column_value except that the value returned is an index into recent rather than the sequence of rows.

get_last_n(*n*)

Parameters **n** : positive integer

Returns The list of the last n elements of the stream. If the

number of elements in the stream is less than n, then it returns all the elements in the stream.

Note

Requirement: n >= self.min_history

get_latest()
Returns the latest element in the stream. If the stream is empty then it returns the empty list

get_latest_n(*n*)
Same as get_last_n()

is_empty()

True if and only if this stream is empty.

print_recent()

reader(*r*, *start_index=0*)
A newly registered reader starts reading recent from index start, i.e., reads recent[start_index:s.stop] If reader has already been registered with this stream its start value is updated to start_index.

set_name(*name*)

set_start(*reader*, *starting_value*)
The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

class IoTPy.code.stream.**StreamArray**(*name='NoName'*, *proc_name='UnknownProcess'*, *dimension=0*, *dtype=<type 'float'>*, *initial_value=None*, *num_in_memory=16*, *min_history=4*)
Bases: *IoTPy.code.stream.Stream*

Methods

append(value)	**Parameters**
call(agent)	Register a subscriber for this stream.
close()	Close this stream."
delete_caller(agent)	Delete a subscriber for this stream.
delete_reader(reader)	Delete this reader from this stream.
extend(lst)	See extend() for the class Stream.
get_contents_after_column_value(...)	Assumes that the stream consists of rows where the number of elements in each row exceeds column_number.
get_contents_after_time(start_time)	
get_index_for_column_value(column_number, value)	Similar to get_contents_after_column_value except that the value returned is an index into recent rather than the sequence of rows.
get_last_n(n)	**Parameters**
get_latest()	Returns the latest element in the stream.
get_latest_n(n)	Same as get_last_n()
is_empty()	Returns: boolean ——- True if and only if this stream is empty.

Continued on next page

Table 1.4 – continued from previous page

print_recent()	
reader(r[, start_index])	A newly registered reader starts reading recent from index start, i.e., reads recent[start_index:s.stop] If reader has already been registered with this stream its start value is updated to start_index.
set_name(name)	
set_start(reader, starting_value)	The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

append(*value*)

Parameters value: 1-D numpy array

The value appended to the StreamArray

Notes

See self._create_recent() for a description of the elements of the stream.

extend(*lst*)

See extend() for the class Stream. Extend the stream by an numpy ndarray.

Parameters lst: np.ndarray

Notes

See self._create_recent() for a description of the elements of the stream.

get_contents_after_time(*start_time*)

class IoTPy.code.stream.**StreamSeries**(*name=None*)

Bases: *IoTPy.code.stream.Stream*

Methods

append(value)	Append a single value to the end of the stream.
call(agent)	Register a subscriber for this stream.
close()	Close this stream."
delete_caller(agent)	Delete a subscriber for this stream.
delete_reader(reader)	Delete this reader from this stream.
extend(value_list)	Extend the stream by value_list.
get_contents_after_column_value(...)	Assumes that the stream consists of rows where the number of elements in each row exceeds column_number.
get_index_for_column_value(column_number, value)	Similar to get_contents_after_column_value except that the value returned is an index into recent rather than the sequence of rows.
get_last_n(n)	**Parameters**

Continued on next page

Table 1.5 – continued from previous page

get_latest()	Returns the latest element in the stream.
get_latest_n(n)	Same as get_last_n()
is_empty()	Returns: boolean ——- True if and only if this stream is empty.
print_recent()	
reader(r[, start_index])	A newly registered reader starts reading recent from index start, i.e., reads recent[start_index:s.stop] If reader has already been registered with this stream its start value is updated to start_index.
set_name(name)	
set_start(reader, starting_value)	The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

class IoTPy.code.stream.**StreamTimed**

Bases: *IoTPy.code.stream.StreamArray*

Methods

append(value)	**Parameters**
call(agent)	Register a subscriber for this stream.
close()	Close this stream.”
delete_caller(agent)	Delete a subscriber for this stream.
delete_reader(reader)	Delete this reader from this stream.
extend(lst)	See extend() for the class Stream.
get_contents_after_column_value(...)	Assumes that the stream consists of rows where the number of elements in each row exceeds column_number.
get_contents_after_time(start_time)	
get_index_for_column_value(column_number, value)	Similar to get_contents_after_column_value except that the value returned is an index into recent rather than the sequence of rows.
get_last_n(n)	**Parameters**
get_latest()	Returns the latest element in the stream.
get_latest_n(n)	Same as get_last_n()
is_empty()	Returns: boolean ——- True if and only if this stream is empty.
print_recent()	
reader(r[, start_index])	A newly registered reader starts reading recent from index start, i.e., reads recent[start_index:s.stop] If reader has already been registered with this stream its start value is updated to start_index.
set_name(name)	
set_start(reader, starting_value)	The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

class IoTPy.code.stream.**TimeAndValue**
Bases: tuple

Attributes

time	Alias for field number 0
value	Alias for field number 1

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

time
Alias for field number 0

value
Alias for field number 1

IoTPy.code.stream.**main**()

IoTPy.code.stream.**remove_novalue_and_open_multivalue**(*l*)
This function returns a list which is the same as the input parameter l except that (1) _no_value elements in l are deleted and (2) each _multivalue element in l is opened

i.e., for an object _multivalue(list_x) each element of list_x appears in the returned list.

IoTPy.code.system_parameters module

SYSTEM_PARAMETERS

Module contents

IoTPy.encapsulation_functions package

Subpackages

IoTPy.encapsulation_functions.attribute package

Submodules

IoTPy.encapsulation_functions.attribute.assemble module

IoTPy.encapsulation_functions.attribute.assemble.**assemble**(*params*)
Gets the attribute of a value

Parameters **params** : dict

Dict of values.

Module contents

IoTPy.encapsulation_functions.dynamic_window package

Submodules

IoTPy.encapsulation_functions.dynamic_window.assemble module

IoTPy.encapsulation_functions.dynamic_window.assemble.**assemble**(*params*)
Assembles dynamic window

Parameters **params** : dict

Dict of values

Returns agent

Agent for dynamic window

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window module

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window.**f**(*v*)

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window.**g**(*v*, *state*)

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.dynamic_window.make_part_from_dynamic_window.**make_part_from_**
Makes a part from the dynamic window template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the dynamic window template. The parameters are: in: a Stream out: a Stream function: a Python function min_window_size : int max_window_size : int step_size : int initial_state: (optional) arbitrary parameters : (optional) list

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.func package

Submodules

IoTPy.encapsulation_functions.func.assemble module

IoTPy.encapsulation_functions.func.assemble.**assemble**(*params*)
Runs a function

Parameters **params** : dict

Dict of values

Module contents

IoTPy.encapsulation_functions.map package

Submodules

IoTPy.encapsulation_functions.map.assemble module

IoTPy.encapsulation_functions.map.assemble.**assemble**(*params*)
Assembles map

Parameters **params** : dict

Dict of values

Returns agent

Agent for map

IoTPy.encapsulation_functions.map.make_part_from_map module

IoTPy.encapsulation_functions.map.make_part_from_map.**f**(*v*)

IoTPy.encapsulation_functions.map.make_part_from_map.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.map.make_part_from_map.**g**(*v*, *state*)

IoTPy.encapsulation_functions.map.make_part_from_map.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.map.make_part_from_map.**make_part_from_map**(*params*)
Makes a part from the map template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the map template. The parameters are: in: a Stream out: a Stream function: a Python function initial_state: (optional) arbitrary parameters: (optional) list of arguments of function

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.mix package

Submodules

IoTPy.encapsulation_functions.mix.assemble module

IoTPy.encapsulation_functions.mix.assemble.**assemble**(*params*)
Assembles mix

Parameters **params** : dict

Dict of values

Returns agent

Agent for mix

IoTPy.encapsulation_functions.mix.make_part_from_mix module

IoTPy.encapsulation_functions.mix.make_part_from_mix.**f**(*v*)

IoTPy.encapsulation_functions.mix.make_part_from_mix.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.mix.make_part_from_mix.**g**(*v*, *state*)

IoTPy.encapsulation_functions.mix.make_part_from_mix.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.mix.make_part_from_mix.**make_part_from_mix**(*params*)
Makes a part from the mix template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the mix template. The parameters are: in: list of Stream out: a Stream

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.separate package

Submodules

IoTPy.encapsulation_functions.separate.assemble module

IoTPy.encapsulation_functions.separate.assemble.**assemble**(*params*)
Assembles separate

Parameters **params** : dict

Dict of values

Returns agent

Agent for separate

IoTPy.encapsulation_functions.separate.make_part_from_separate module

IoTPy.encapsulation_functions.separate.make_part_from_separate.**f**(*v*)

IoTPy.encapsulation_functions.separate.make_part_from_separate.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.separate.make_part_from_separate.**g**(*v*, *state*)

IoTPy.encapsulation_functions.separate.make_part_from_separate.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.separate.make_part_from_separate.**make_part_from_separate**(*para*
Makes a part from the separate template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the separate template. The parameters are: in: a Stream out: list of Stream

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.sink package

Submodules

IoTPy.encapsulation_functions.sink.assemble module

IoTPy.encapsulation_functions.sink.assemble.**assemble**(*params*)
Assembles sink

Parameters **params** : dict

Dict of values

Returns agent

Agent for sink

IoTPy.encapsulation_functions.sink.make_part_from_sink module

IoTPy.encapsulation_functions.sink.make_part_from_sink.**f**(*v*)

IoTPy.encapsulation_functions.sink.make_part_from_sink.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.sink.make_part_from_sink.**g**(*v*, *state*)

IoTPy.encapsulation_functions.sink.make_part_from_sink.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.sink.make_part_from_sink.**make_part_from_sink**(*params*)
Makes a part from the sink template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the sink template. The parameters are: in: a Stream function: a Python function initial_state: (optional) arbitrary parameters: (optional) list of arguments of function

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.source package

Submodules

IoTPy.encapsulation_functions.source.assemble module

IoTPy.encapsulation_functions.source.assemble.**assemble**(*params*)
Assembles source

Parameters **params** : dict

Dict of values

Returns agent

Agent for source

IoTPy.encapsulation_functions.source.make_part_from_source module

IoTPy.encapsulation_functions.source.make_part_from_source.**f**(*v*)

IoTPy.encapsulation_functions.source.make_part_from_source.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.source.make_part_from_source.**g**(*v*, *state*)

IoTPy.encapsulation_functions.source.make_part_from_source.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.source.make_part_from_source.**make_part_from_source**(*params*)
Makes a part from the source template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the source template. The parameters are: out: a Stream function: a Python function initial_state: (optional) arbitrary parameters: (optional) list of arguments of function

Module contents

IoTPy.encapsulation_functions.unzip package

Submodules

IoTPy.encapsulation_functions.unzip.assemble module

IoTPy.encapsulation_functions.unzip.assemble.**assemble**(*params*)
Assembles unzip

Parameters **params** : dict

Dict of values

Returns agent

Agent for unzip

IoTPy.encapsulation_functions.unzip.make_part_from_unzip module

IoTPy.encapsulation_functions.unzip.make_part_from_unzip.**f**(*v*)

IoTPy.encapsulation_functions.unzip.make_part_from_unzip.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.unzip.make_part_from_unzip.**g**(*v*, *state*)

IoTPy.encapsulation_functions.unzip.make_part_from_unzip.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.unzip.make_part_from_unzip.**make_part_from_unzip**(*params*)
Makes a part from the unzip template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the unzip template. The parameters are: in: a Stream out: list of Stream

Returns The part that was made.

Module contents

IoTPy.encapsulation_functions.zip package

Submodules

IoTPy.encapsulation_functions.zip.assemble module

IoTPy.encapsulation_functions.zip.assemble.**assemble**(*params*)
Assembles zip

Parameters **params** : dict

Dict of values

Returns agent

Agent for zip

IoTPy.encapsulation_functions.zip.make_part_from_zip module

IoTPy.encapsulation_functions.zip.make_part_from_zip.**f**(*v*)

IoTPy.encapsulation_functions.zip.make_part_from_zip.**ff**(*v*, *arg_0*)

IoTPy.encapsulation_functions.zip.make_part_from_zip.**g**(*v*, *state*)

IoTPy.encapsulation_functions.zip.make_part_from_zip.**h**(*v*, *state*, *arg_0*)

IoTPy.encapsulation_functions.zip.make_part_from_zip.**make_part_from_zip**(*params*)
Makes a part from the zip template given its arguments.

Parameters **template_arguments: str**

A JSON string that specifies the values of the parameters of the zip template. The parameters are: in: a Stream out: list of Stream

Returns The part that was made.

Module contents

Module contents

IoTPy.modules package

Subpackages

IoTPy.modules.Geomap package

Submodules

IoTPy.modules.Geomap.geomap module

class IoTPy.modules.Geomap.geomap.**Geomap**(*figsize=(1000, 800)*, ***kwargs*)

Mapping framework for plotting data onto a map.

Given latitude - longitude coordinates, this framework allows data to be plotted onto a world map with specified region and projection. Data can be plotted with different colors as well as labels. Previous data can also be cleared.

Parameters **figsize** : tuple

A tuple containing the width and height of the plot for the map (the default is (1000, 800)).

kwargs : keyword arguments

Keyword arguments. The valid keywords are the keywords for the __init__ method of GMapOptions.

Methods

clear()	Clears all plotted data on the map.
plot(x[, index, text, color, s])	Plots data onto the map.

clear()

Clears all plotted data on the map.

plot(*x*, *index=None*, *text=None*, *color='Blue'*, *s=30*)

Plots data onto the map.

This function allows data in the form of latitude-longitude coordinates to be plotted on the map. Supports coloring by index or name as well as text labels.

Parameters **x** : numpy.ndarray

A numpy array containing data to be plotted. Dimensions must be n * 2, where n is the number of data points. The first column is the latitude and the second column is the longitude.

index : numpy.ndarray or list, optional

A numpy array or list containing indices for coloring the data. Dimensions must be n * 1, where n is the number of data points. If not provided, data is colored with *color*.

text : numpy.ndarray, optional

A numpy array containing string labels for each data point. Dimensions must be n * 1, where n is the number of data points.

color : string, optional

A string specifying the color of the data points (the default is blue). Used if index is not provided.

s : int, optional

An int specifying the size of the data points (the default is 30).

Module contents

IoTPy.modules.ML package

Subpackages

IoTPy.modules.ML.KMeans package

Submodules

IoTPy.modules.ML.KMeans.KMeansStream module

class IoTPy.modules.ML.KMeans.KMeansStream.**KMeansStream**(*draw*, *output*, *k*, *incremental=True*, *figsize=(1000, 500)*)

Helper class for kmeans clustering.

This class provides train and predict functions for using kmeans with *Stream_Learn*.

Parameters **draw** : boolean

Describes whether the data is to be plotted (data must have 2 or less dimensions).

output : boolean

Describes whether debug info is to be printed. Info includes average error, average number of iterations, current number of iterations, and number of changed points over time.

k : int

Describes the number of clusters to train.

incremental : boolean, optional

Describes whether the kmeans algorithm is run incrementally or not (the default is True). If incremental, then previous clusters are used to initialize new clusters. Otherwise, clusters are reinitialized randomly for each window.

figsize : tuple, optional

A tuple containing the width and height of the plot for the map (the default is (15, 8)).

Attributes

train	(function) The train function with signature as required by *Stream_Learn*.
predict	(function) The predict function with signature as required by 'Stream_Learn'.
avg_iterations	(float) The average number of iterations per window of data trained.
avg_error	(float) The average error per window of data trained.

Methods

reset()	Resets the KMeans functions and average values.

reset()
Resets the KMeans functions and average values.

Resets: train, predict, avg_iterations, avg_error

class IoTPy.modules.ML.KMeans.KMeansStream.**Model**(*k*)

IoTPy.modules.ML.KMeans.kmeans module

IoTPy.modules.ML.KMeans.kmeans.**computeCentroids**(*X*, *index*, *k*)
Finds the centroids for the data given the index of the closest centroid for each data point.

Parameters **X** : numpy.ndarray

A numpy array with dimensions n * 2 for some integer n.

index : numpy.ndarray

A numpy array with dimensions n * 1 that describes the closest centroid to each point in *X*.

k : int

Describes the number of centroids. k - 1 is the maximum value that appears in *index*.

Returns **centroids** : numpy.ndarray

A numpy array with dimensions *k* * 2.

Notes

The centroids are computed by taking the mean of each group of points in *X* with the same index value. For i in [0, k), centroids[i] is the mean of all data points X[j] where index[j] is i.

IoTPy.modules.ML.KMeans.kmeans.**evaluate_error**(*X*, *centroids*, *index*)
Returns the mean squared error.

Parameters **X** : numpy.ndarray

A numpy array with 2 columns.

centroids : numpy.ndarray

A numpy array with 2 columns.

index : numpy.ndarray

A numpy array with 1 column.

Returns float

The mean squared error.

Notes

The mean squared error is calculated as the average squared distance of each point from the closest centroid.

IoTPy.modules.ML.KMeans.kmeans.**findClosestCentroids**(*X*, *centroids*)
Returns a numpy array containing the index of the closest centroid for each point in X.

Parameters **X** : numpy.ndarray

A numpy array with 2 columns.

centroids : numpy.ndarray

A numpy array with 2 columns.

Returns **index** : numpy.ndarray

A numpy array with dimensions n * 1, where n is the number of rows in *X*. For each row i in *index*, index[i] is in [0, k) where k is the number of rows in *centroids*.

IoTPy.modules.ML.KMeans.kmeans.**init_plot**(*figsize=(1000, 500)*)
Initializes the plot.

Parameters **figsize** : tuple, optional

A tuple containing the width and height of the plot (the default is (1000, 800)).

IoTPy.modules.ML.KMeans.kmeans.**initialize**(*k*, *low*, *high*)
Returns k random points with x and y coordinates in [low, high).

Parameters **k** : int

The number of points to return.

low : int

The lower bound (inclusive) for a point.

high : int

The upper bound (exclusive) for a point.

Returns **centroids** : numpy.ndarray

Numpy array with dimensions *k* by 2.

IoTPy.modules.ML.KMeans.kmeans.**initializeCentroids**(*X*, *k*)
Returns k random points from the data *X* without replacement.

Parameters **X** : numpy.ndarray

A numpy array with dimensions n * 2, where n >= *k*.

k : int

The number of points to return

Returns numpy.ndarray

Numpy array with dimensions *k* by 2.

IoTPy.modules.ML.KMeans.kmeans.**initializeData**(*n*, *k*, *scale*, *low*, *high*)

Initialize n points around k random centroids each with a normal distribution and scale.

Parameters **n** : int

Describes the numbe of points to make around each centroid.

k : int

Describes the number of centroids.

scale : int

Describes the scale for the distribution.

low : int

The lower bound (inclusive) for a centroid.

high : int

The upper bound (exclusive) for a centroid.

Returns **X** : numpy.ndarray

A numpy array with dimensions (*n* * *k*) * 2.

IoTPy.modules.ML.KMeans.kmeans.**initializeDataCenter**(*centroid*, *scale*, *n*)

Initialize n points with a normal distribution and scale around a centroid.

Parameters **centroid** : numpy.ndarray

Numpy array with dimensions 1 * 2.

scale : int

Describes the scale for the distribution.

n : int

Describes the number of points to make.

Returns **X** : numpy.ndarray

A numpy array with dimensions *n* * 2.

IoTPy.modules.ML.KMeans.kmeans.**kmeans**(*X*, *k*, *initial_centroids=None*, *draw=False*, *output=False*, *source=None*)

Runs kmeans until clusters stop moving.

Parameters **X** : numpy.ndarray

A numpy array with 2 columns.

k : int

Describes the number of centroids.

initial_centroids : numpy.ndarray, optional

A numpy array with initial centroids to run the algorithm. This array has with dimensions *k* * 2. If not provided, algorithm is initialized with random centroids from the data *X*.

draw : boolean, optional

Describes whether the data is to be plotted (data must have 2 or less dimensions). The default is False.

output : boolean, optional

Describes whether debug info is to be printed (the default is False). Info includes current number of iterations and number of changed points over time.

Returns **centroids** : numpy.ndarray

Numpy array with learned centroids (dimensions are *k* * 2).

index : numpy.ndarray

Numpy array with dimensions n * 1, where n is the number of rows in *X*. Each value describes the closest centroid to each data point in *X*.

num_iters : int

Describes the number of iterations taken to run kmeans.

`IoTPy.modules.ML.KMeans.kmeans.`**`plotKMeans`**(*X*, *centroids*, *previous*, *index*, *source*)

Plots the data and centroids.

This function plots the data with the current centroids and shows the movement of the centroids.

Parameters **X** : numpy.ndarray

A numpy array with 2 columns.

centroids : numpy.ndarray

A numpy array with 2 columns.

previous : numpy.ndarray

A numpy array with 2 columns and the same number of rows as *centroids*.

index : numpy.ndarray

A numpy array with 1 column.

source : list

List of ColumnDataSource

Module contents

IoTPy.modules.ML.LinearRegression package

Submodules

IoTPy.modules.ML.LinearRegression.LinearRegressionStream module

class IoTPy.modules.ML.LinearRegression.LinearRegressionStream.**LinearRegressionStream**(*draw*, *output*, *incremental=True*, *alpha=0.01*, *figsize=(100 500)*)

Helper class for linear regression.

This class provides train and predict functions for using linear regression with *Stream_Learn*.

Parameters **draw** : boolean

Describes whether the data is to be plotted (data must have 1 dimension).

output : boolean

Describes whether debug info is to be printed. Info includes average error and current error.

incremental : boolean, optional

Describes whether the linear regression algorithm is run incrementally or not (the default is True). If incremental, then the algorithm uses incremental calculations for matrix inversion and matrix multiplication if the data has 1 feature, or stochastic gradient descent if the data has more than 1 feature. Otherwise, the algorithm uses linear algebra.

alpha : float, optional

Learning rate for stochastic gradient descent (the default is 0.01). Ignored if incremental is False or if incremental is True and data has 1 feature.

figsize : tuple, optional

A tuple containing the width and height of the plot for the map (the default is (15, 8)).

Attributes

train	(function) The train function with signature as required by *Stream_Learn*.
predict	(function) The predict function with signature as required by *Stream_Learn*.
w	(tuple) The learned weight vector.
avg_error	(float) The average error per window of data trained.

Methods

reset()	Resets the KMeans functions and average values.

reset()
Resets the KMeans functions and average values.

Resets: train, predict, avg_error

class IoTPy.modules.ML.LinearRegression.LinearRegressionStream.**Model**(*num_features*, *incremental=False*)

IoTPy.modules.ML.LinearRegression.linear_regression module

IoTPy.modules.ML.LinearRegression.linear_regression.**evaluate_error**(*X*, *y*, *w*)
Returns the mean squared error.

X [numpy.ndarray] Numpy array of data.

y [numpy.ndarray] Numpy array of outputs. Dimensions are n * 1, where n is the number of rows in *X*.

w [numpy.ndarray] Numpy array with dimensions (m + 1) * 1, where m is the number of columns in *X*.

Returns float

The mean squared error

IoTPy.modules.ML.LinearRegression.linear_regression.**init_plot**(*figsize=(1000, 500)*)
Initializes the plot.

Parameters **figsize** : tuple, optional

A tuple containing the width and height of the plot (the default is (1000, 800)).

IoTPy.modules.ML.LinearRegression.linear_regression.**plot**(*X*, *y*, *w*, *source*, *step_size*, *max_window_size*)
Plot X data, the actual y output, and the prediction line.

Parameters **X** : numpy.ndarray

Numpy array of data with 1 column.

y : numpy.ndarray

Numpy array of outputs. Dimensions are n * 1, where n is the number of rows in *X*.

w : numpy.ndarray

Numpy array with dimensions 2 * 1.

source : list

List of ColumnDataSource

step_size : int

The step size

max_window_size : int

The max window size

IoTPy.modules.ML.LinearRegression.linear_regression.**predict**(*X*, *w*)
Returns the prediction for one data point.

Parameters **X** : numpy.ndarray

Numpy array of data

w : numpy.ndarray

Numpy array with dimensions (m + 1) * 1, where m is the number of columns in *X*.

Returns float

The mean squared error

IoTPy.modules.ML.LinearRegression.linear_regression.**train**(*X*, *y*)

Trains a linear regression model using linear algebra.

Parameters **X** : numpy.ndarray

Numpy array of data

y : numpy.ndarray

Numpy array of outputs. Dimensions are n * 1, where n is the number of rows in *X*.

Returns **w** : numpy.ndarray

Trained vector with dimensions (m + 1) * 1, where m is the number of columns in *X*.

IoTPy.modules.ML.LinearRegression.linear_regression.**train_sgd**(*X*, *y*, *alpha*, *w=None*)

Trains a linear regression model using stochastic gradient descent.

Parameters **X** : numpy.ndarray

Numpy array of data

y : numpy.ndarray

Numpy array of outputs. Dimensions are n * 1, where n is the number of rows in *X*.

alpha : float

Describes the learning rate.

w : numpy.ndarray, optional

The initial w vector (the default is zero).

Returns **w** : numpy.ndarray

Trained vector with dimensions (m + 1) * 1, where m is the number of columns in *X*.

Module contents

IoTPy.modules.ML.examples package

Submodules

IoTPy.modules.ML.examples.linear_regression module

IoTPy.modules.ML.examples.linear_regression.**source**(*state*)

Returns a list of two random values

Parameters **state** : int

The step

Returns list

List of two random values

IoTPy.modules.ML.examples.meetup module

IoTPy.modules.ML.examples.meetup.**plot**(*x*, *y*, *model*, *state*)
Plots Meetup RSVPs and centroids

Parameters **x** : numpy.ndarray

The location data to plot

y : numpy.ndarray

No data

model : object

Kmeans model

state : object

The plot state

Returns Geomap.Geomap

The current map

IoTPy.modules.ML.examples.meetup.**source**(*state*)
Returns the next Meetup RSVP

This function gets the next Meetup RSVP from the Meetup stream and returns the location data.

Parameters **state** : object

The Meetup stream

Returns list

A list containing the location data and the state

IoTPy.modules.ML.examples.twitter module

Module contents

Submodules

IoTPy.modules.ML.plot module

IoTPy.modules.ML.plot.**plot**(*lst*, *state*, *plot_func*, *num_features*)
This function plots data using the plot_func

Parameters **lst** : list

Data to plot

state : object

State used for predict and plot

plot_func : function

A function that processes the data for usage such as visualization. This function takes parameters x and y data, a model object, a state object, and returns an updated state object. This function has the signature (np.ndarray np.ndarray Object Object tuple) -> (Object). The first numpy array x has dimensions i x *num_features*, where *min_window_size*

<= i <= *max_window_size*. The second numpy array y has dimensions i x num_outputs, where num_outputs refers to the number of y outputs for an input. The third parameter is the model object defined by *train_func*. The fourth parameter is a state object defined by this function.

num_features : int

An int that describes the number of features in the data.

IoTPy.modules.ML.predict module

IoTPy.modules.ML.predict.**predict**(*lst*, *state*, *predict_func*, *num_features*)

This function predicts values using predict_func

Parameters **lst** : list

Data to predict

state : object

State for model

predict_func : function

A function that takes as input 2 tuples corresponding to 1 row of data and a model and returns the prediction output. This function has the signature (tuple tuple Object) -> (Object). The first tuple x has *num_features* values and the second tuple y has num_outputs values, where num_outputs refers to the number of y outputs for an input. In the case of unsupervised learning, y is empty.

num_features : int

An int that describes the number of features in the data.

IoTPy.modules.ML.train module

IoTPy.modules.ML.train.**train**(*lst*, *state*, *train_func*, *num_features*)

This function trains a model using train_func

Parameters **lst** : list

Data to train on

state : object

State for train

train_func : function

A function that trains a model. This function takes parameters x and y data, a model object, and a window_state tuple, and returns a trained model object. In the case of *data_train* as a *Stream*, this function has the signature (numpy.ndarray numpy.ndarray Object) -> (Object). The first parameter x will have dimensions i x *num_features*, where *min_window_size* <= i <= *max_window_size*. The second parameter y will have dimensions i x num_outputs, where num_outputs refers to the number of y outputs for an input. For example, num_outputs is 1 for 1 scalar output. For unsupervised learning, num_outputs is 0. In the case of *data_train* as a *numpy* array, this function has the signature (numpy.ndarray numpy.ndarray Object) -> (Object). The first parameter x will have dimensions N x *num_features*, where N refers to the total number of training examples. The second parameter y will have dimensions N x num_outputs where num_outputs is

defined as before. If *data_train* is none of the above, the function has the signature (Object None Object) -> (Object). The first parameter is *data_train*. The third parameter is a model defined by this function. The fourth parameter is a window_state tuple with the values (current_window_size, steady_state, reset, *step_size*, *max_window_size*), where current_window_size describes the number of points in the window, steady_state is a boolean that describes whether the window has reached *max_window_size*, and reset is a boolean that can be set to True to reset the window.

num_features : int

An int that describes the number of features in the data.

Module contents

Submodules

IoTPy.modules.helper_functions module

`IoTPy.modules.helper_functions.`**`iden`**`(*x)`

IoTPy.modules.logger module

`IoTPy.modules.logger.`**`log`**`(x, filename)`

Module contents

IoTPy.tools package

Subpackages

IoTPy.tools.distributed package

Submodules

IoTPy.tools.distributed.server module

`IoTPy.tools.distributed.server.`**`run`**`(host='localhost', user='guest', password='guest')`

Runs a listener to assemble templates

This function assembles parts received from rabbitmq. Each part is run in a process.

Parameters **host** : str, optional

Name of the server for rabbitmq (the default is localhost).

user : str, optional

Name of the user for rabbitmq (the default is guest)

password : str, optional

User password for rabbitmq (the default is guest)

`IoTPy.tools.distributed.server.`**`start`**`(body, host, user, password)`

Assembles a template and starts sink and source

Parameters **body** : str

String containing parameters for template

host : str

Name of the server for rabbitmq

user : str

Name of the user for rabbitmq

password : str

User password for rabbitmq

IoTPy.tools.distributed.sink module

IoTPy.tools.distributed.sink.**addToQueue**(*value*, *exchange*, *channel*, *part_name*, *name*, *index=None*)

Adds a value to a queue for rabbitmq

Parameters **value** : object

Value to send

exchange : str

Name of the exchange

channel : pika channel

Channel to send value

part_name : str

Name of the part to send to

name : str

Name of the parameter

index : int, optional

Index of the parameter (the default is None)

IoTPy.tools.distributed.sink.**sink**(*exchange*, *fields*, *dict_parts*, *host='localhost'*, *user='guest'*, *password='guest'*)

Create sinks for each stream in fields

Parameters **exchange** : str

Name of the exchange

fields : list

List of field names

dict_parts : dict

Dict containing values for fields

host : str, optional

Name of the server for rabbitmq (the default is localhost).

user : str, optional

Name of the user for rabbitmq (the default is guest)

password : str, optional

User password for rabbitmq (the default is guest)

IoTPy.tools.distributed.source module

IoTPy.tools.distributed.source.**source**(*exchange*, *name*, *dict_parts*, *host='localhost'*, *user='guest'*, *password='guest'*)

Listens on rabbitmq queue and adds values to streams

Parameters **exchange** : str

Name of the exchange

name : str

Name of the part

dict_parts : dict

Dict containing values for fields

host : str, optional

Name of the server for rabbitmq (the default is localhost).

user : str, optional

Name of the user for rabbitmq (the default is guest)

password : str, optional

User password for rabbitmq (the default is guest)

Module contents

IoTPy.tools.multicore package

Submodules

IoTPy.tools.multicore.run module

IoTPy.tools.multicore.run.**run**(*name*, *template_name*, *module_name*, *dict_parts*, *queues*)

Assembles a template and starts sink and source

Parameters **name** : str

Name of the part

template_name : str

Name of the template

module_name : str

Name of the module

dict_parts : dict

Dict containing values for fields

queues : dict

Dict of queues for each part

IoTPy.tools.multicore.sink module

IoTPy.tools.multicore.sink.**addToQueue**(*value*, *queue*, *part_name*, *name*, *index=None*)
Adds a value to a queue

Parameters **value** : object

Value to send

queue : multiprocessing.Queue

Queue to add value

part_name : str

Name of the part to send to

name : str

Name of the parameter

index : int, optional

Index of the parameter (the default is None)

IoTPy.tools.multicore.sink.**sink**(*queues*, *fields*, *dict_parts*)
Create sinks for each stream in fields

Parameters **queues** : dict

Dict of queues for each part

fields : list

List of field names

dict_parts : dict

Dict containing values for fields

IoTPy.tools.multicore.source module

IoTPy.tools.multicore.source.**source**(*queue*, *dict_parts*)
Listens on queue and adds values to streams

Parameters **queue** : multiprocessing.Queue

Queue to listen to

dict_parts : dict

Dict containing values for fields

Module contents

Submodules

IoTPy.tools.assemble module

IoTPy.tools.assemble.**assemble**(*name*, *template_name*, *module_name*, *multiprocessing=False*, *distributed=False*, *host='localhost'*, *user='guest'*, *password='guest'*, ***kwargs*)

Assembles a part.

Parameters **name** : str

Name of the part

template_name : str

Name of the template

module_name : str

Name of the module calling assemble

multiprocessing : bool, optional

Describes whether to run the template using multiprocessing (the default is False).

distributed : bool, optional

Describes whether to run the template using distributed computing (the default is False).

host : str, optional

Name of the server for rabbitmq if using distributed (the default is localhost).

user : str, optional

Name of the user for rabbitmq (the default is guest)

password : str, optional

User password for rabbitmq (the default is guest)

kwargs : keyword arguments

Keyword arguments. All inputs and outputs (non-optional) must be keywords

Returns Component

The component for the template

IoTPy.tools.component module

class IoTPy.tools.component.**Component**(*name*, *template_name*, *module_name*, *dict_parts*, *multiprocessing=False*, *distributed=False*, *host='localhost'*, *user='guest'*, *password='guest'*)

Class for a template

This class creates a template and assembles its subparts.

Parameters **name** : str

Name of the part

template_name : str

Name of the template

module_name : str

Name of the module calling assemble

dict_parts : dict

Keyword arguments. All inputs and outputs (non-optional) must be keywords

multiprocessing : bool, optional

Describes whether to run the template using multiprocessing (the default is False).

distributed : bool, optional

Describes whether to run the template using distributed computing (the default is False).

host : str, optional

Name of the server for rabbitmq if using distributed (the default is localhost).

user : str, optional

Name of the user for rabbitmq (the default is guest)

password : str, optional

User password for rabbitmq (the default is guest)

Attributes

name	(str) The name of the part
template_name	(str) The name of the template

IoTPy.tools.db module

IoTPy.tools.db.**find**(*name*)

Finds a template in the db

Parameters **name** : str

Name of the template

Returns list

List of templates

IoTPy.tools.db.**get_template**(*name*)

Finds and returns a template in the db

Parameters **name** : str

Name of the template

Returns dict or None

If template is found, dict is returned. Otherwise None is returned.

IoTPy.tools.db.**is_stream**(*name*)

Returns whether a template has stream fields

Parameters **name** : str

Name of the template

Returns bool

Returns true if the template has stream inputs/outputs, False otherwise. If template does not exist, False is returned.

IoTPy.tools.db.**save_template**(*template*)
Saves template in the db.

This function takes as input a dict specifying a template and saves it in the db. If the template already exists, it is updated.

Parameters **template** : dict

Dict specifying template

IoTPy.tools.helper_functions module

IoTPy.tools.helper_functions.**extend**(*lst*, *index*)
Extends a list to index

This function takes as parameters a list and an index, and extends the list to have length = index + 1.

Parameters **lst** : list

The list to extend

index : int

The index to extend list to

IoTPy.tools.helper_functions.**get_function**(*func_name*, *module_name*)
Returns a function.

This function takes as input a function name and a module name and returns the function.

Parameters **func_name** : str or function

Name of the function. If func_name is a function, it is returned. Otherwise if func_name has module namespaces, it is returned from the modules directory. E.g. if func_name == module_1.func, func is defined in modules/module_1.py

module_name : str

Name of the module where the function is defined. Used if func_name does not have modules.

Returns function

Function with name func_name

IoTPy.tools.helper_functions.**get_value**(*dict_parts*)
Returns values in dict

This function returns values in Value objects in dict_parts.

Parameters **dict_parts: dict**

Dict of Value objects

Returns dict

Dict of values

IoTPy.tools.helper_functions.**get_valueR**(*value*)
Recursively returns values in Value objects

Parameters **value** : list or Value

If value is a list, this function recursively returns the values in the list. Otherwise if value is Value, returns the value.

Returns list or Value or object

IoTPy.tools.helper_functions.**parse_fields**(*field*, *dict_parts*)

Parses fields and returns the field if it is enclosed in ” or if it is a number, otherwise returns the value in dict_parts

Parameters **field** : str or float or list

The field to parse

dict_parts : dict

Dictionary containing values for the part

Returns

———

field

Parsed field

IoTPy.tools.parameter module

class IoTPy.tools.parameter.**Parameter**(*name*, *param*, *index=None*)

Stores parameter info for assemble

Parameters **name** : str

Name of the part

param : str

Name of the field

index : int

Index of the field.

Attributes

name	(str) Name of the part
param	(str) Name of the field
index	(int, optional) Index of the field (the default is None)

Methods

to_list()	Returns the parameter info as a list

to_list()

Returns the parameter info as a list

Returns lst

List containing name and param, and index if valid

IoTPy.tools.parameter.**get_external**(*external*)
Constructs a Parameter object from a list

Parameters **external** : list

List containing name, param, and index (optional)

Returns Parameter

IoTPy.tools.parameter.**get_internal**(*internal*)
Constructs Parameter objects from internal list

This function constructs a source and a destination from a list.

Parameters **internal** : list

List containing source name, source param name, source index (optional), destination name, destination param name, destination index (optional)

Returns Parameter, Parameter

Source and destination Parameter objects

IoTPy.tools.parameter.**make_internal**(*source*, *des*)
Creates list from source and destination Parameter objects

Parameters **source** : Parameter

Parameter object for source

des : Parameter

Parameter object for destination

Returns list

List for connection, parameter for get_internal

See also:

get_internal

IoTPy.tools.parser module

IoTPy.tools.parser.**get_templates**(*x*)
Parses a string containing templates to dict

Parameters **x** : str

String that describes templates

Returns list

List containing dict for templates

IoTPy.tools.parser.**parse_string**(*x*)
Parses a string using the parser

Parameters **x** : str

String that describes template

Returns pyparsing.ParseResults

IoTPy.tools.parser.**parse_template**(*template*)
Creates a dict for a template

Parameters **template** : pyparsing.ParseResults

Template parsed using parser

Returns dict

Dict for template

IoTPy.tools.value module

class IoTPy.tools.value.**Value**(*value=None*)

Class that stores values

Parameters **value** : object, optional

Value to store

Attributes

value	(object) Value stored
dest	(list) List of Parameter destinations

Module contents

Submodules

IoTPy.config module

IoTPy.config.**get**(*key*, *value*)

This function returns the value for a key from config

Parameters **key** : str

The key to find

value : str

The value to find

Returns str

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

V

W

Z