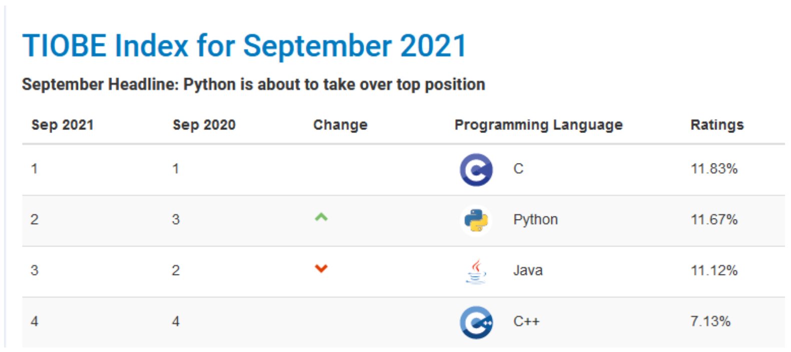
Adding Python to Your Simulation Modeling Workflow

The Pypeline and Alpyne Libraries

Python Popularity

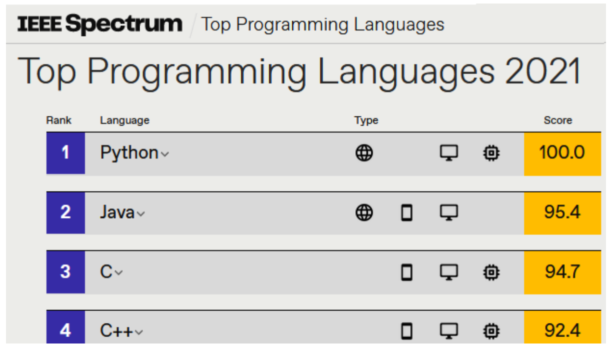
- Multiple independent indices rank Python as #1 or #2 – right next to Java



TIOBE Index for September 2021

September Headline: Python is about to take over top position

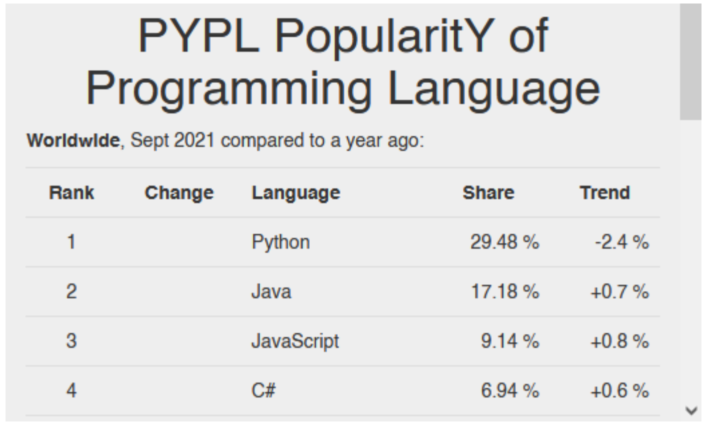
Sep 2021	Sep 2020	Change	Programming Language	Ratings
1	1		C	11.83%
2	3	^	Python	11.67%
3	2	˅	Java	11.12%
4	4		C++	7.13%



IEEE Spectrum Top Programming Languages

Top Programming Languages 2021

Rank	Language	Type	Score
1	Python	🌐 🖥 ⚙	100.0
2	Java	🌐 📱 🖥	95.4
3	C	📱 🖥 ⚙	94.7
4	C++	📱 🖥 ⚙	92.4



PYPL PopularitY of Programming Language

Worldwide, Sept 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.48 %	-2.4 %
2		Java	17.18 %	+0.7 %
3		JavaScript	9.14 %	+0.8 %
4		C#	6.94 %	+0.6 %

https://www.tiobe.com/tiobe-index/ https://spectrum.ieee.org/top-programming-languages/ https://pypl.github.io/PYPL.html

- It's easy to see why considering its:
  - Simple syntax – beginner friendly, elegant syntax
  - Integration with other more efficient languages, such as C and C++
  - Large library covering data science and analytics, AI, and both enterprise and web dev

Python + AnyLogic

- Desirable for simulation modeling workflows, in a wide range of possibilities:
  - Running (Monte Carlo, Parameter Variation) experiments
  - Using optimizers during a simulation
  - Training RL algorithms
  - Testing trained RL policies
  - Enhancing models with data visualization libraries
- Note: These aren't unique to Python!
  - Focus is on opening Python as a convenience / practicality in certain scenarios

Support for Python

- Current and future support for native Python in the AnyLogic ecosystem
  - AnyLogic Cloud API
  - AnyLogic 9

- So why these libraries?
  - Available now (more on that later)
  - Free for all (including PLE users)
  - Desirable for academia or professionals (e.g., tests, experimentation, PoC)

- And why two libraries?
  - Calling Python from within a running AnyLogic model (definition)
  - Using Python to control an exported AnyLogic model (implementation)

Alpyne - Introduction

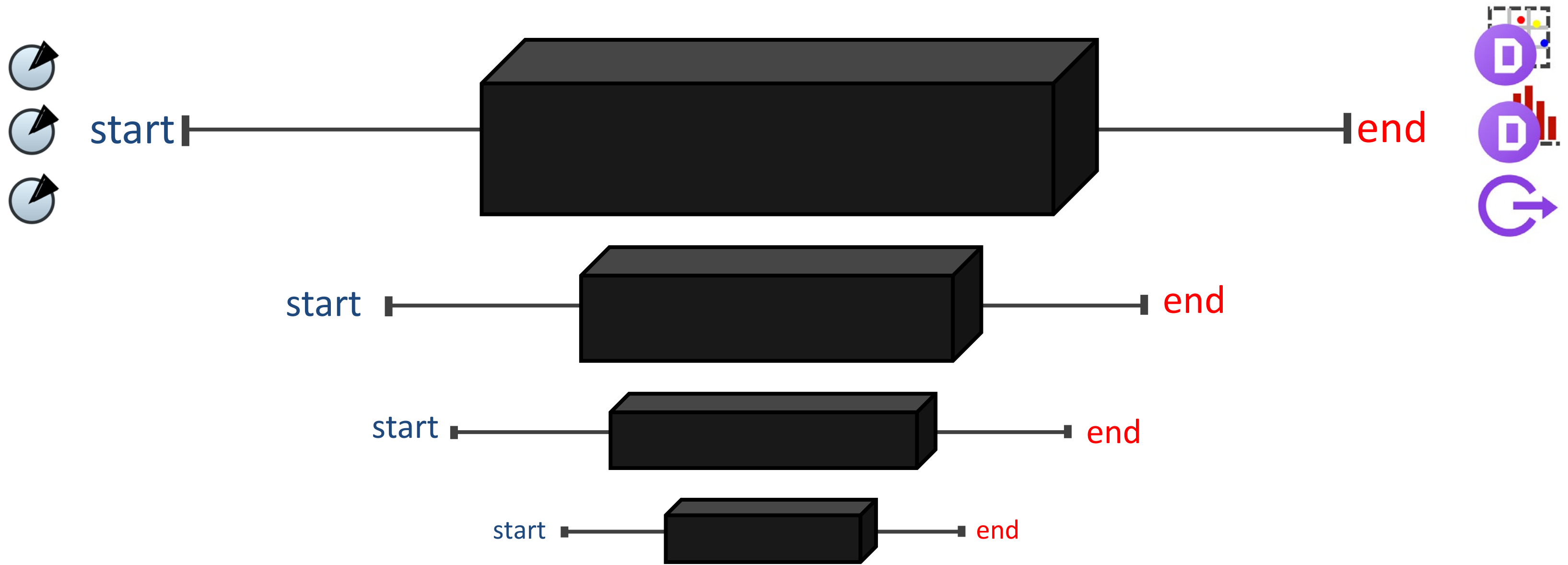
- Alpyne is a Python library that allows any model exported from the RL experiment to be interactively run from a local environment.

Technical Topics Approaching

Intended for any advanced, technical users who wish to train *manually defined RL agents* outside the AnyLogic ecosystem

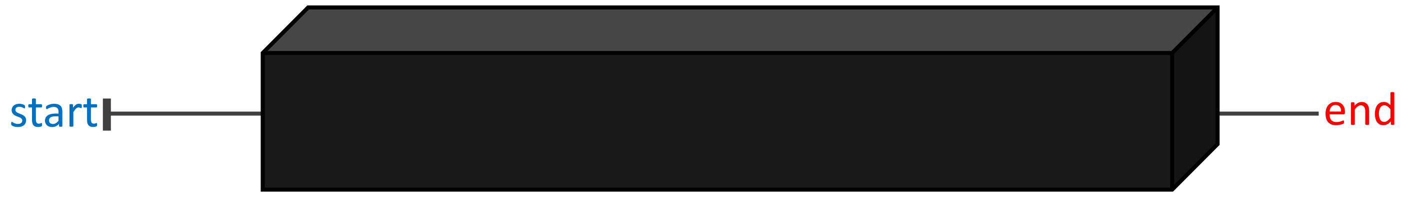
Context

- Alpyne makes use of the RL experiment – added in AnyLogic 8.7
- Next few minutes will explain what it is – and RL more generally – drawing from familiar concepts
- A typical experiment will consist of:
  - Inputs (parameters)
  - A model (logic)
  - Outputs (e.g., datasets)



Context

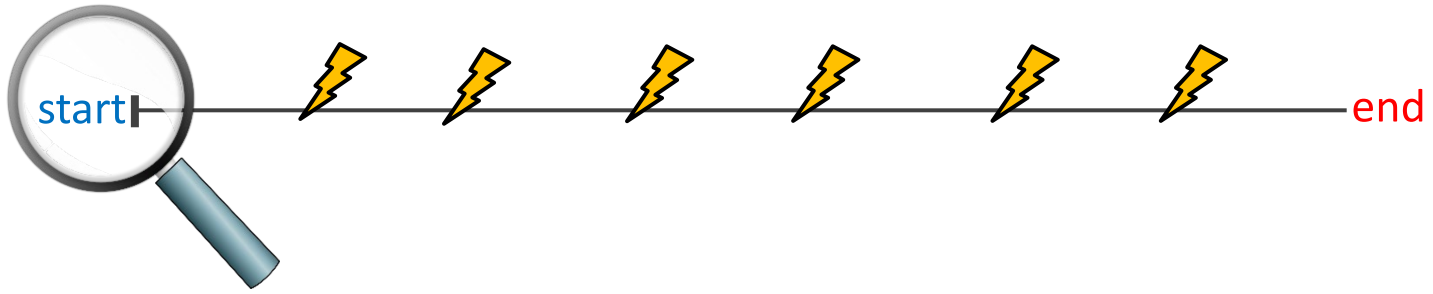
- A simulation model that is used for reinforcement learning delegates some of the decisions (actions) that are being taken throughout its execution to the AI agent (RL algorithms)



Triggered "steps" via the takeAction function

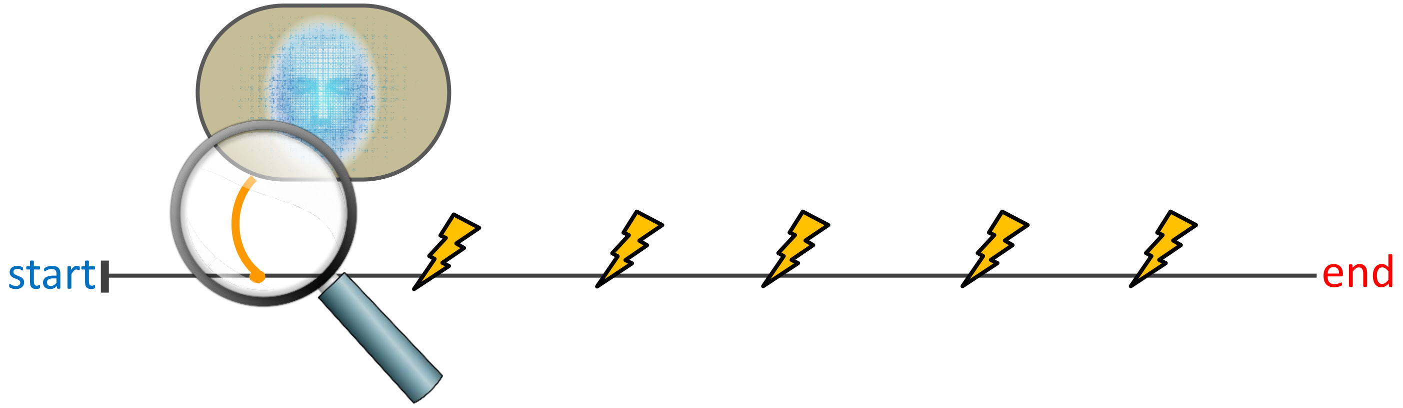
Context

- At the start of each simulation run ("episode") a **configuration** is set as the initial state



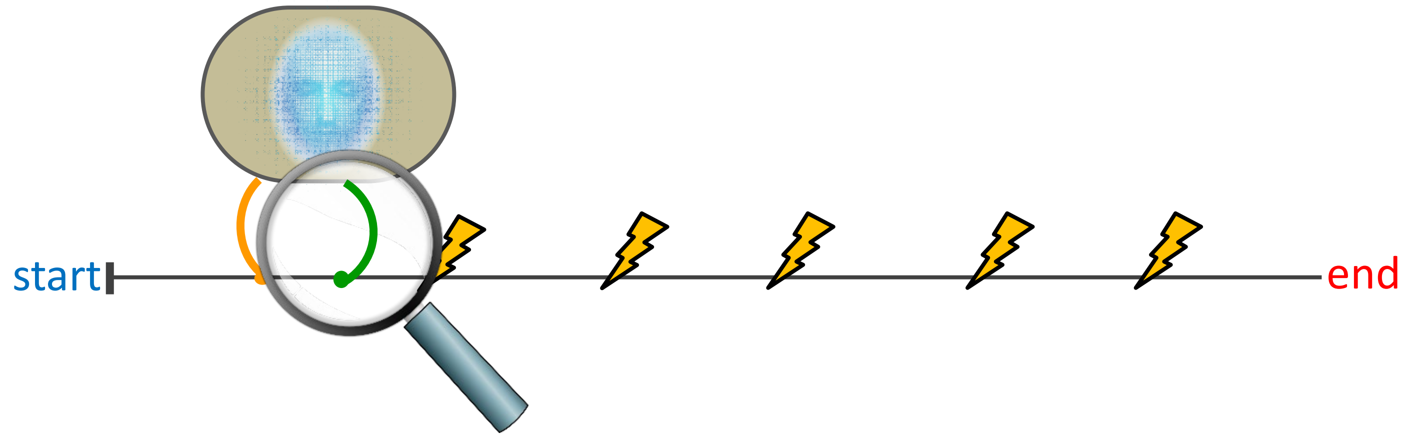
Context

- At the decision point (step event) the model first pauses itself. This allows its current state to be sent to the RL agent (**“observations”**)



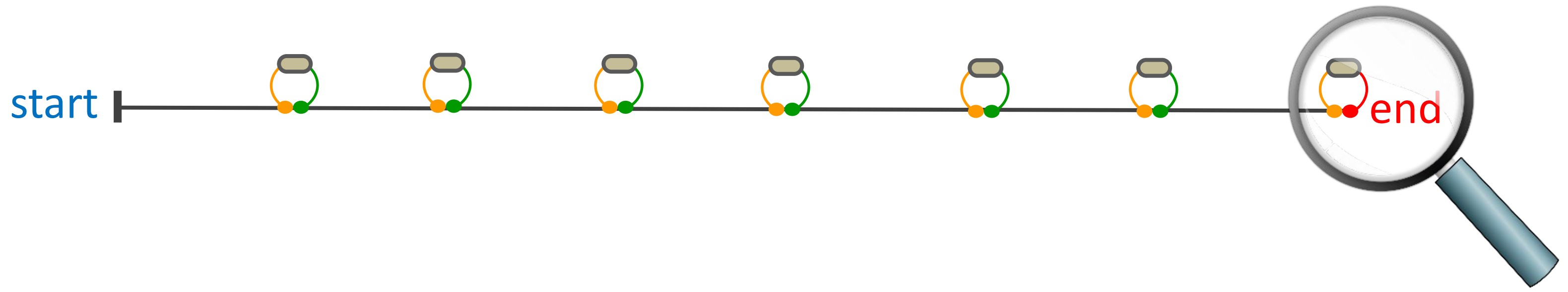
Context

- The RL agent responds with a chosen **action** for the model to take. The model is then allowed to continue…



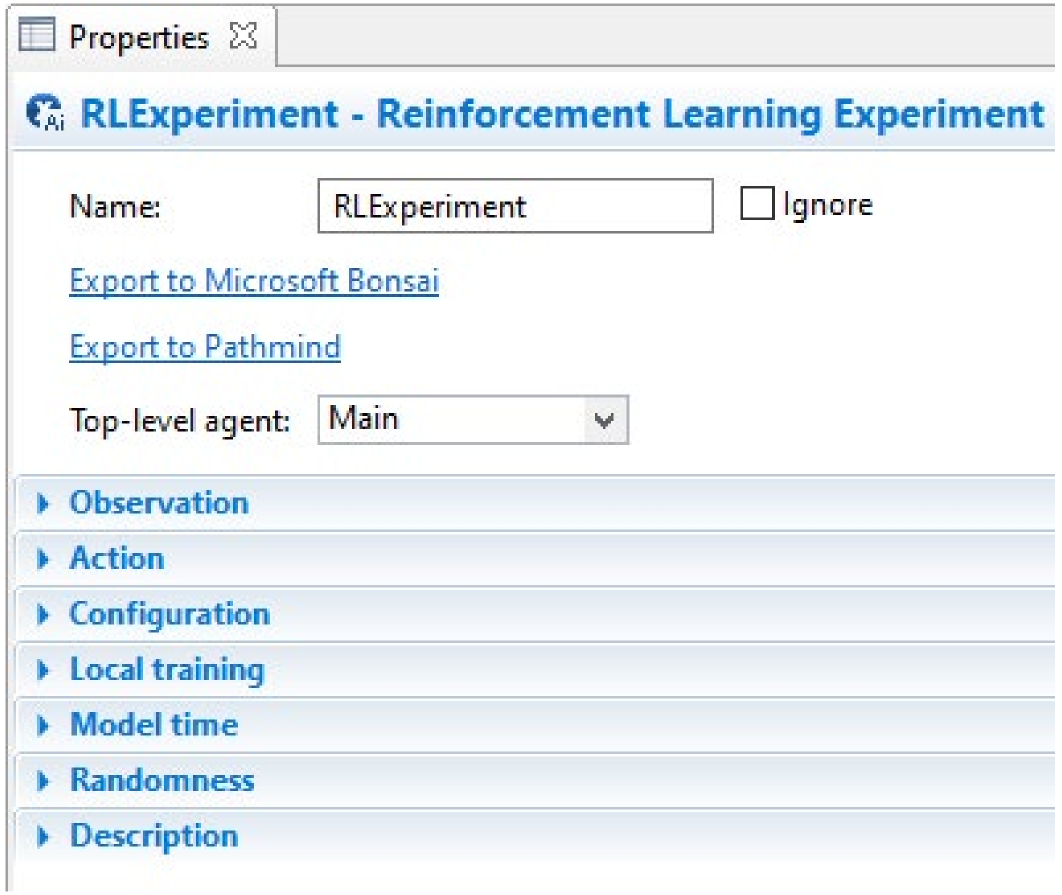
Context

- This process continues until a **terminal** or **“done” condition** is met



Context

- RL Experiment added in AnyLogic 8.7.0
- No graphical interface; only has properties for you to implement the required sections
- Contains sections to define data fields that the Configuration, Observation, and Action are composed of
- Code fields provide an area for you to fill out the objects or apply them to your model
- Also contain a condition-based stopping field, in additional to the usual start/stop fields and choice for RNG seed


Properties

RLExperiment - Reinforcement Learning Experiment

Name:	RLExperiment	☐ Ignore

Export to Microsoft Bonsai

Export to Pathmind

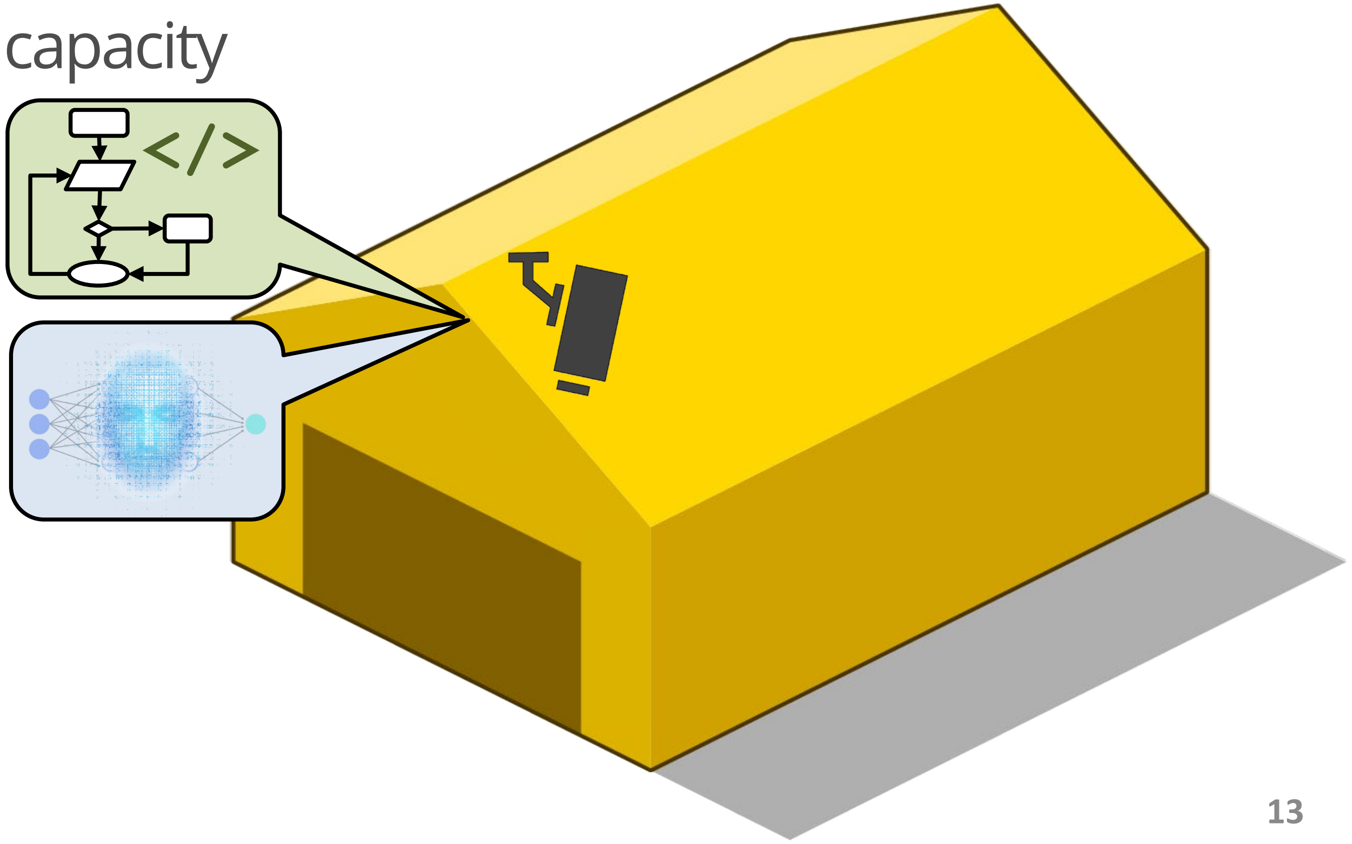
Top-level agent:	Main

Explanation through example

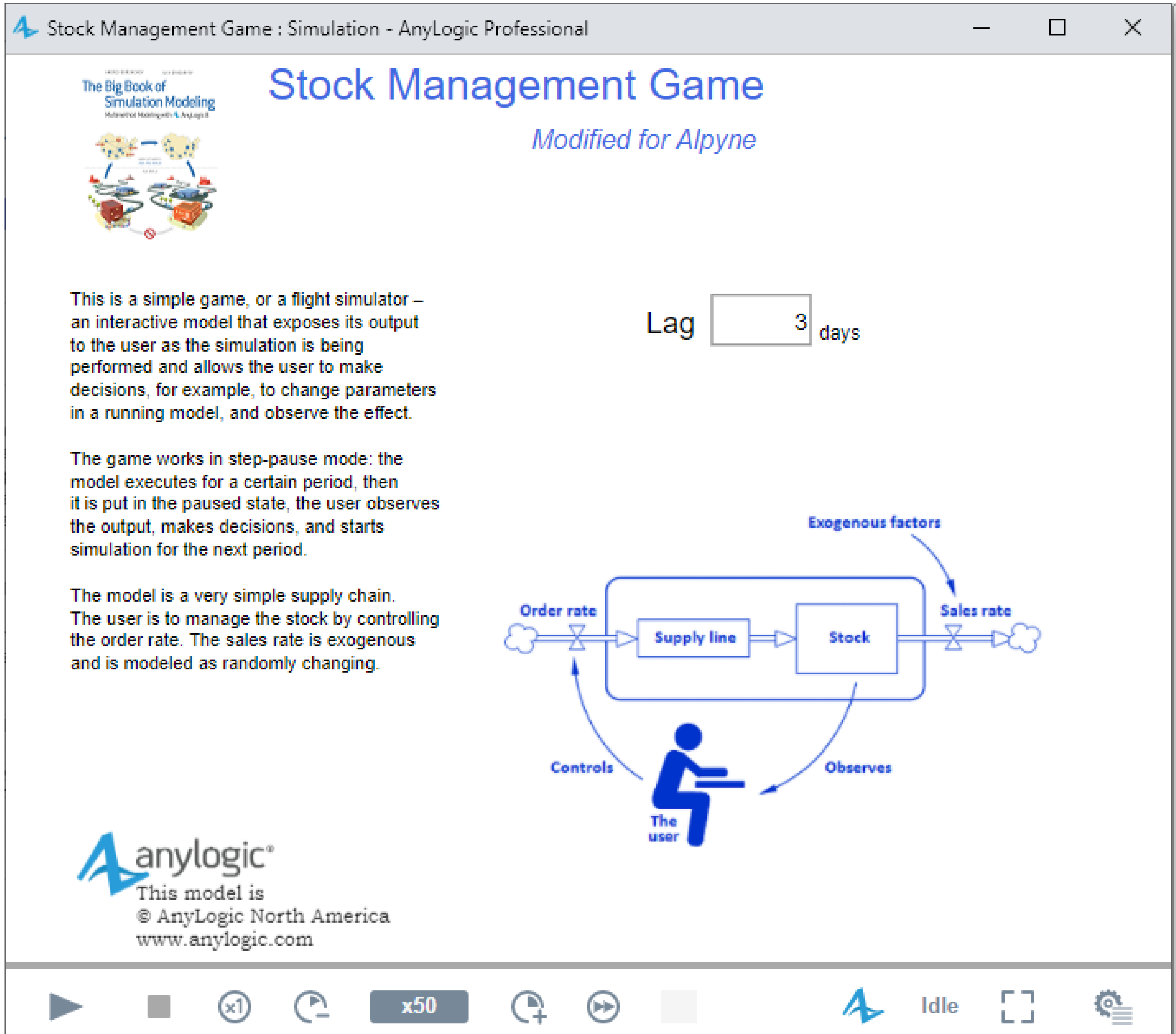
- Simple warehouse that holds a limited amount of product (“stock”)
- Depleted at a rate, controlled by exogenous factors
- Replenished by a daily rate of new stock (optionally changeable once a month)
- Overall goal is to maintain ~50% capacity

Ways to accomplish this:

1. Hire a human worker
2. Algorithmic inventory policy
3. Reinforcement learning policy



Example model



Stock Management Game : Simulation - AnyLogic Professional

The Big Book of Simulation Modeling

Stock Management Game

Modified for Alpyne

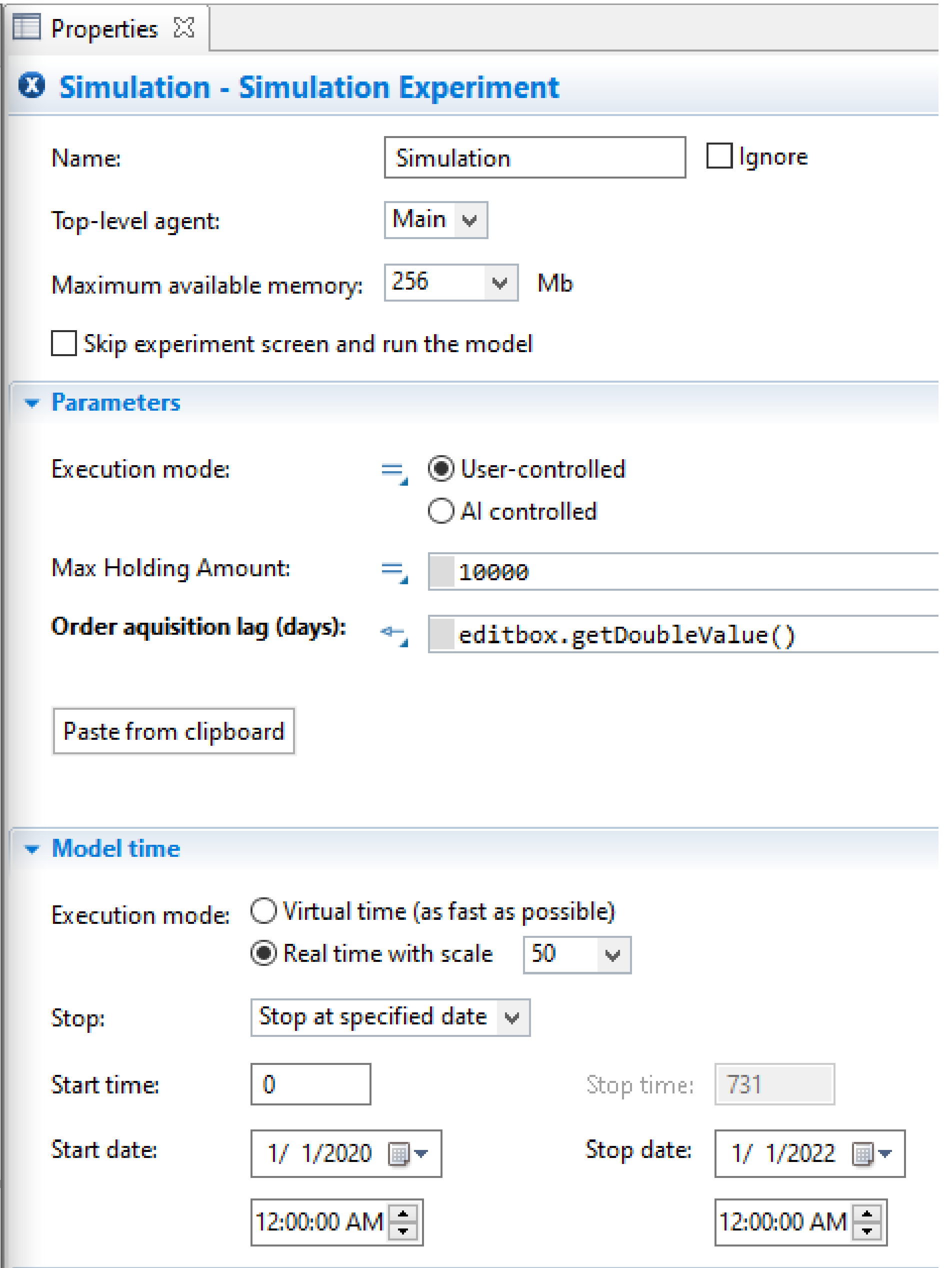
Lag 3 days

This is a simple game, or a flight simulator – an interactive model that exposes its output to the user as the simulation is being performed and allows the user to make decisions, for example, to change parameters in a running model, and observe the effect.

The game works in step-pause mode: the model executes for a certain period, then it is put in the paused state, the user observes the output, makes decisions, and starts simulation for the next period.

The model is a very simple supply chain. The user is to manage the stock by controlling the order rate. The sales rate is exogenous and is modeled as randomly changing.

anylogic
This model is
© AnyLogic North America
www.anylogic.com



Properties

Simulation - Simulation Experiment

Name: Simulation ☐ Ignore

Top-level agent: Main

Maximum available memory: 256 Mb

☐ Skip experiment screen and run the model

Parameters

Execution mode: [✓ User-controlled] AI controlled

Max Holding Amount: `10000`

Order aquisition lag (days): `editbox.getDoubleValue()`

Paste from clipboard

Model time

Execution mode: Virtual time (as fast as possible) [✓ Real time with scale] 50

Stop: Stop at specified date

Start time: 0 Stop time: 731

Start date: 1/ 1/2020 Stop date: 1/ 1/2022

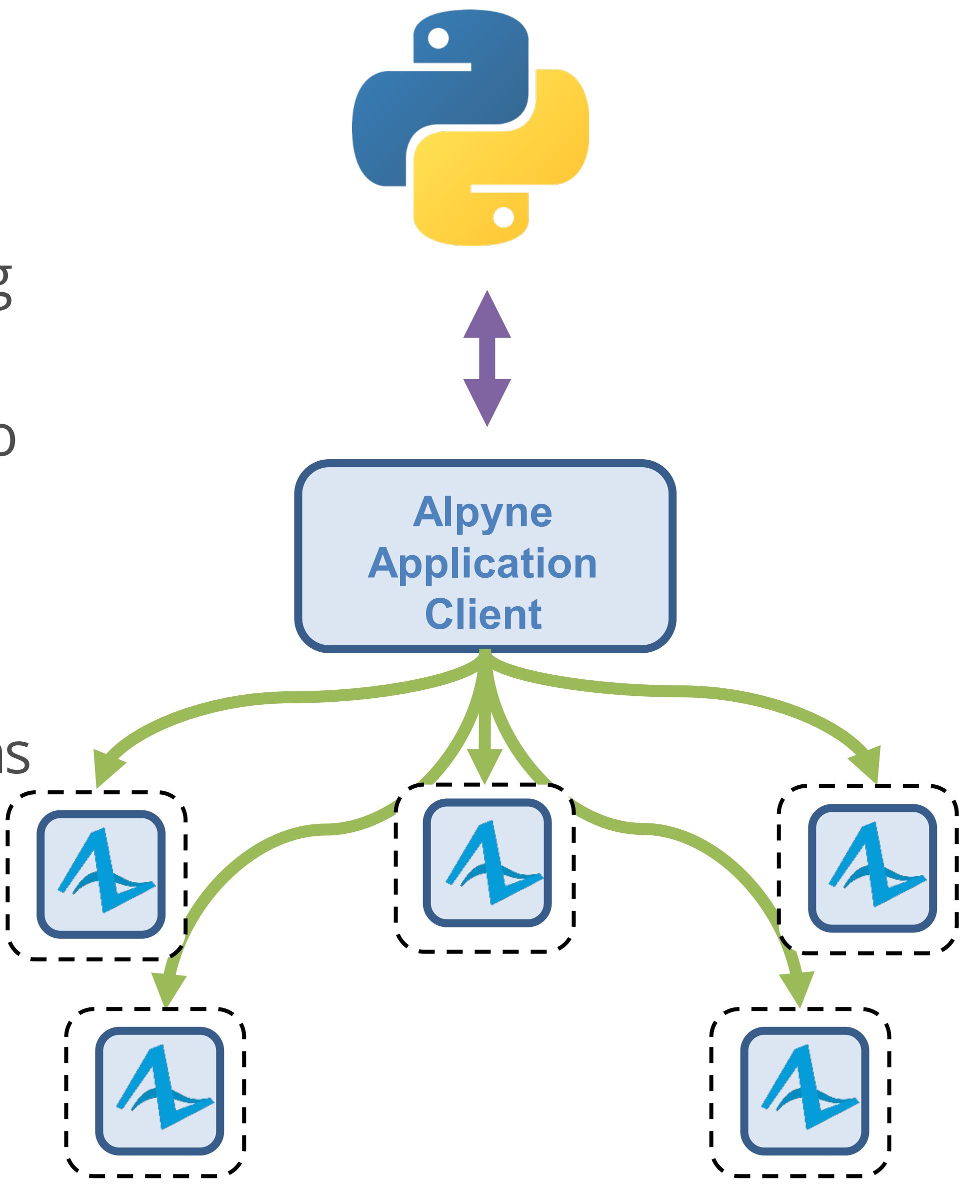
12:00:00 AM 12:00:00 AM

High-level Alpyne Workflow

1. Export your model from the RL experiment (.zip)
2. In Python, create an `AlpyneClient` object passing the exported location
3. Build Configuration objects which are then used to create new model runs
4. Interact with the run through the provided methods until stop conditions are met
5. Each run can be reset and reused for as many runs as desired

If some of this sounds familiar to the Cloud API

– this is intentional!



Quick mentions

- As there are many concepts in this library, I'll build up to the true capabilities through four Python scripts
    1. Single run with pre-defined actions (no RL)
    2. Parallel runs with pre-defined actions (no RL) and manual implementation
    3. Parallel runs with pre-defined actions (no RL) and automated implementation
    4. Parallel runs with RL
    - The first 3 are not considered "practical" or intended use-cases, they are meant for demoing purposes
- Final warning: the rest of this section is in a Python environment (and thus is code heavy)
    - As previously stated, the use of this library is centered in an implementation environment
    - The following is separate from AnyLogic and what I will show is the RL side of things

Additional remarks…

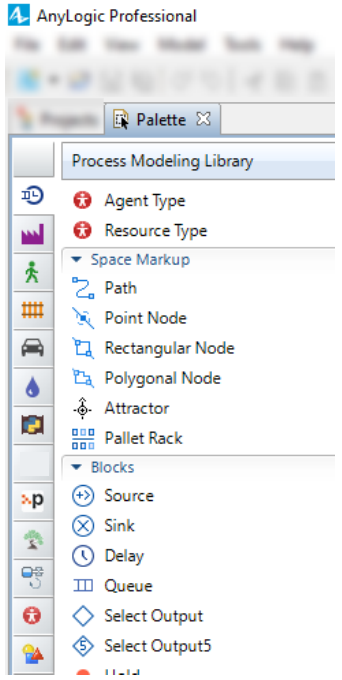
- Allows RL-ready models to be trained in a manually defined Python environment
- Intended for technical users who are familiar Python and RL libraries
- Interaction is only through the RL experiment; meant for RL training and not a replacement for other AnyLogic experiments
- Execution is on a local environment; limited by constraints of machine's hardware
  - For scalable solutions, wait for the Cloud's interactive API
  - For automated RL solutions, consult the Pathmind or Bonsai platforms

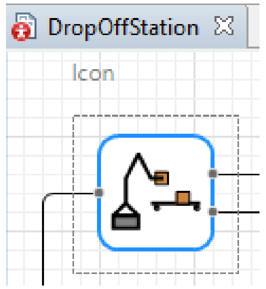
What exactly is Pypeline?

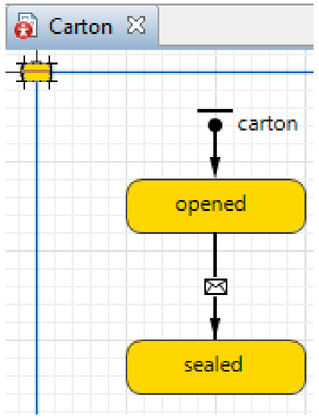
- A custom addon library for AnyLogic for connecting to and communicating with a local installation of Python from within a running AnyLogic model

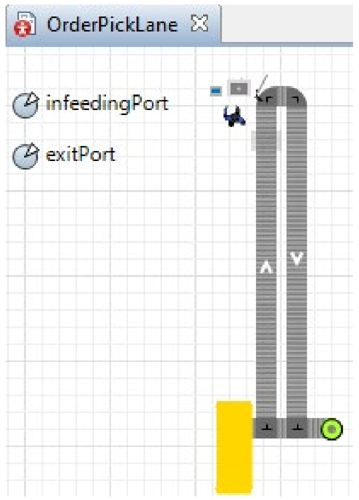
What exactly is Pypeline?

- A custom addon library for AnyLogic for connecting to and communicating with a local installation of Python from within a running AnyLogic model
- Assets from an AnyLogic model can be bundled into a library
  - Libraries can be added into the development environment for future use across other models
  - Exportable as .jar files, allowing distribution to others



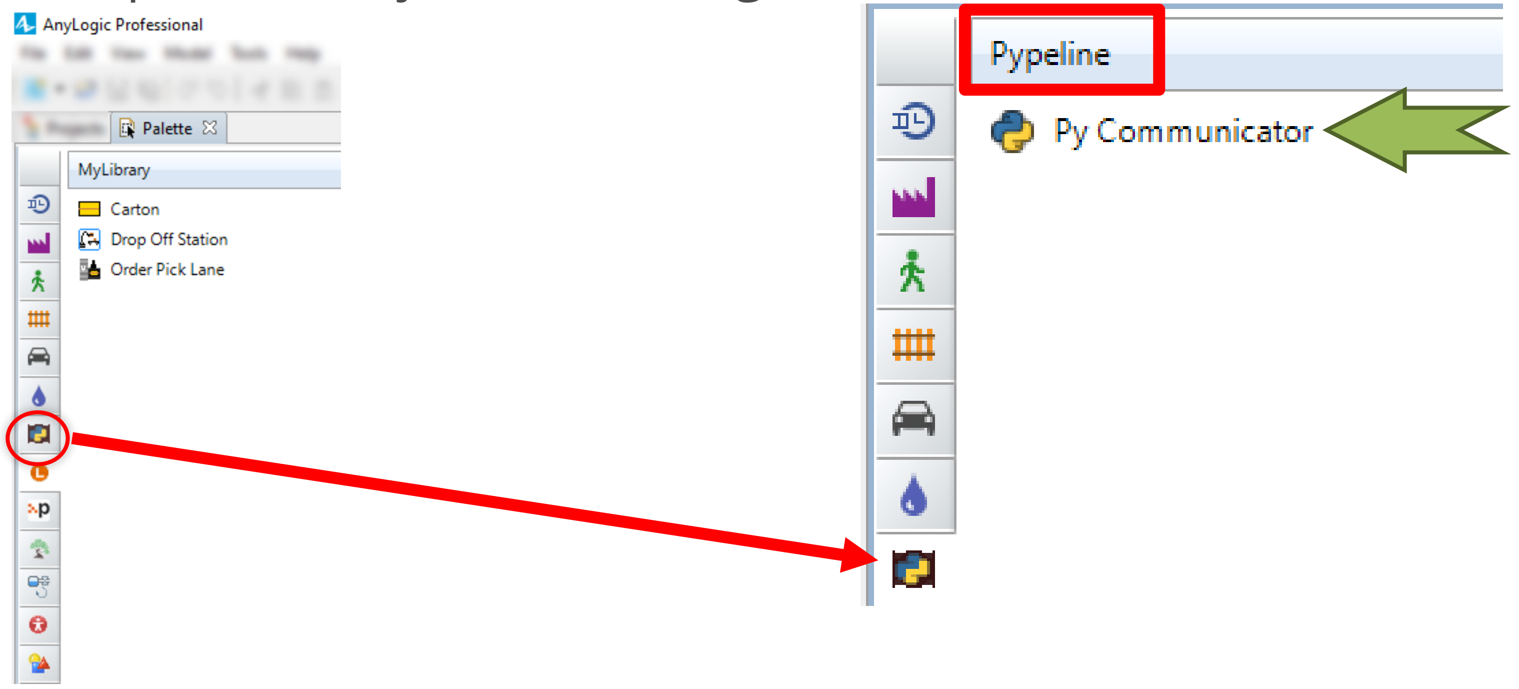






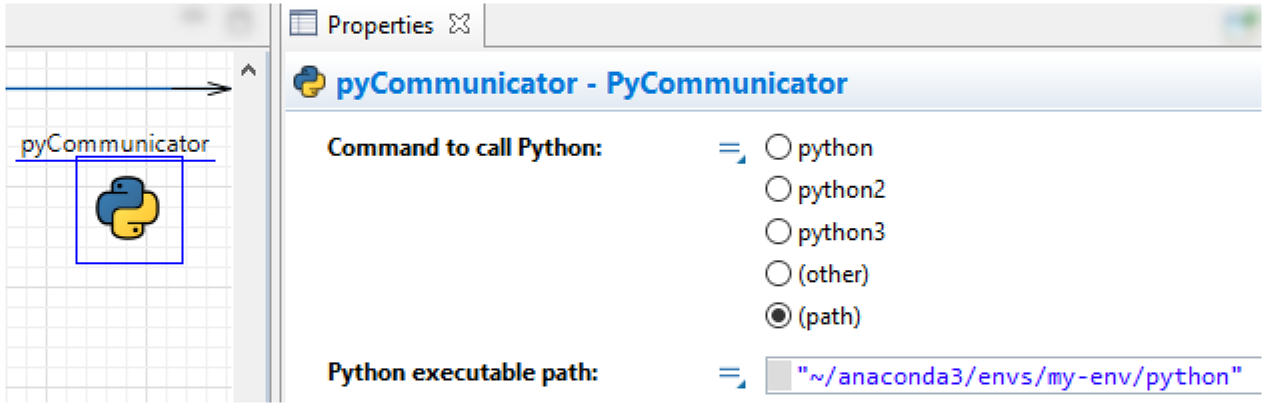
What exactly is Pypeline?

- A custom addon library for AnyLogic for connecting to and communicating with a local installation of Python from within a running AnyLogic model
- Assets from an AnyLogic model can be bundled into a library
  - Libraries can be added into the development environment for future use across other models
  - Exportable as .jar files, allowing distribution to others



What exactly is Pypeline?

- A custom addon library for AnyLogic for connecting to and communicating with a local installation of Python from within a running AnyLogic model
- Pypeline does not provide any version of Python
- It uses the version of Python that's installed on the user's machine
- Options are available to choose how Python gets started
  - Compatible with any install type (official installer, Anaconda, etc.) and virtual environments
  - Chosen from the properties of the "Py Communicator object"



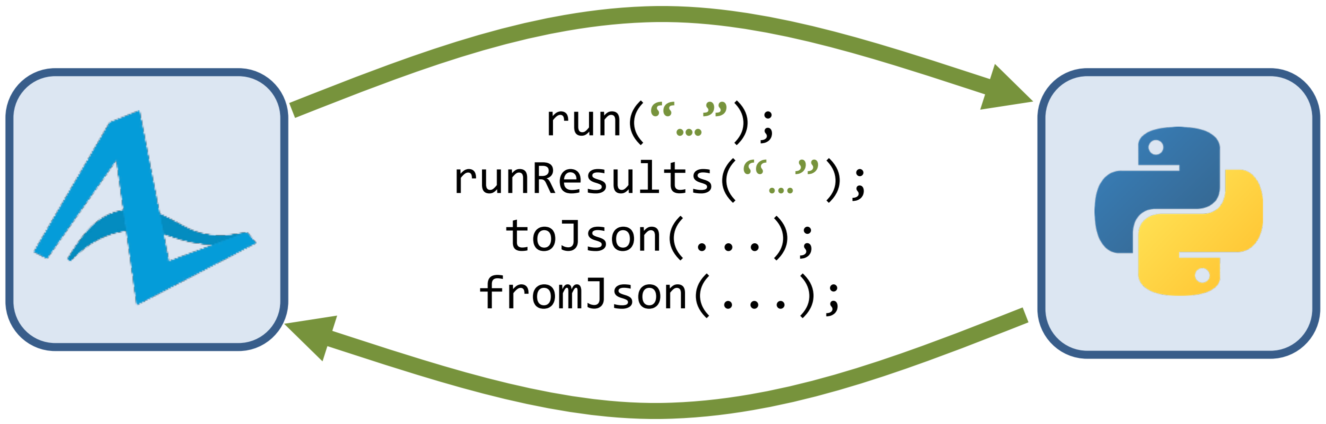
pyCommunicator - PyCommunicator

Command to call Python: ○ python ○ python2 ○ python3 ○ (other) [✓ (path)]

Python executable path: "~/anaconda3/envs/my-env/python"

What exactly is Pypeline?

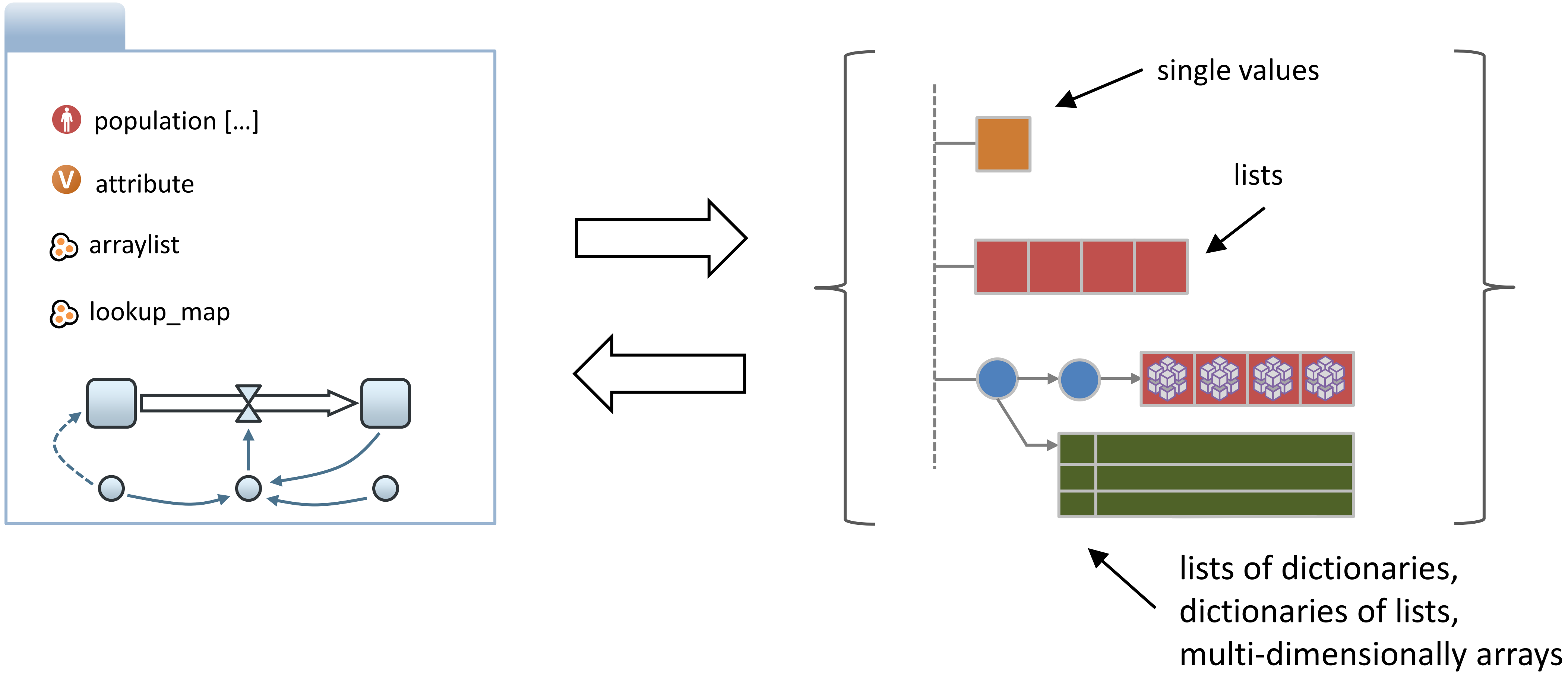
- A custom addon library for AnyLogic for connecting to and **communicating with a local installation of Python** from within a running AnyLogic model
- Provided functions within the "Py Communicator" object allow you to:
  - Send arbitrary Python code (assigning variables, importing libraries, etc.)
  - Retrieving values within the interactive Python environment
  - Easily convert data structures, including agents and populations, to a format both Java and Python can parse (i.e., JSON)



```
run("…");
runResults("…");
toJson(...);
fromJson(...);
```

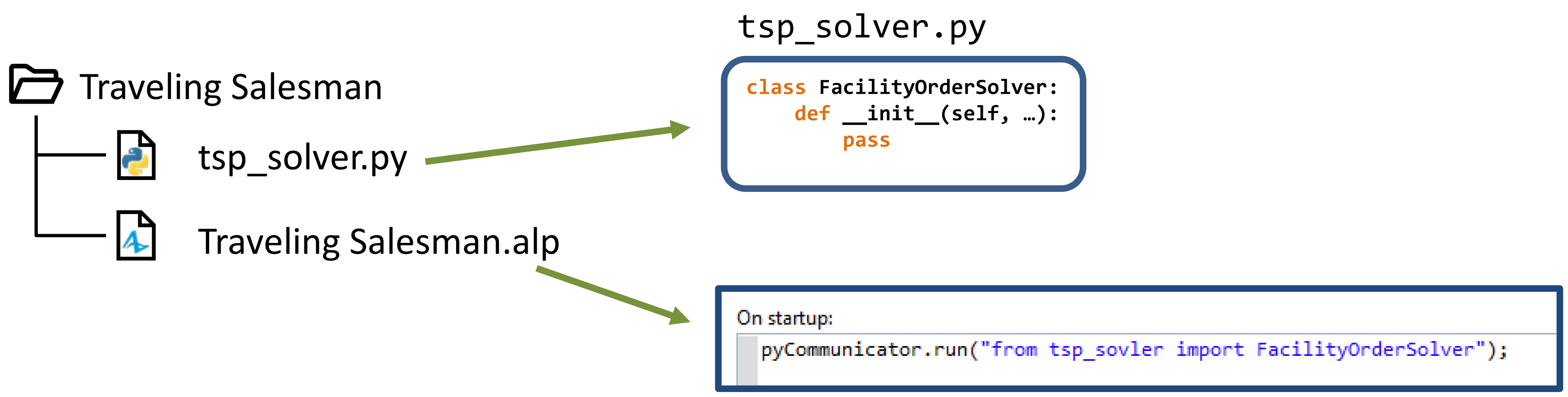
Passing/receiving data

- JSON is used as the intermediary language for data passing
- Custom functions allows complex data structures to be passed between a model and Python



Quick note about recommended usage

- The concept of local imports applies to Python files in your model directory



tsp_solver.py

```
class FacilityOrderSolver:
    def __init__(self, …):
        pass
```

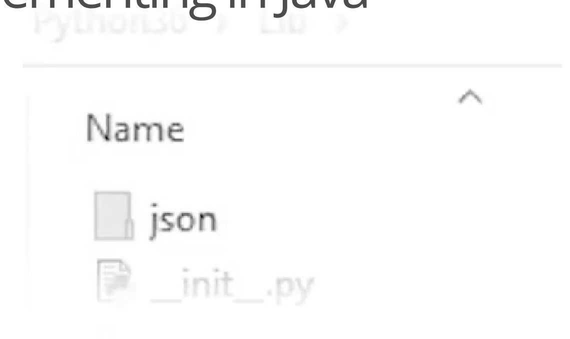
On startup:

```
pyCommunicator.run("from tsp_sovler import FacilityOrderSolver");
```

- With this approach:
  - Less prone to syntax errors
  - Easier to debug and test Python code

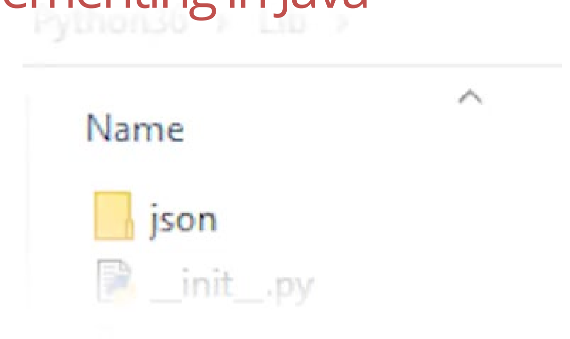
Use-cases

- Specific reasons dependent on your use for Python
- In general, some example use-cases include:
  1. Using existing Python scripts (e.g., custom algorithms) without reimplementing in Java
  2. Enhancing models with popular data visualization libraries
  3. Testing machine learning models in a simulation model



Use-cases

- Specific reasons dependent on your use for Python
- In general, some example use-cases include:
  1. Using existing Python scripts (e.g., custom algorithms) without reimplementing in Java



Python36 > Lib >

Name

json

__init__.py

  2. Enhancing models with popular data visualization libraries

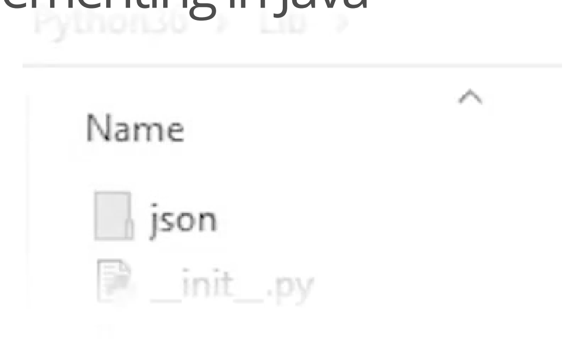
matplotlib geoplotlib plotly

  3. Testing machine learning models in a simulation model

TensorFlow PyTorch

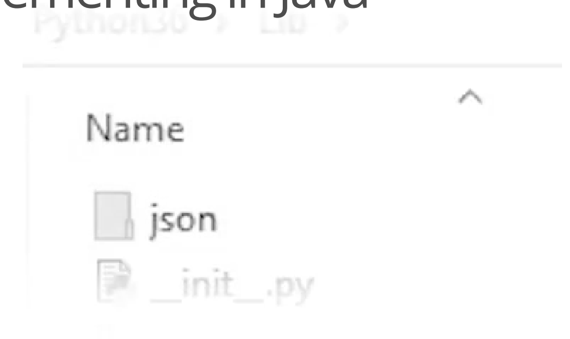
Use-cases

- Specific reasons dependent on your use for Python
- In general, some example use-cases include:
  1. Using existing Python scripts (e.g., custom algorithms) without reimplementing in Java
  2. Enhancing models with popular data visualization libraries
  3. Testing machine learning models in a simulation model



Use-cases

- Specific reasons dependent on your use for Python
- In general, some example use-cases include:
  1. Using existing Python scripts (e.g., custom algorithms) without reimplementing in Java
  2. Enhancing models with popular data visualization libraries
  3. Testing machine learning models in a simulation model



Additional remarks

- Python, through libraries, is **not** a replacement for Java coding in AL models
  - Native solutions will [almost] always be faster (crucial in high performance models)
- Usage of Pypeline shouldn't change how you build models in the AnyLogic GUI; it's purpose is a supplemental library
- You cannot use Pypeline to train reinforcement learning agents
  - For that: Pathmind, Bonsai, Alpyne, and (soon) the AnyLogic Cloud API

Resources

Alpyne
(public beta)

- https://git.io/alpyne
  - Includes:
    - *README with link to install instructions, walkthrough, API docs*
    - *Example models*
    - *Source for Python library*

Pypeline

- https://git.io/al_py
  - Includes:
    - *Library jar file*
    - *Example models*
    - *Wiki (instructions, usage guide, troubleshooting)*
    - *Complete source*

thank you!