Policy Enforcement Assessment: Stage 1

Definitions

A **policy** is a group of definitions of what traffic is allowed or forbidden in a network.

Each policy has the following fields:

- `name` - a textual name, consisting of at most 32 alphanumeric characters and underscores
- `description` - free text
- `type` - either of the strings `"Arupa"` or `"Frisco"`

An Arupa policy must not have the same `name` as another Arupa policy. Multiple Frisco policies may have the same `name`.

Requirements

Given the provided skeleton, fill in methods of the `PolicyAPI` class to implement a programmatic API for managing policies. Data should be stored in memory and does not need to be saved to a persistent data store. All methods should take arguments and return values which are JSON strings. Error conditions should raise exceptions.

- The `create_policy()` method should take as its only argument a JSON string containing an object whose keys are the fields of the policy, and should return a JSON string containing something which can be used to identify the policy in the future.
- The `read_policy()`, `update_policy()`, and `delete_policy()` methods should take as their first argument the same object identifiers which the `create_policy()` method returns.
- The `read_policy()` method should return a JSON string containing an object which has keys corresponding to each of the policy's fields.
- The `update_policy()` method should take as its second argument the same input which is passed to the `create_policy()` method, supporting the same set of fields.
- The `list_policies()` method should return a JSON string containing an array of objects, each of which is of the same or similar form to that returned by `read_policy()`.

Other Instructions

- Base your solution on the skeleton provided. Don't change the signatures of the class or methods or the names of the policy fields. You may either edit the skeleton file itself or copy it aside to a new solution file.
- There are a couple of baseline tests provided to help you bootstrap. Make sure they pass before you improve anything else. The tests assume that the solution is implemented in a file named `stage1.py` on the python path.
- The problem specification may be intentionally vague or incomplete. If you encounter something which is under-specified, make reasonable assumptions about the correct behavior. If you're unsure which assumption to make, pick the one which requires the least amount of work to implement. **It's important that you document all assumptions you make and communicate them clearly in your submission.**