Rego

Rego

- Rego is a declarative language for defining policies
- In Rego, policies are expressed as predicates over structured data
    - This structured data is represented using JavaScript Object Notation (JSON)
- This training covers the basic Rego features needed to implement provenance policies

Rego Syntax

- In Rego, policies are defined in the form
  <head> { <expr 1; expr 2; …; expr n>}
- Head refers to the name of the policy
- The expressions 1 to n are evaluated to determine if the policy is true or false
- Policy1 is true if all of the expressions 1 to n are true

```
policy1 {
        expression 1
        expression 2
        …
        expression n
}
```

Rego Syntax: Expressions

- Rego offers a variety of operators seen in other languages
  - Arithmetic (+, -, etc.), assignment (:=), equality (==)
- Note that variables are immutable (cannot be changed after their initial assignment)

```
final_policy {
        # Comments appear after a pound symbol
        x := 1 + 1  # Assign 2 to variable x
        x == 2 #  Returns true if x is equal to 2
}
```

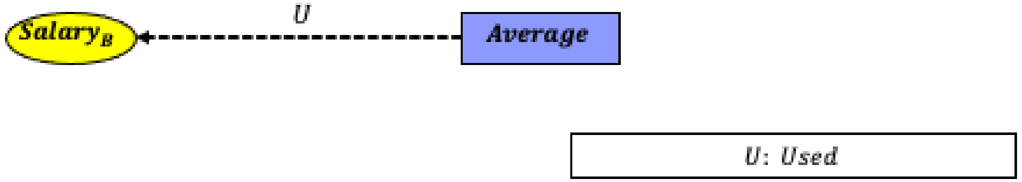
Rego Syntax: Policy Arguments

- In Rego, arguments may be passed to policies
    - The syntax is similar to functions in many other languages

```
final_policy {
        isZero(0)
        not isZero(1)
}

isZero(x){  # Returns true if x is 0
        x == 0
}
```

Rego Syntax: Input

- Uses JavaScript Object Notation (JSON) format
- Provenance graphs can be defined as a set of vertices and edges
    - Each vertex contains a name and type
    - Each edge has a relation type, source, and destination



```
{
  "vertices":[
    {
      "name":"Average",
      "type":"activity"
    },
    {
      "name":"SalaryB",
      "type": "dataEntity"
    }
  ],
  "edges":[
    {
      "relation":"used",
      "source": "Average",
      "destination": "SalaryB"
    }
  ]
}
```

Rego Syntax: Input

- JSON data consists of a name and value
- Two key types: objects and arrays
  - {} defines an object, use . to access fields
  - [] defines an array, use [] to access elements
- The lists of vertices and edges are declared within an implicit 'input' object
  - The list of vertices can be accessed using the input keyword followed by a dot and then the vertices keyword
  - A specific vertex can be accessed by providing an index in this list. As shown on the right, we access the name of the first vertex using input.vertices[0].name

```
{
    "vertices":[
        {
            "name":"Average",
            "type":"activity"
        },
        {
            "name":"SalaryB",
            "type": "dataEntity"
        }
    ],
    "edges":[
        {
            "relation":"used",
            "source": "Average",
            "destination": "SalaryB"
        }
    ]
}
```

Negation policy

- Asserts that an expression is not true
- A policy name may appear in a expression
    - These examples are equivalent

```
final_policy {
        x := 1 + 1
        not x == 2
}
```

```
final_policy {
        not xEquals2
}

xEquals2{
        x := 1 + 1
        x == 2
}
```

Conjunction Policy

- Also called AND
- Asserts that both expressions e_1 and expressions e_2 are true

```
final_policy {
        e1
        e2
}
```

Disjunction Policy

- Also called OR
- In Rego, implemented by creating two policies with the same name
  - If either policy is true (or both are true) then the policy evaluates to true
- In the example, final_policy evaluates to true if e1 or e2 are true

```
final_policy {
        e1
}

final_policy {
        e2
}
```

Existential Policy

- Asserts that an expression is true for some variable x. Variable x can be used in the expression

```
final_policy {
        some x
        # Use x in the expression, for example:
         input.vertices[x].name == “average”
}
# In English: there exists some vertex x named “average”
```

Universal Policy

- Asserts that an expression is true for every variable x. Variable x can be used in the expression
- Not explicitly supported by Rego

```
final_policy {
        not averageExists # For universal, create an existential, then negate it
} # In English: for all vertices x, x is not named “average”

averageExists {
        some x
        input.vertices[x].name == “average”
} # In English: there exists some vertex x named “average”
```

Edge policy

- Asserts that there is an edge between two vertices with a certain label or relation
- A label may any of the following: wasAttributedTo, wasDerivedFrom, wasGeneratedBy, used, actedOnBehalfOf, wasAssociatedWith, or wasInformedBy

```
final_policy {
        some x
        # Use x to define policy below, for example:
         input.edges[x].relation == “used”
         input.edges[x].source == “Average”
         input.edges[x].destination == “SalaryB”
}
```

Combining Existential and Universal

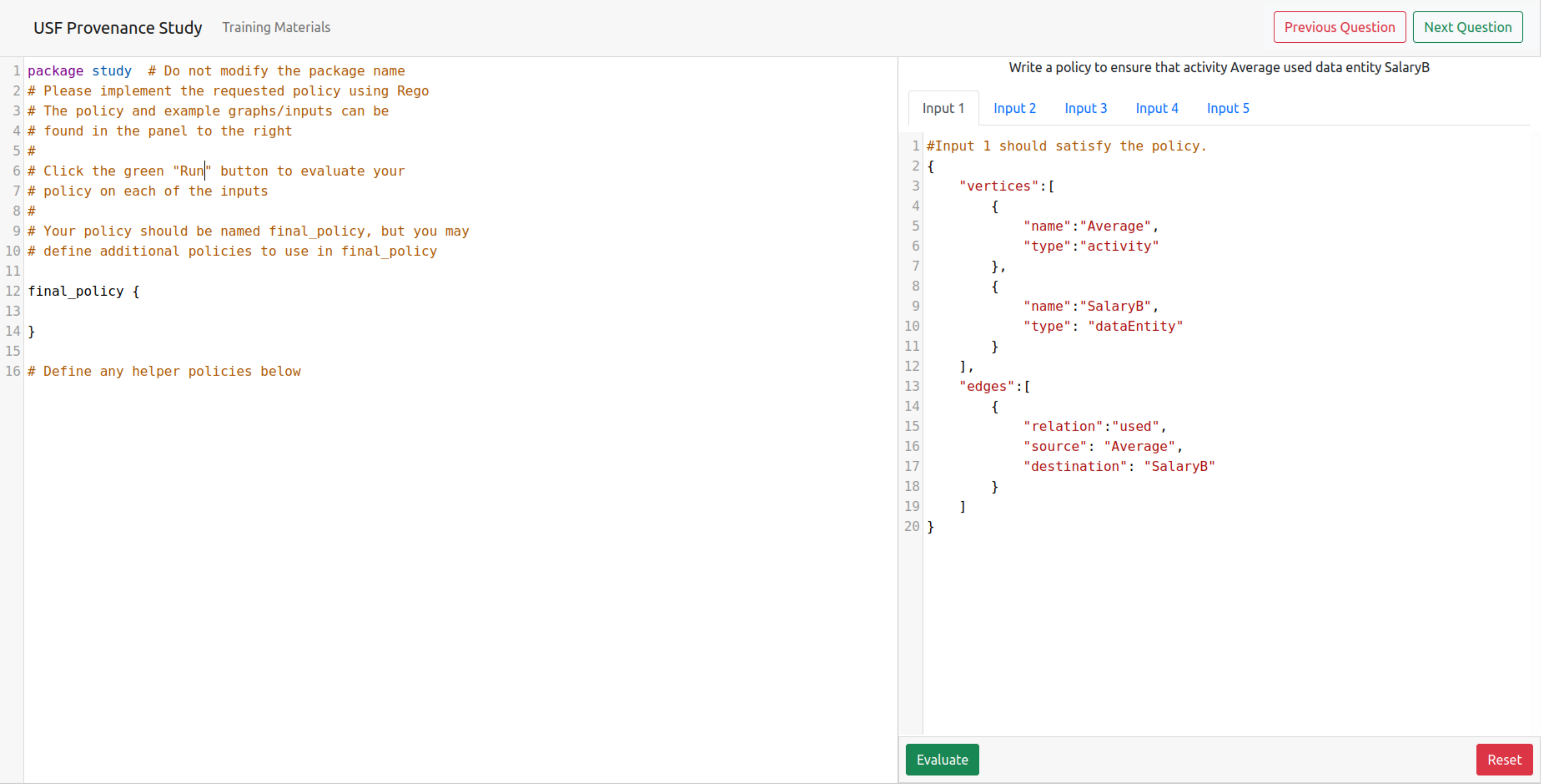
- To write a policy that combines the existential and universal policies in Rego, the policy argument feature is required.
- The following slide has an example

Combining Existential and Universal

- For all node agents, there exists some outgoing edge.

```
final_policy {
        not nodeNoEdge  # True if all agents have an outgoing edge
}
nodeNoEdge{   # True if there exists a nodeAgent without an outgoing edge
        some x
        input.vertices[x].type == “nodeAgent”
        not hasEdges(x)
}
hasEdges(x){     # True if vertex x has an outgoing edge
        some y
        input.vertices[x].name == input.edges[y].source
}
```

Provenance policy examples



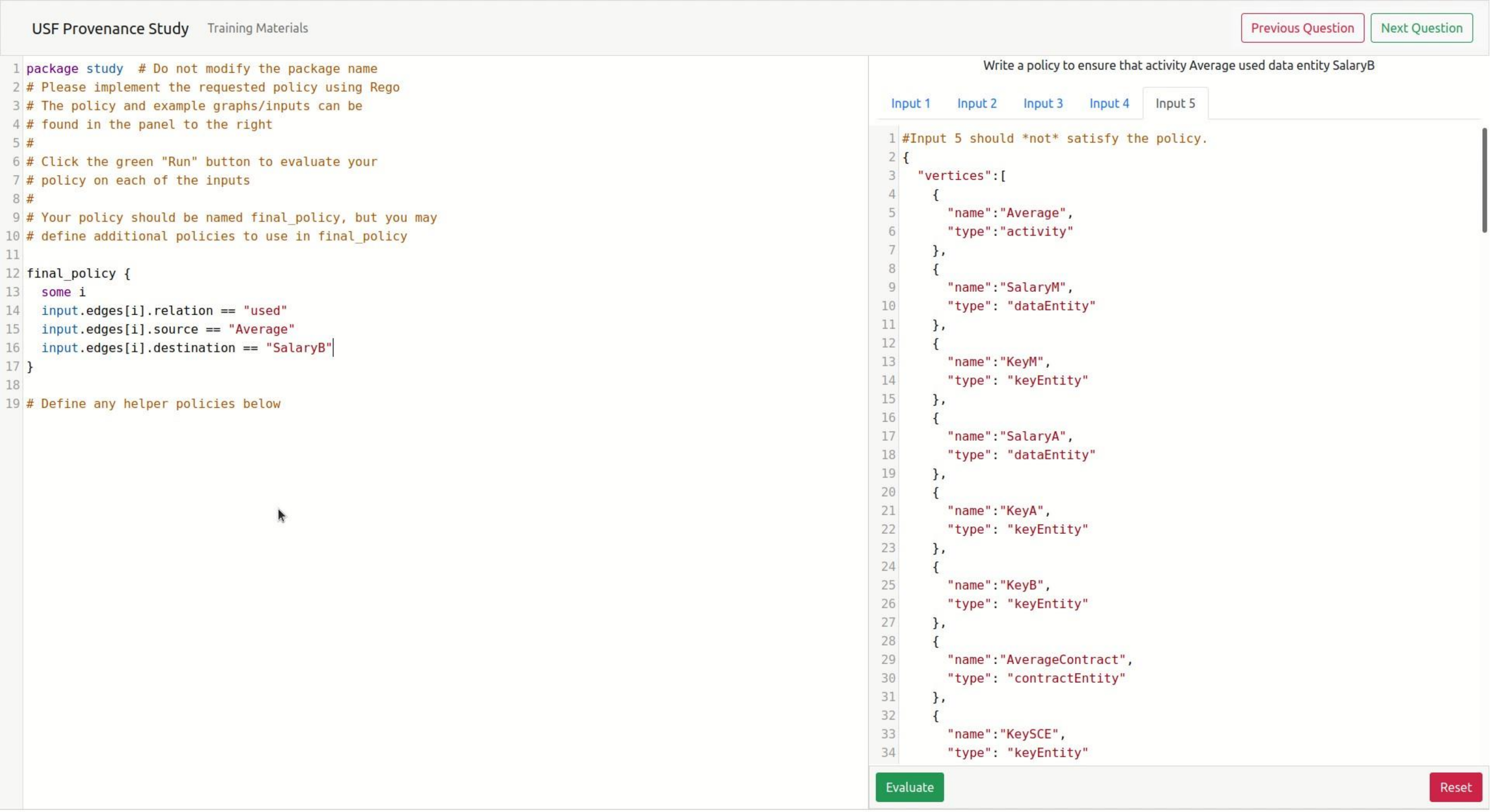
```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {

}

# Define any helper policies below
```

Write a policy to ensure that activity Average used data entity SalaryB

```
#Input 1 should satisfy the policy.
{
    "vertices":[
        {
            "name":"Average",
            "type":"activity"
        },
        {
            "name":"SalaryB",
            "type": "dataEntity"
        }
    ],
    "edges":[
        {
            "relation":"used",
            "source": "Average",
            "destination": "SalaryB"
        }
    ]
}
```

Provenance policy examples

- Write a policy to ensure that activity *Average* used data entity *SalaryB*



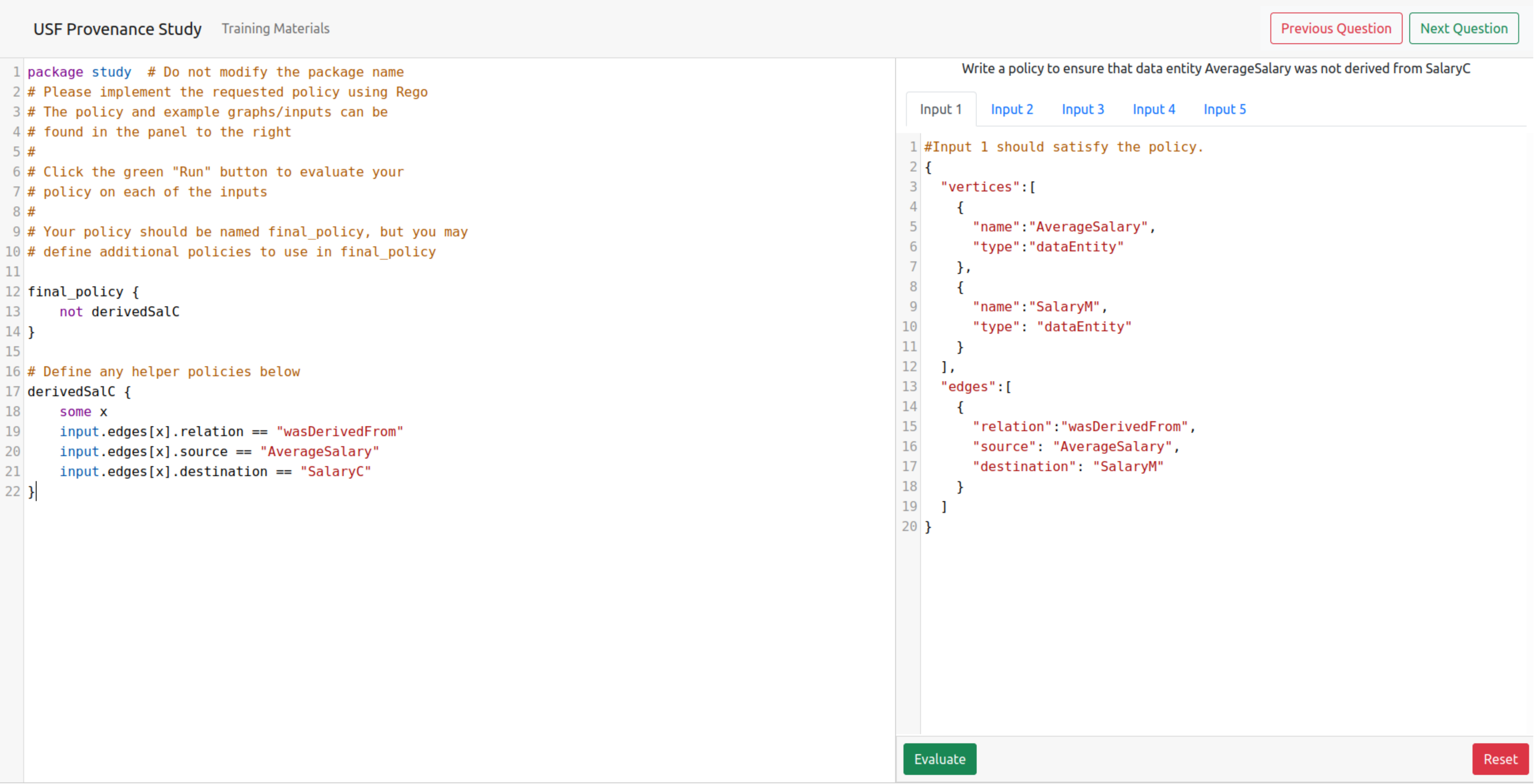
```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
  some i
  input.edges[i].relation == "used"
  input.edges[i].source == "Average"
  input.edges[i].destination == "SalaryB"
}

# Define any helper policies below
```

Write a policy to ensure that activity Average used data entity SalaryB

```
#Input 5 should *not* satisfy the policy.
{
  "vertices":[
    {
      "name":"Average",
      "type":"activity"
    },
    {
      "name":"SalaryM",
      "type": "dataEntity"
    },
    {
      "name":"KeyM",
      "type": "keyEntity"
    },
    {
      "name":"SalaryA",
      "type": "dataEntity"
    },
    {
      "name":"KeyA",
      "type": "keyEntity"
    },
    {
      "name":"KeyB",
      "type": "keyEntity"
    },
    {
      "name":"AverageContract",
      "type": "contractEntity"
    },
    {
      "name":"KeySCE",
      "type": "keyEntity"
```

Provenance policy examples

- Write a policy to ensure that data entity *AverageSalary* was not derived from *SalaryC*



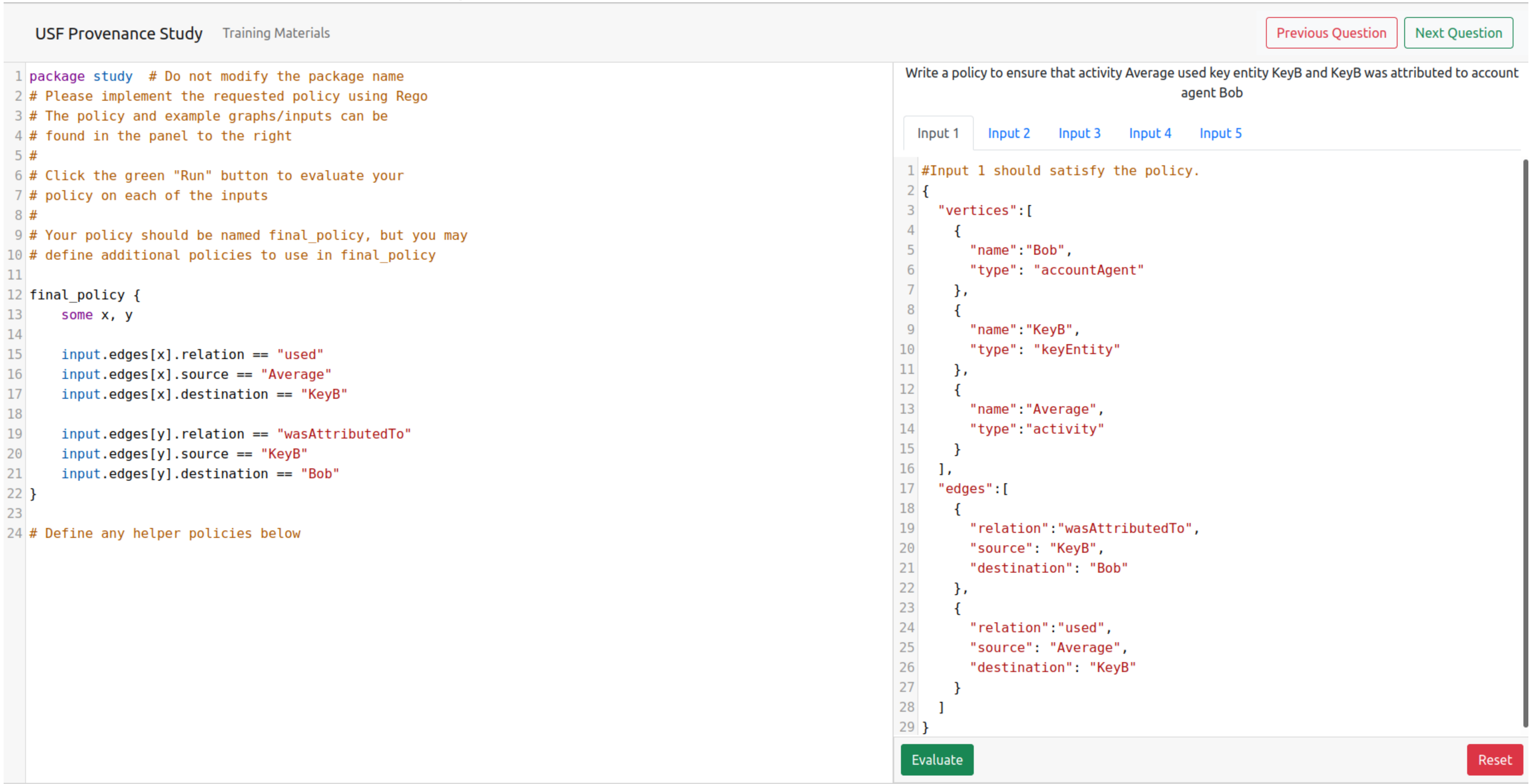
```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
    not derivedSalC
}

# Define any helper policies below
derivedSalC {
    some x
    input.edges[x].relation == "wasDerivedFrom"
    input.edges[x].source == "AverageSalary"
    input.edges[x].destination == "SalaryC"
}
```

Write a policy to ensure that data entity AverageSalary was not derived from SalaryC

```
#Input 1 should satisfy the policy.
{
  "vertices":[
    {
      "name":"AverageSalary",
      "type":"dataEntity"
    },
    {
      "name":"SalaryM",
      "type": "dataEntity"
    }
  ],
  "edges":[
    {
      "relation":"wasDerivedFrom",
      "source": "AverageSalary",
      "destination": "SalaryM"
    }
  ]
}
```

Provenance policy examples

- Write a policy to ensure that activity *Average* used key entity *KeyB* and *KeyB* was attributed to account agent *Bob*



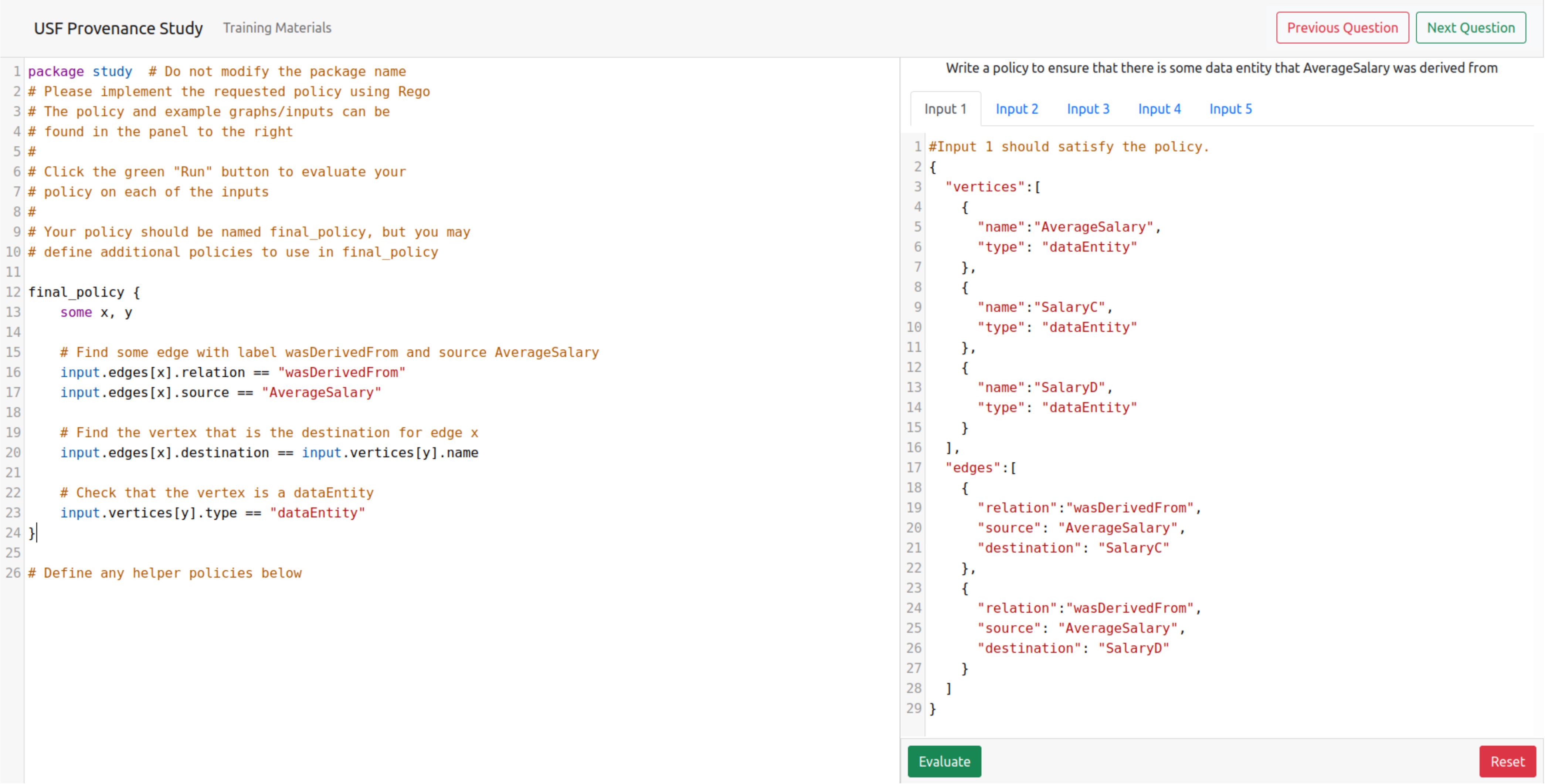
```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
    some x, y

    input.edges[x].relation == "used"
    input.edges[x].source == "Average"
    input.edges[x].destination == "KeyB"

    input.edges[y].relation == "wasAttributedTo"
    input.edges[y].source == "KeyB"
    input.edges[y].destination == "Bob"
}

# Define any helper policies below
```

Write a policy to ensure that activity Average used key entity KeyB and KeyB was attributed to account agent Bob

```
#Input 1 should satisfy the policy.
{
  "vertices":[
    {
      "name":"Bob",
      "type": "accountAgent"
    },
    {
      "name":"KeyB",
      "type": "keyEntity"
    },
    {
      "name":"Average",
      "type":"activity"
    }
  ],
  "edges":[
    {
      "relation":"wasAttributedTo",
      "source": "KeyB",
      "destination": "Bob"
    },
    {
      "relation":"used",
      "source": "Average",
      "destination": "KeyB"
    }
  ]
}
```

Provenance policy examples

- Write a policy to ensure that there is some data entity that *AverageSalary* was derived from



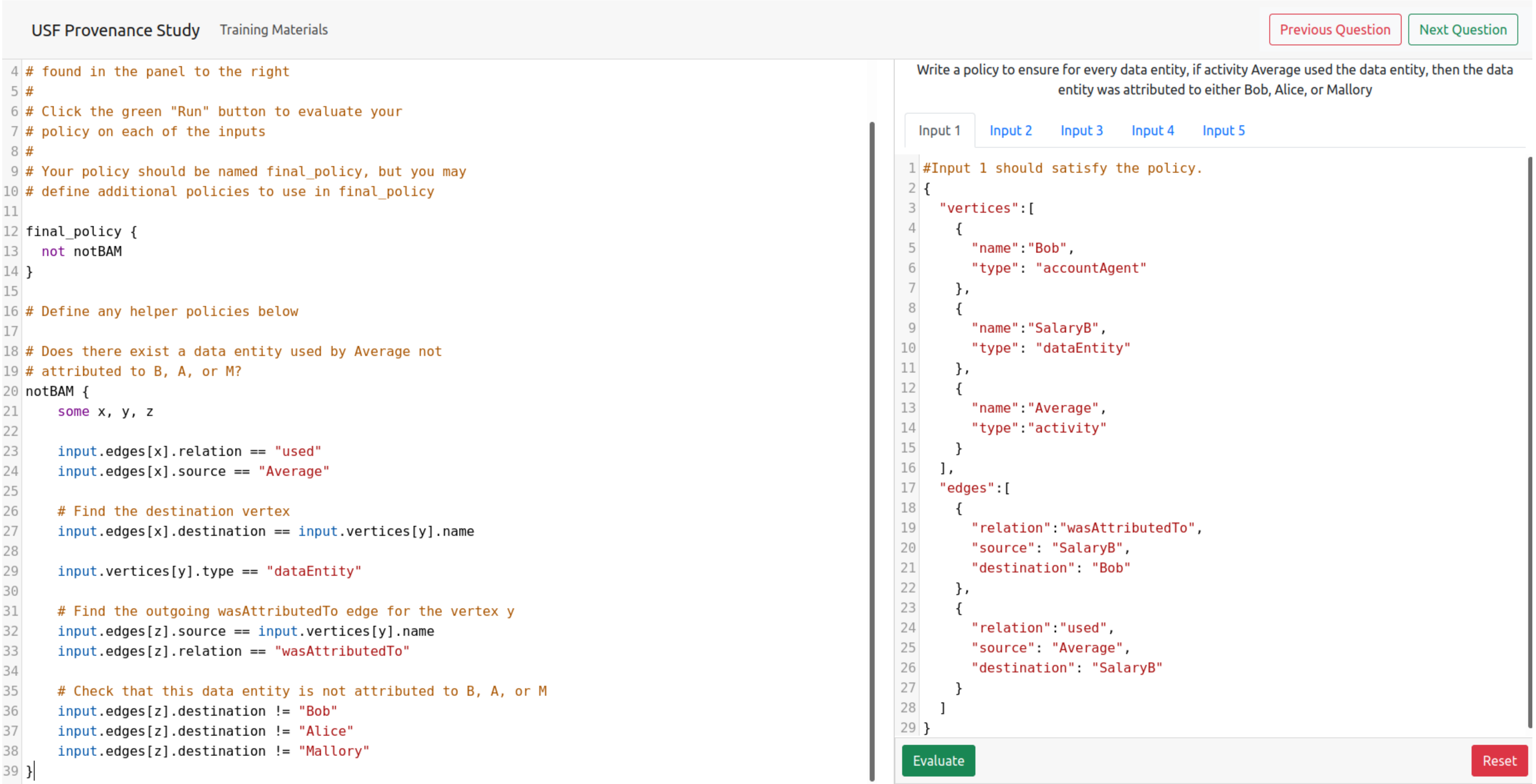
```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
    some x, y

    # Find some edge with label wasDerivedFrom and source AverageSalary
    input.edges[x].relation == "wasDerivedFrom"
    input.edges[x].source == "AverageSalary"

    # Find the vertex that is the destination for edge x
    input.edges[x].destination == input.vertices[y].name

    # Check that the vertex is a dataEntity
    input.vertices[y].type == "dataEntity"
}

# Define any helper policies below
```

Write a policy to ensure that there is some data entity that AverageSalary was derived from

```
#Input 1 should satisfy the policy.
{
  "vertices":[
    {
      "name":"AverageSalary",
      "type": "dataEntity"
    },
    {
      "name":"SalaryC",
      "type": "dataEntity"
    },
    {
      "name":"SalaryD",
      "type": "dataEntity"
    }
  ],
  "edges":[
    {
      "relation":"wasDerivedFrom",
      "source": "AverageSalary",
      "destination": "SalaryC"
    },
    {
      "relation":"wasDerivedFrom",
      "source": "AverageSalary",
      "destination": "SalaryD"
    }
  ]
}
```

Provenance policy examples

- Write a policy to ensure for every data entity, if activity *Average* used the data entity, then the data entity was attributed to either Bob, Alice, or Mallory

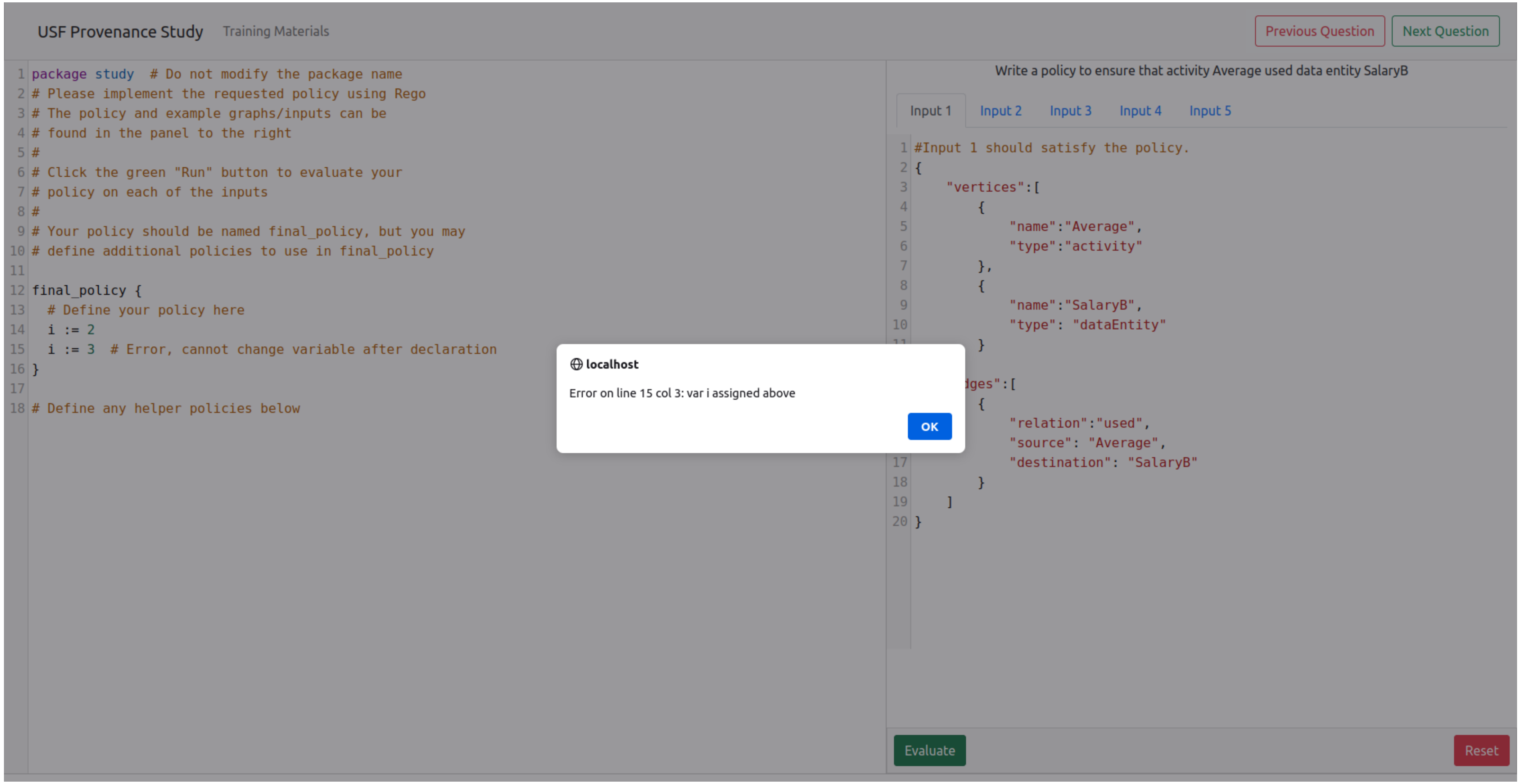


```
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
  not notBAM
}

# Define any helper policies below

# Does there exist a data entity used by Average not
# attributed to B, A, or M?
notBAM {
    some x, y, z

    input.edges[x].relation == "used"
    input.edges[x].source == "Average"

    # Find the destination vertex
    input.edges[x].destination == input.vertices[y].name

    input.vertices[y].type == "dataEntity"

    # Find the outgoing wasAttributedTo edge for the vertex y
    input.edges[z].source == input.vertices[y].name
    input.edges[z].relation == "wasAttributedTo"

    # Check that this data entity is not attributed to B, A, or M
    input.edges[z].destination != "Bob"
    input.edges[z].destination != "Alice"
    input.edges[z].destination != "Mallory"
}
```

Write a policy to ensure for every data entity, if activity Average used the data entity, then the data entity was attributed to either Bob, Alice, or Mallory

```
#Input 1 should satisfy the policy.
{
  "vertices":[
    {
      "name":"Bob",
      "type": "accountAgent"
    },
    {
      "name":"SalaryB",
      "type": "dataEntity"
    },
    {
      "name":"Average",
      "type":"activity"
    }
  ],
  "edges":[
    {
      "relation":"wasAttributedTo",
      "source": "SalaryB",
      "destination": "Bob"
    },
    {
      "relation":"used",
      "source": "Average",
      "destination": "SalaryB"
    }
  ]
}
```

Rego Incorrect Answers



```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
  # Define your policy here
  i := 2
  i := 3  # Error, cannot change variable after declaration
}

# Define any helper policies below
```

localhost

Error on line 15 col 3: var i assigned above

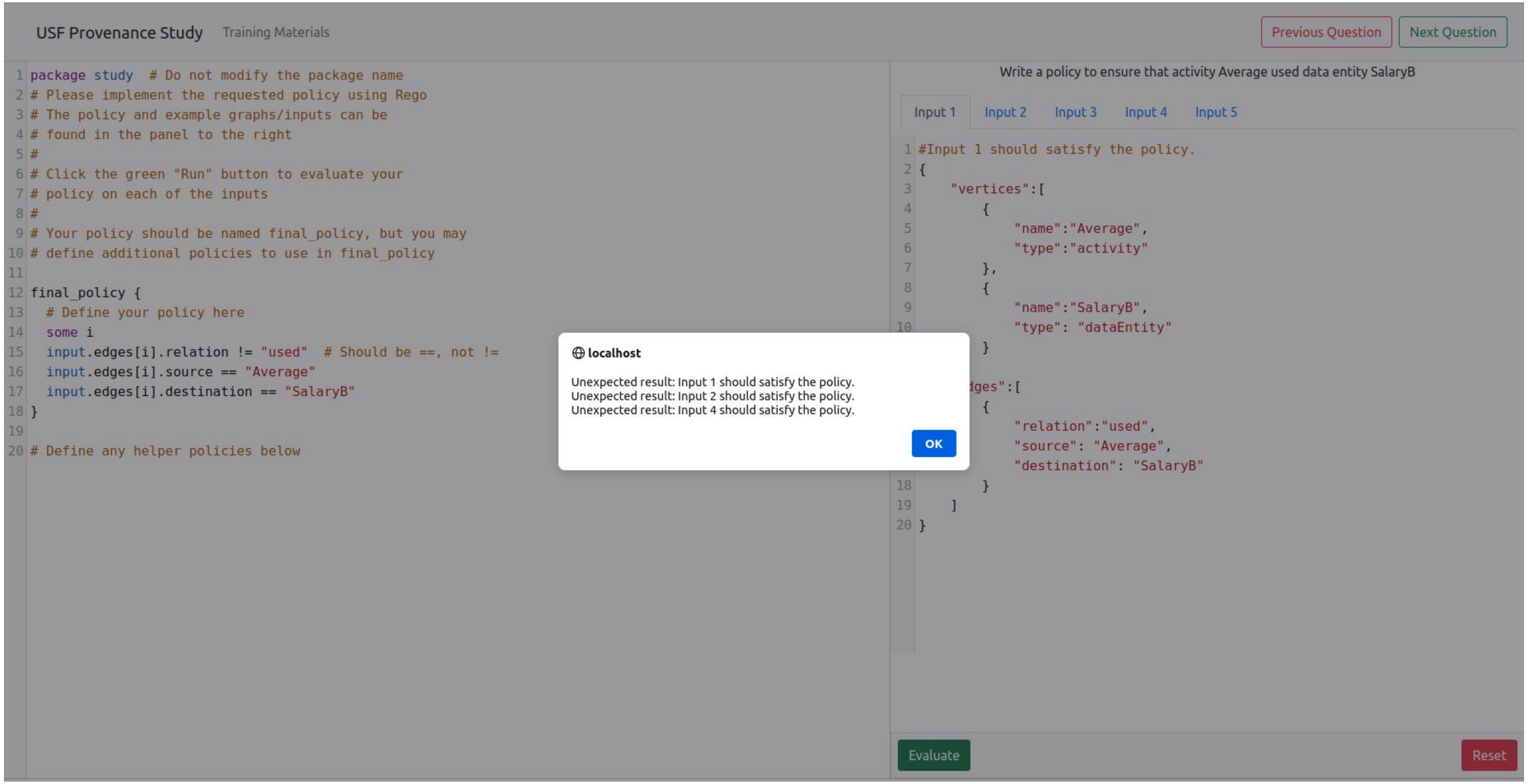
OK

Write a policy to ensure that activity Average used data entity SalaryB

```
#Input 1 should satisfy the policy.
{
    "vertices":[
        {
            "name":"Average",
            "type":"activity"
        },
        {
            "name":"SalaryB",
            "type": "dataEntity"
        }
```

```
    dges":[
        {
            "relation":"used",
            "source": "Average",
            "destination": "SalaryB"
        }
    ]
}
```

Rego Incorrect Answers



```
package study  # Do not modify the package name
# Please implement the requested policy using Rego
# The policy and example graphs/inputs can be
# found in the panel to the right
#
# Click the green "Run" button to evaluate your
# policy on each of the inputs
#
# Your policy should be named final_policy, but you may
# define additional policies to use in final_policy

final_policy {
  # Define your policy here
  some i
  input.edges[i].relation != "used"  # Should be ==, not !=
  input.edges[i].source == "Average"
  input.edges[i].destination == "SalaryB"
}

# Define any helper policies below
```

Write a policy to ensure that activity Average used data entity SalaryB

```
#Input 1 should satisfy the policy.
{
    "vertices":[
        {
            "name":"Average",
            "type":"activity"
        },
        {
            "name":"SalaryB",
            "type": "dataEntity"
        }

  dges":[
        {
            "relation":"used",
            "source": "Average",
            "destination": "SalaryB"
        }
    ]
}
```

```
Unexpected result: Input 1 should satisfy the policy.
Unexpected result: Input 2 should satisfy the policy.
Unexpected result: Input 4 should satisfy the policy.
```