Announcement

- Some exercises and homeworks need to be submitted at the end of the semester
  → will be indicated on the slides

- Please consider to stick to a consistent folder structure; e.g.:

```
[student-ID]-AI-handin.zip
|-----> Lab 1 (folder)
---------|-----------> Exercises (folder containing .py files)
---------|-----------> Homework (folder containing .py files)
|-----> Lab 2
---------|-----------> Exercises
---------|-----------> Homework
etc…
```

Agents

Lab 1

Agenda

1. Running example: vacuum-cleaner world
2. Table-driven agent
3. Simple reflex agent
4. Reflex agent with state/memory
5. Homework

Vacuum-cleaner world

Percepts:

- Location, status (e.g., [A, dirty])

Actions:

- Left, Right, Suck, NoOperation

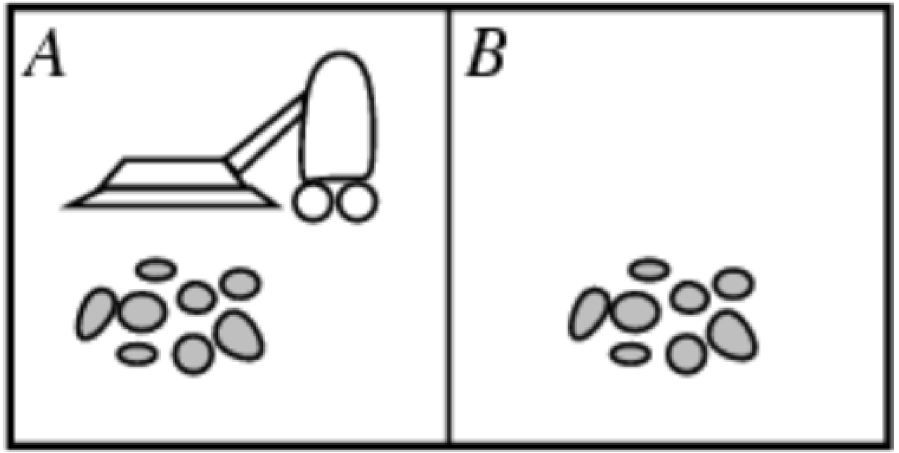


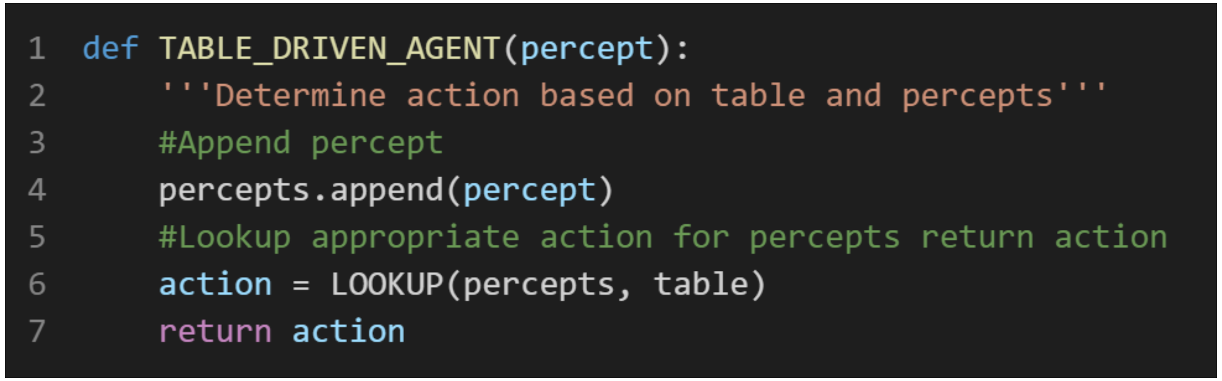
Table-driven agent

Table-driven agent

- Refer to table_driven_agent.png
- **Table** contains all possible *percepts* that can occur
- Each step appends current *percept* to list of *percepts*
- **LOOKUP** current *percepts* in *table*

Table-driven agent

```
function TABLE-DRIVEN-AGENT(percept) returns an action
    static: percepts, a sequence, initially empty
            table, a table of actions, indexed by percept sequences, initially fully specified

    append percept to the end of percepts
    action = LOOKUP(percepts, table)
    return action
```



```python
def TABLE_DRIVEN_AGENT(percept):
    '''Determine action based on table and percepts'''
    #Append percept
    percepts.append(percept)
    #Lookup appropriate action for percepts return action
    action = LOOKUP(percepts, table)
    return action
```

Exercise 1

1. Run the module (using run())
2. The percepts should now be: [(‘A’, ’Clean’), (‘A’, ’Dirty’), (‘´B’, ’Clean’)]
    - The table contains all possible percept sequences to match with the percept history
    - Enter:
      print(TABLE_DRIVEN_AGENT((B, 'Clean')), '\t', percepts)
    - Explain the results
3. How many table entries would be required if only the *current* percept was used to select and action rather than the percept history?
4. How many table entries are required for an agent lifetime of T steps?

Simple reflex agent

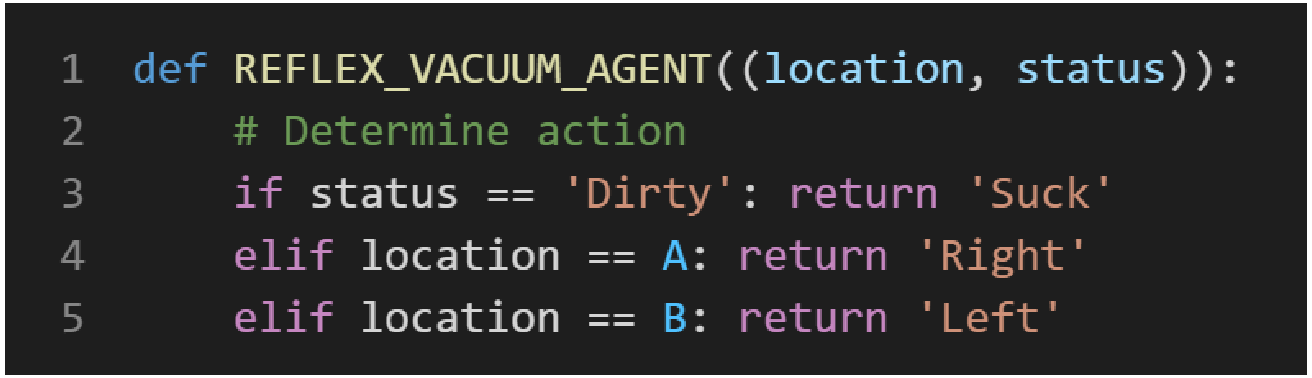
using condition-action rules and if statements

Simple reflex agent

- Refer to reflex_vacuum_agent.png
- Only responds to current percept (location and status) ignoring percept history
- Uses *condition-action* rules rather than a table
  - **if** *conditio*n **then return** *action*
  - **if** *status = Dirty* **then return** *Suck*
- **Sensors()** – Function to sense current location and status of environment (i.e., *location* of agent and *status* of square)
- **Actuators(action)** – Function to affect current environment location by some action (i.e., *Suck, Left, Right, NoOp*)

Simple reflex agent

```
function REFLEX-VACUUM-AGENT( [location, status] )
  returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```



```python
def REFLEX_VACUUM_AGENT((location, status)):
    # Determine action
    if status == 'Dirty': return 'Suck'
    elif location == A: return 'Right'
    elif location == B: return 'Left'
```

Exercise 2

1. Run the module
2. Enter *run(10)*
3. Should bogus actions be able to corrupt the environment? Change the REFLEX_VACUUM_AGENT to return bogus action, such as *Left* when it should go *Right* etc. Run the agent. Do the Actuators allow bogus actions?

Simple reflex agent

using condition-action rules and dictionaries

Simple reflex agent

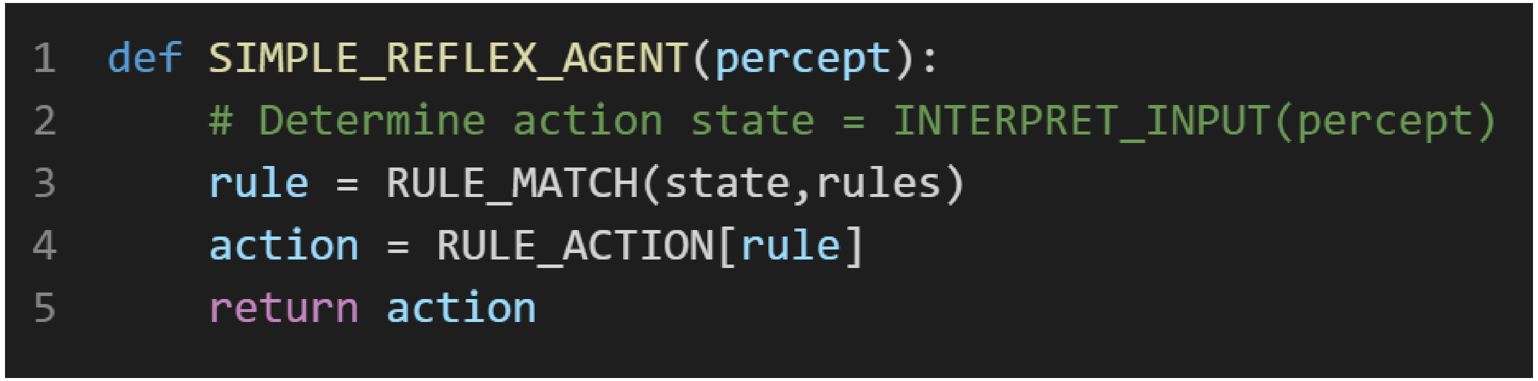
- Refer to simple_reflex_agent.png
- **Condition-action rules**

  - **rules** = { (A,'Dirty'):1, (B,'Dirty'):1, (A,'Clean'):2, (B,'Clean'):3, (A, B, 'Clean'):4 }
    Defines *rule* for each *condition* such as: condition == (A,’Dirty’) uses rule 1

  - RULE_ACTION = { 1:'Suck', 2:'Right', 3:'Left', 4:'NoOp’ }
    Defines *action* for each *rule* such as: rule 1 produces action ‘Suck’

Simple reflex agent

```
function SIMPLE-REFLEX-AGENT( percept ) returns an action
    static: rules, a set of condition-action rules

    state = INTERPRET-INPUT( percept )
    rule = RULE-MATCH( state, rules )
    action = RULE-ACTION[ rule ]
    return action
```



```python
def SIMPLE_REFLEX_AGENT(percept):
    # Determine action state = INTERPRET_INPUT(percept)
    rule = RULE_MATCH(state,rules)
    action = RULE_ACTION[rule]
    return action
```

Exercise 3

1. Run the module
2. Enter *run(10)*
3. Change the SIMPLE_REFLEX_AGENT *condition-action* rules to return bogus actions, such as *Left* when should go *Right*, or *Crash*, etc. Rerun the agent. Do the Actuators allow bogus actions?

Reflex agent with state/memory

Reflex agent with state

- Reflex agent only responded to current percepts; no history or knowledge
- Model-based reflex agents:
  - Maintain internal state that depends upon percept history
  - Agent has a model of how the world works
  - The model requires two types of information to update:
    - How environment evolves independent of the agent (e.g., Clean square stays clean)
    - How agent’s action affect the environment (e.g., Suck cleans square)

Reflex agent with state

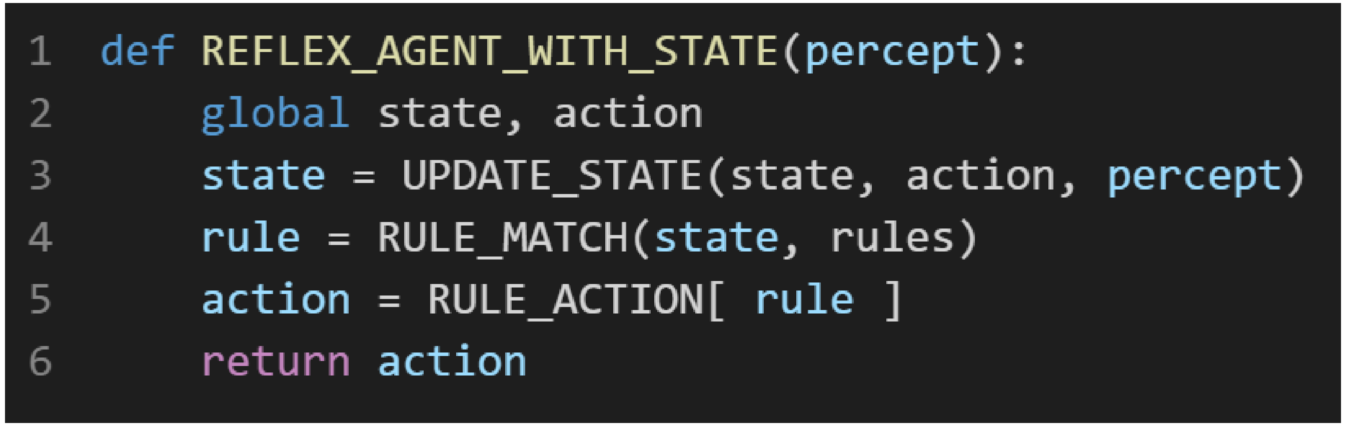
- Refer to reflex_agent_with_state.png
- Model – used to update history
  - History initially empty:

    model = {A: None, B: None}
  - Model only used to change state when A == B == ‘Clean’

    if model[A] == model[B] == ‘Clean’: state = (A, B, ‘Clean’)

Simple reflex agent

```
function REFLEX-AGENT-WITH-STATE( percept ) returns an action
    static: state, a description of the current world state
            rules, a sequence, a set of condition-action rules
            action, the most recent action, initially none
    state = UPDATE-STATE( state, action, percept )
    rule   = RULE-MATCH( state, rules )
    action = RULE-ACTION[ rule ]
    return action
```



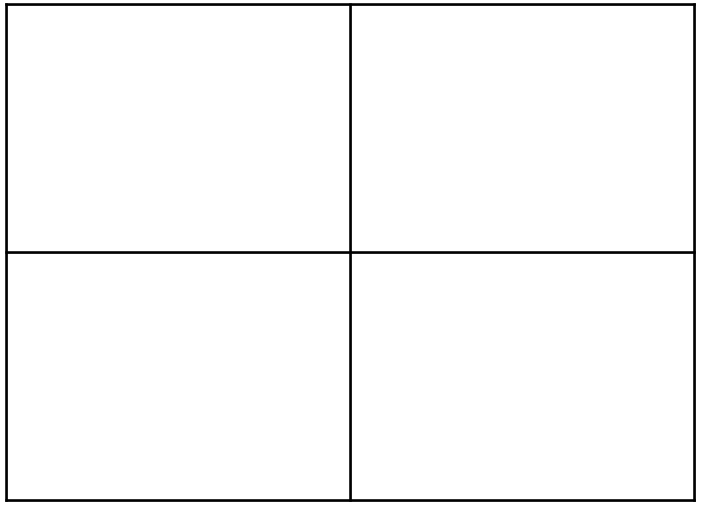
```python
def REFLEX_AGENT_WITH_STATE(percept):
    global state, action
    state = UPDATE_STATE(state, action, percept)
    rule = RULE_MATCH(state, rules)
    action = RULE_ACTION[ rule ]
    return action
```

Homework

Homework 1 – Simple Reflex Agent

Must be submitted

- Extend the REFLEX_VACUUM_AGENT (Exercise 2) program to have 4 locations (4 squares)
  - The agent should only sense and act on the square where it is located
  - Allow any starting square
  - Use run(20) to test and display results



Homework 2 – Reflex agent with state

Must be submitted

- Extend the REFLEX_AGENT_WITH_STATE program to have 4 locations
  - The agent should only sense and act on the square where it is located
  - Allow any starting square
  - Use run(20) to test and display results

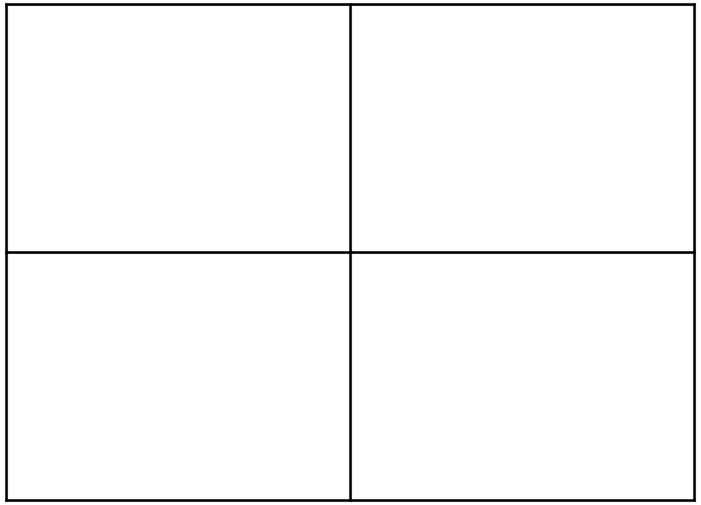
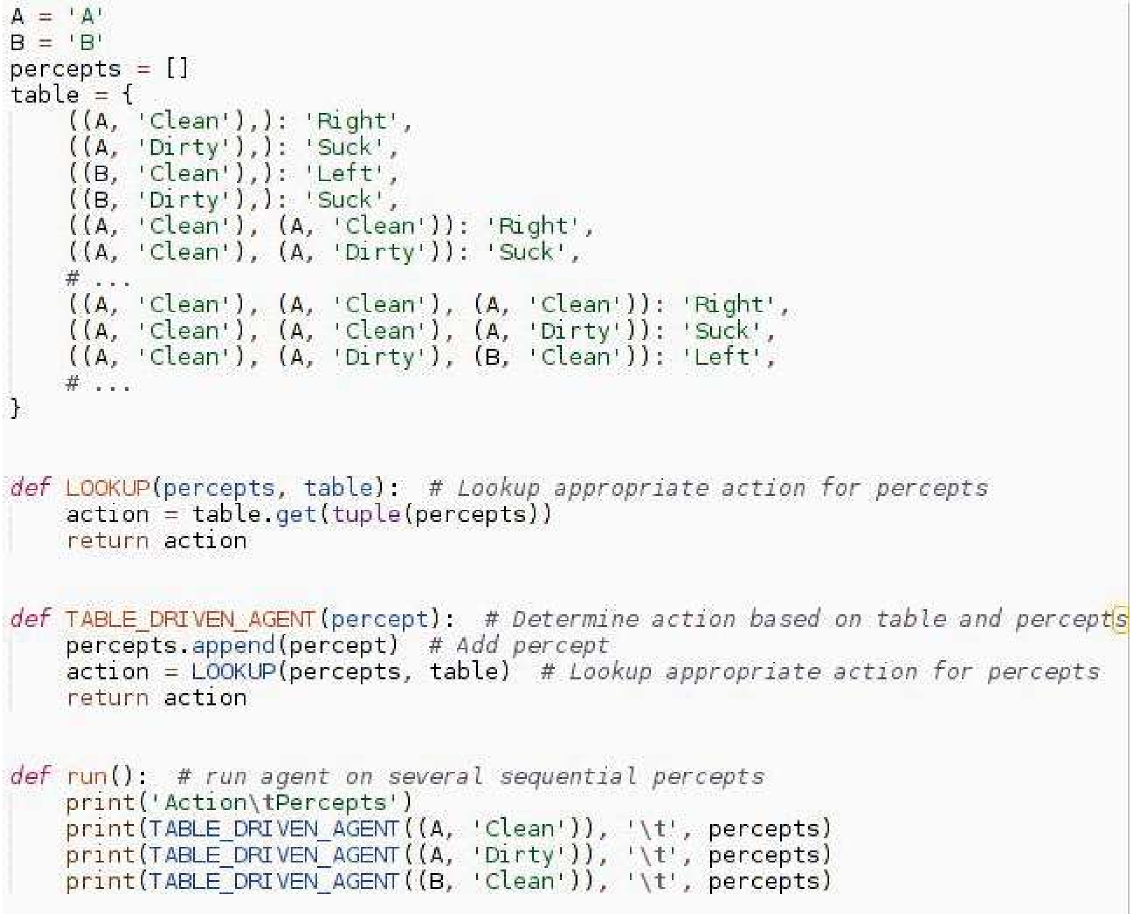
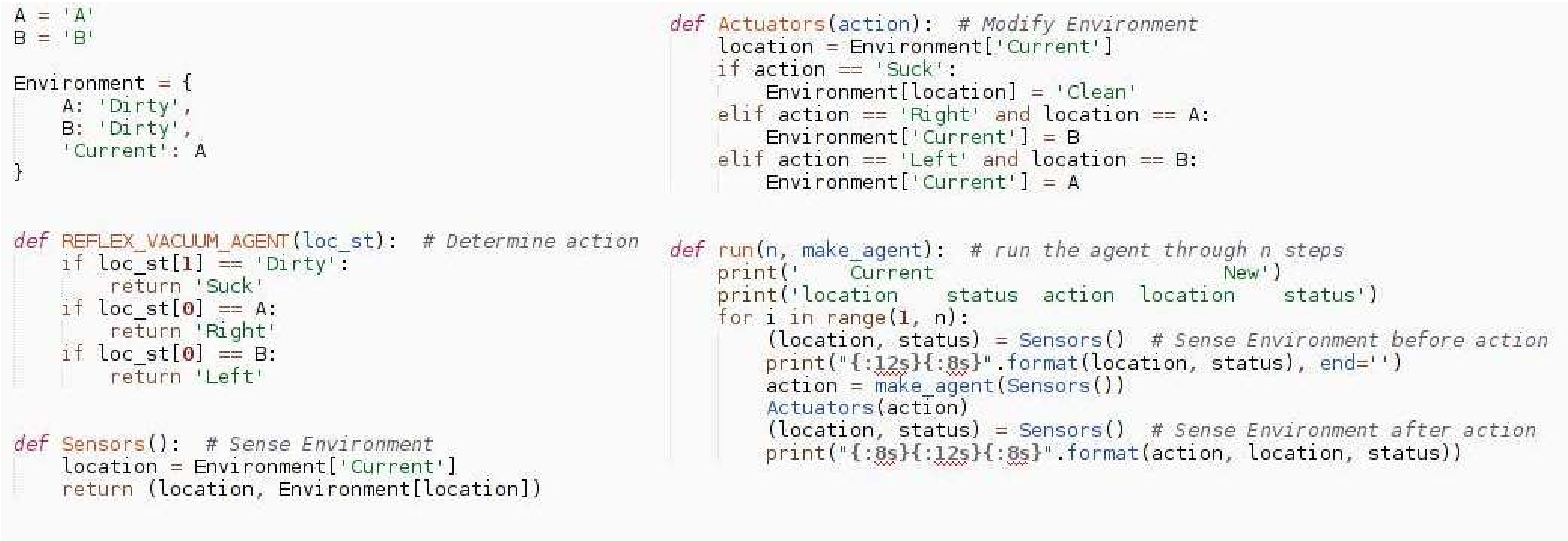


Table driven agent



```python
A = 'A'
B = 'B'
percepts = []
table = {
    ((A, 'Clean'),): 'Right',
    ((A, 'Dirty'),): 'Suck',
    ((B, 'Clean'),): 'Left',
    ((B, 'Dirty'),): 'Suck',
    ((A, 'Clean'), (A, 'Clean')): 'Right',
    ((A, 'Clean'), (A, 'Dirty')): 'Suck',
    # ...
    ((A, 'Clean'), (A, 'Clean'), (A, 'Clean')): 'Right',
    ((A, 'Clean'), (A, 'Clean'), (A, 'Dirty')): 'Suck',
    ((A, 'Clean'), (A, 'Dirty'), (B, 'Clean')): 'Left',
    # ...
}


def LOOKUP(percepts, table):  # Lookup appropriate action for percepts
    action = table.get(tuple(percepts))
    return action


def TABLE_DRIVEN_AGENT(percept):  # Determine action based on table and percepts
    percepts.append(percept)  # Add percept
    action = LOOKUP(percepts, table)  # Lookup appropriate action for percepts
    return action


def run():  # run agent on several sequential percepts
    print('Action\tPercepts')
    print(TABLE_DRIVEN_AGENT((A, 'Clean')), '\t', percepts)
    print(TABLE_DRIVEN_AGENT((A, 'Dirty')), '\t', percepts)
    print(TABLE_DRIVEN_AGENT((B, 'Clean')), '\t', percepts)
```

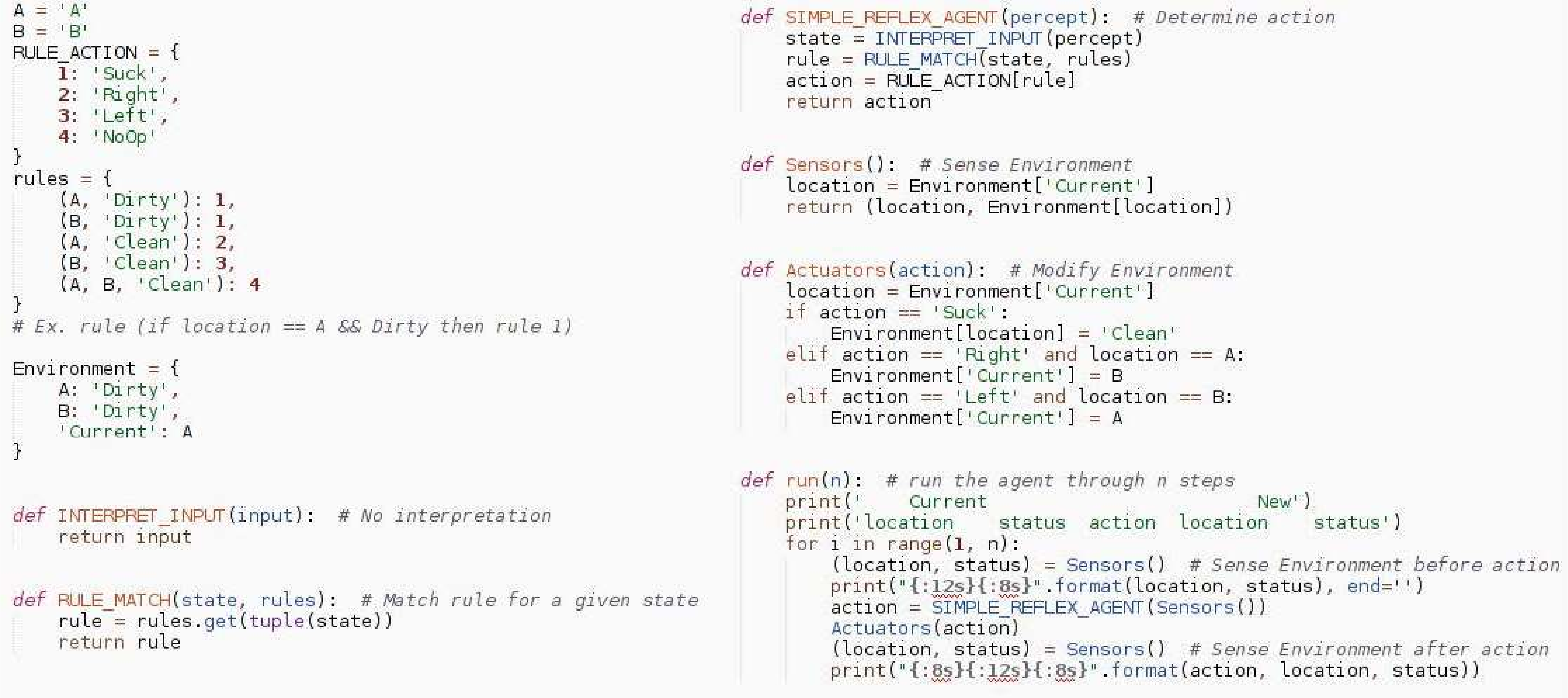
Reflex vacuum agent



```
A = 'A'
B = 'B'

Environment = {
    A: 'Dirty',
    B: 'Dirty',
    'Current': A
}


def REFLEX_VACUUM_AGENT(loc_st):  # Determine action
    if loc_st[1] == 'Dirty':
        return 'Suck'
    if loc_st[0] == A:
        return 'Right'
    if loc_st[0] == B:
        return 'Left'


def Sensors():  # Sense Environment
    location = Environment['Current']
    return (location, Environment[location])
```

```
def Actuators(action):  # Modify Environment
    location = Environment['Current']
    if action == 'Suck':
        Environment[location] = 'Clean'
    elif action == 'Right' and location == A:
        Environment['Current'] = B
    elif action == 'Left' and location == B:
        Environment['Current'] = A


def run(n, make_agent):  # run the agent through n steps
    print('    Current                        New')
    print('location    status  action  location    status')
    for i in range(1, n):
        (location, status) = Sensors()  # Sense Environment before action
        print("{:12s}{:8s}".format(location, status), end='')
        action = make_agent(Sensors())
        Actuators(action)
        (location, status) = Sensors()  # Sense Environment after action
        print("{:8s}{:12s}{:8s}".format(action, location, status))
```

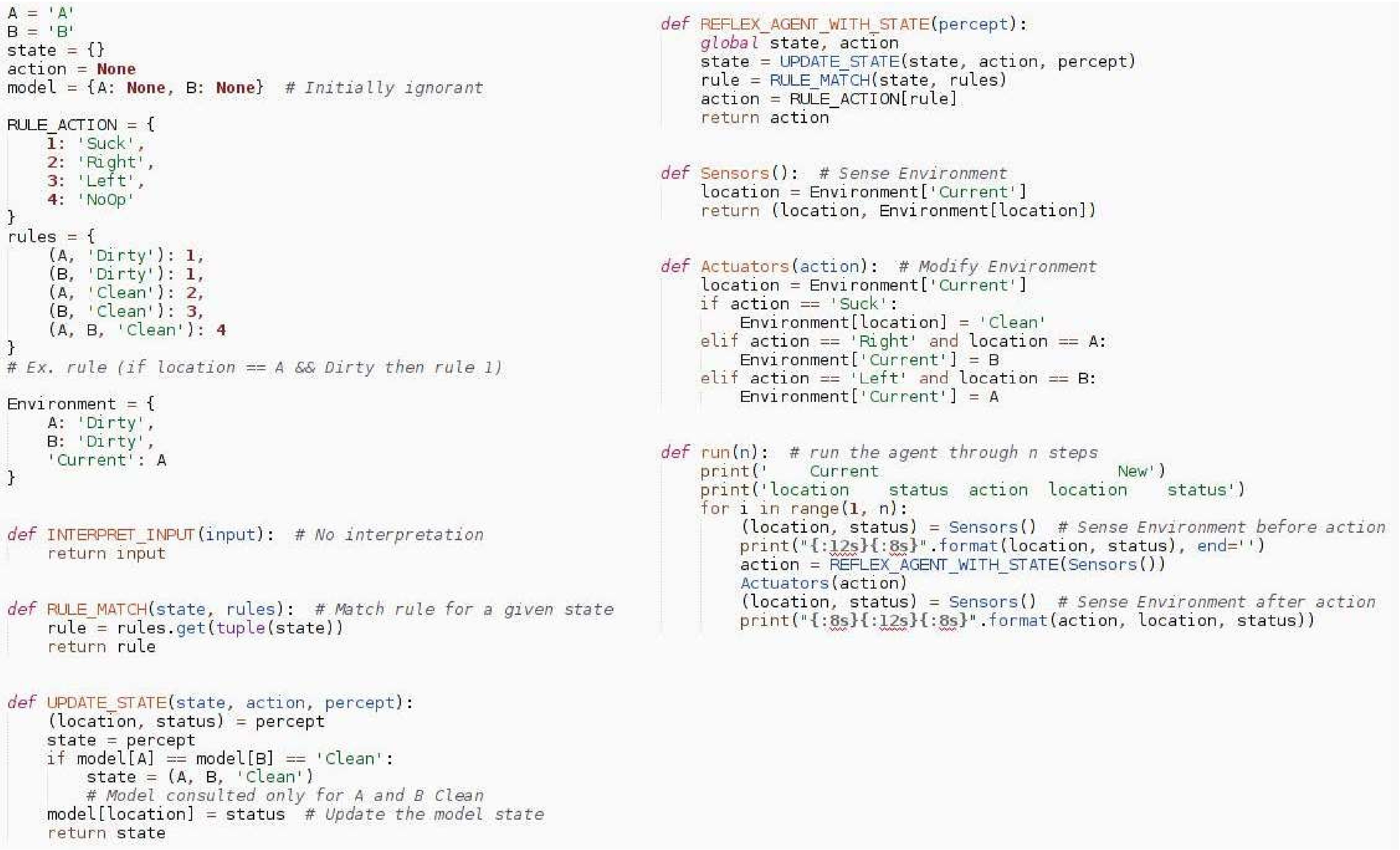
Simple reflex agent



```
A = 'A'
B = 'B'
RULE_ACTION = {
    1: 'Suck',
    2: 'Right',
    3: 'Left',
    4: 'NoOp'
}
rules = {
    (A, 'Dirty'): 1,
    (B, 'Dirty'): 1,
    (A, 'Clean'): 2,
    (B, 'Clean'): 3,
    (A, B, 'Clean'): 4
}
# Ex. rule (if location == A && Dirty then rule 1)

Environment = {
    A: 'Dirty',
    B: 'Dirty',
    'Current': A
}


def INTERPRET_INPUT(input):  # No interpretation
    return input


def RULE_MATCH(state, rules):  # Match rule for a given state
    rule = rules.get(tuple(state))
    return rule
```

```
def SIMPLE_REFLEX_AGENT(percept):  # Determine action
    state = INTERPRET_INPUT(percept)
    rule = RULE_MATCH(state, rules)
    action = RULE_ACTION[rule]
    return action


def Sensors():  # Sense Environment
    location = Environment['Current']
    return (location, Environment[location])


def Actuators(action):  # Modify Environment
    location = Environment['Current']
    if action == 'Suck':
        Environment[location] = 'Clean'
    elif action == 'Right' and location == A:
        Environment['Current'] = B
    elif action == 'Left' and location == B:
        Environment['Current'] = A


def run(n):  # run the agent through n steps
    print('    Current                        New')
    print('location    status  action  location    status')
    for i in range(1, n):
        (location, status) = Sensors()  # Sense Environment before action
        print("{:12s}{:8s}".format(location, status), end='')
        action = SIMPLE_REFLEX_AGENT(Sensors())
        Actuators(action)
        (location, status) = Sensors()  # Sense Environment after action
        print("{:8s}{:12s}{:8s}".format(action, location, status))
```

Reflex agent with state



```python
A = 'A'
B = 'B'
state = {}
action = None
model = {A: None, B: None}  # Initially ignorant

RULE_ACTION = {
    1: 'Suck',
    2: 'Right',
    3: 'Left',
    4: 'NoOp'
}
rules = {
    (A, 'Dirty'): 1,
    (B, 'Dirty'): 1,
    (A, 'Clean'): 2,
    (B, 'Clean'): 3,
    (A, B, 'Clean'): 4
}
# Ex. rule (if location == A && Dirty then rule 1)

Environment = {
    A: 'Dirty',
    B: 'Dirty',
    'Current': A
}


def INTERPRET_INPUT(input):  # No interpretation
    return input


def RULE_MATCH(state, rules):  # Match rule for a given state
    rule = rules.get(tuple(state))
    return rule


def UPDATE_STATE(state, action, percept):
    (location, status) = percept
    state = percept
    if model[A] == model[B] == 'Clean':
        state = (A, B, 'Clean')
        # Model consulted only for A and B Clean
    model[location] = status  # Update the model state
    return state
```

```python
def REFLEX_AGENT_WITH_STATE(percept):
    global state, action
    state = UPDATE_STATE(state, action, percept)
    rule = RULE_MATCH(state, rules)
    action = RULE_ACTION[rule]
    return action


def Sensors():  # Sense Environment
    location = Environment['Current']
    return (location, Environment[location])


def Actuators(action):  # Modify Environment
    location = Environment['Current']
    if action == 'Suck':
        Environment[location] = 'Clean'
    elif action == 'Right' and location == A:
        Environment['Current'] = B
    elif action == 'Left' and location == B:
        Environment['Current'] = A


def run(n):  # run the agent through n steps
    print('    Current                        New')
    print('location    status  action  location    status')
    for i in range(1, n):
        (location, status) = Sensors()  # Sense Environment before action
        print("{:12s}{:8s}".format(location, status), end='')
        action = REFLEX_AGENT_WITH_STATE(Sensors())
        Actuators(action)
        (location, status) = Sensors()  # Sense Environment after action
        print("{:8s}{:12s}{:8s}".format(action, location, status))
```