2.3 Finite State Machine (FSM) Concept and Implementation

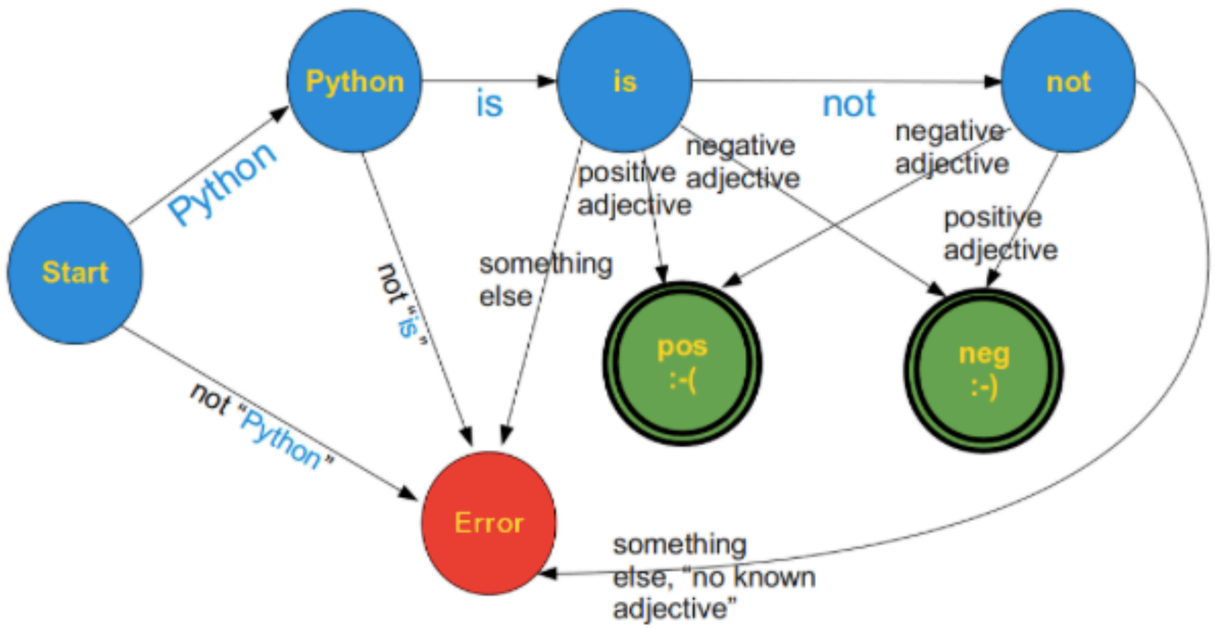
Topics

- Finite State Machine (FSM)
  - What are FSM’s
  - Why / When to use FSM
- Implementing of Finite State Machines
- Home Work Assignment (part 2)

What Is A Finite State Machine

(a.k.a Finite-state Automaton)

An Example



Start
Python
Python
is
is
not
not
not "Python"
not "is"
something else
positive adjective
negative adjective
negative adjective
positive adjective
pos :-(
neg :-)
Error
something else, "no known adjective"

FSM Examples in Daily Live

- Vending Machines
- Traffic Lights
- Elevators
- Alarm Clock
- Microwave
- Cash Registers

Each of these devices can be thought of as a *reactive system* - that is because each of them work by *reacting* to signals or inputs from the external world.

What Is A Finite State Machine

- A reactive system whose response to a particular stimulus (a *signal*, or a piece of *input*) is not the same on every occasion, depending on its current “state”.

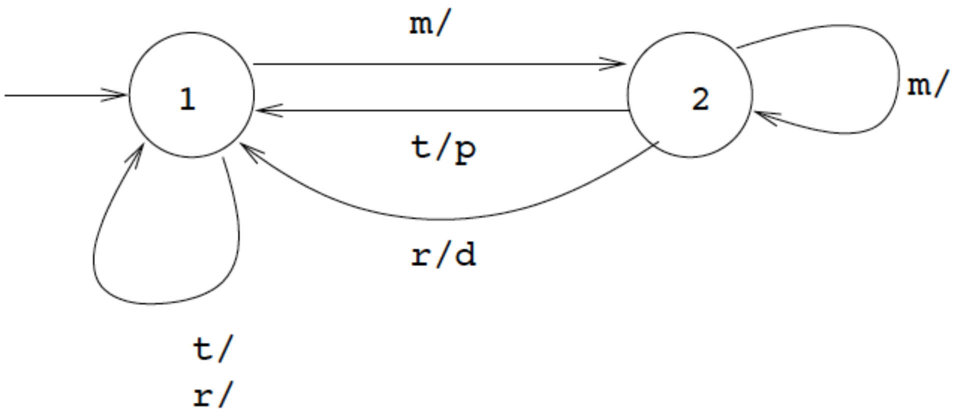
- For example, in the case of a parking ticket machine, it will not print a ticket when you press the button unless you have already inserted some money. Thus the response to the print button depends on the previous *history* of the use of the system.

More Precisely (Formally)

- A Finite State Machine is defined by $(\Sigma, S, s_0, \delta, F)$, where:
  - Σ is the input alphabet (a finite, non-empty set of symbols).
  - S is a finite, non-empty set of states.
  - s_0 is an initial state, an element of S.
  - δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$
  - F is the set of final states, a (possibly empty) subset of S.
  - O is the set (possibly empty) of outputs

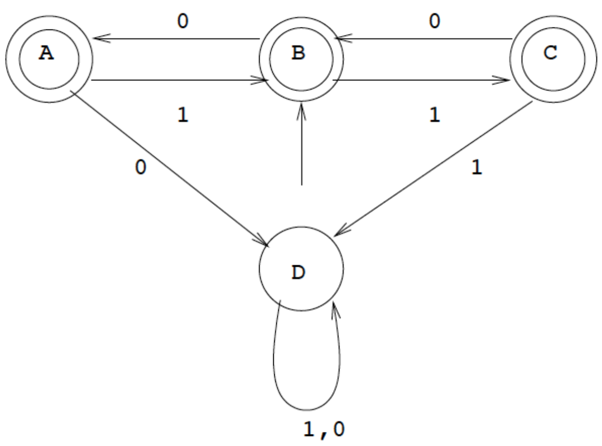
A (Simplified) Ticket Machine

- Σ (m, t, r) : inserting money, requesting ticket, requesting refund
- S (1, 2) : unpaid, paid
- s_0 (1) : an initial state, an element of S.
- δ (shown below) : transition function: $\delta : S \times \Sigma \to S$
- F : empty
- O (p/d) : print ticket, deliver refund

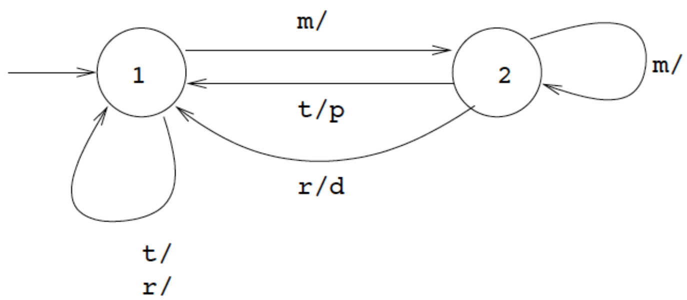


Acceptors and Transducers

- Acceptors: no output, have final states
- Transducers: non-empty set of output



Acceptor



Transducer

Deterministic and Non-Deterministic

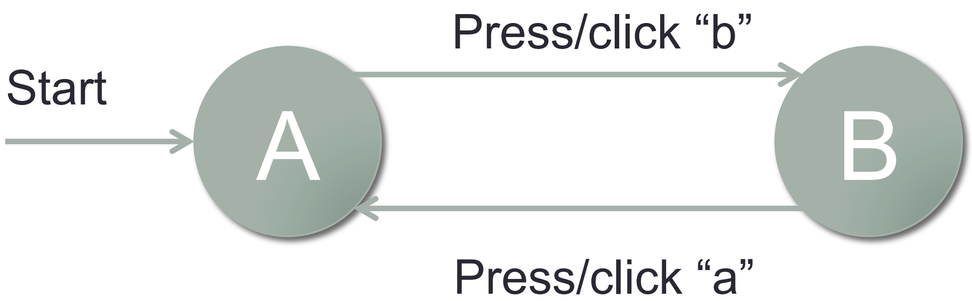
- Non-deterministic: Competing “Transitions” Leaving Same State

We only concern ourselves with Deterministic FSM in this class

How To Implement an FSM

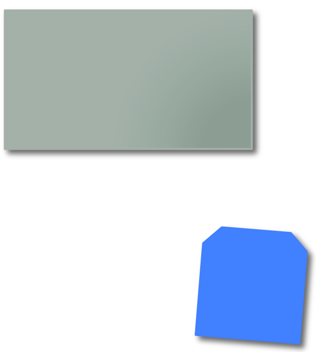
- The Finite State Machine class keeps track of the current state, and the list of valid state transitions.
- You define each transition by specifying :
  - FromState - the starting state for this transition
  - ToState - the end state for this transition
  - condition - a callable which when it returns True means this transition is valid
  - callback - an optional callable function which is invoked when this transition is executed.

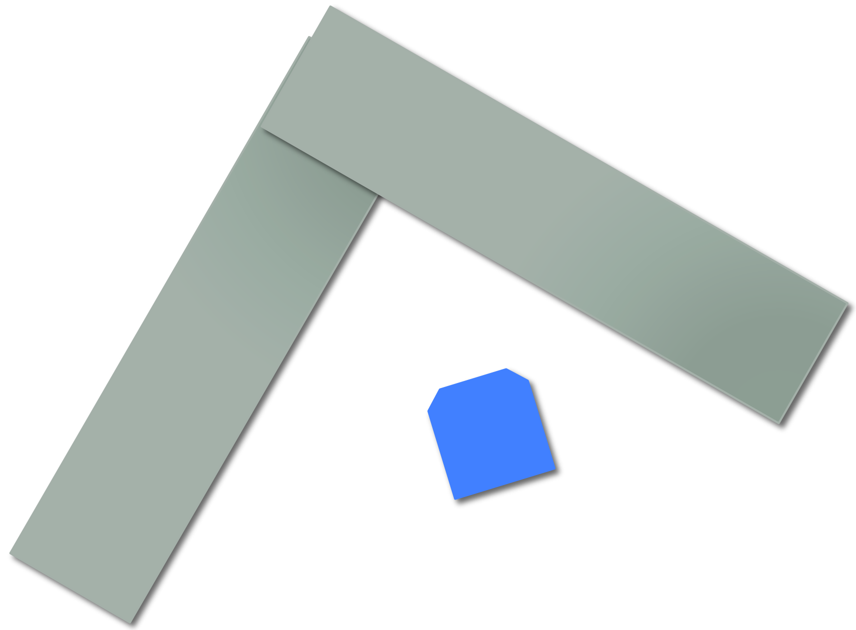
Simplest FSM



Why Finite State Machines For Robot

- Response to an event is dependent on the “state” of the robot



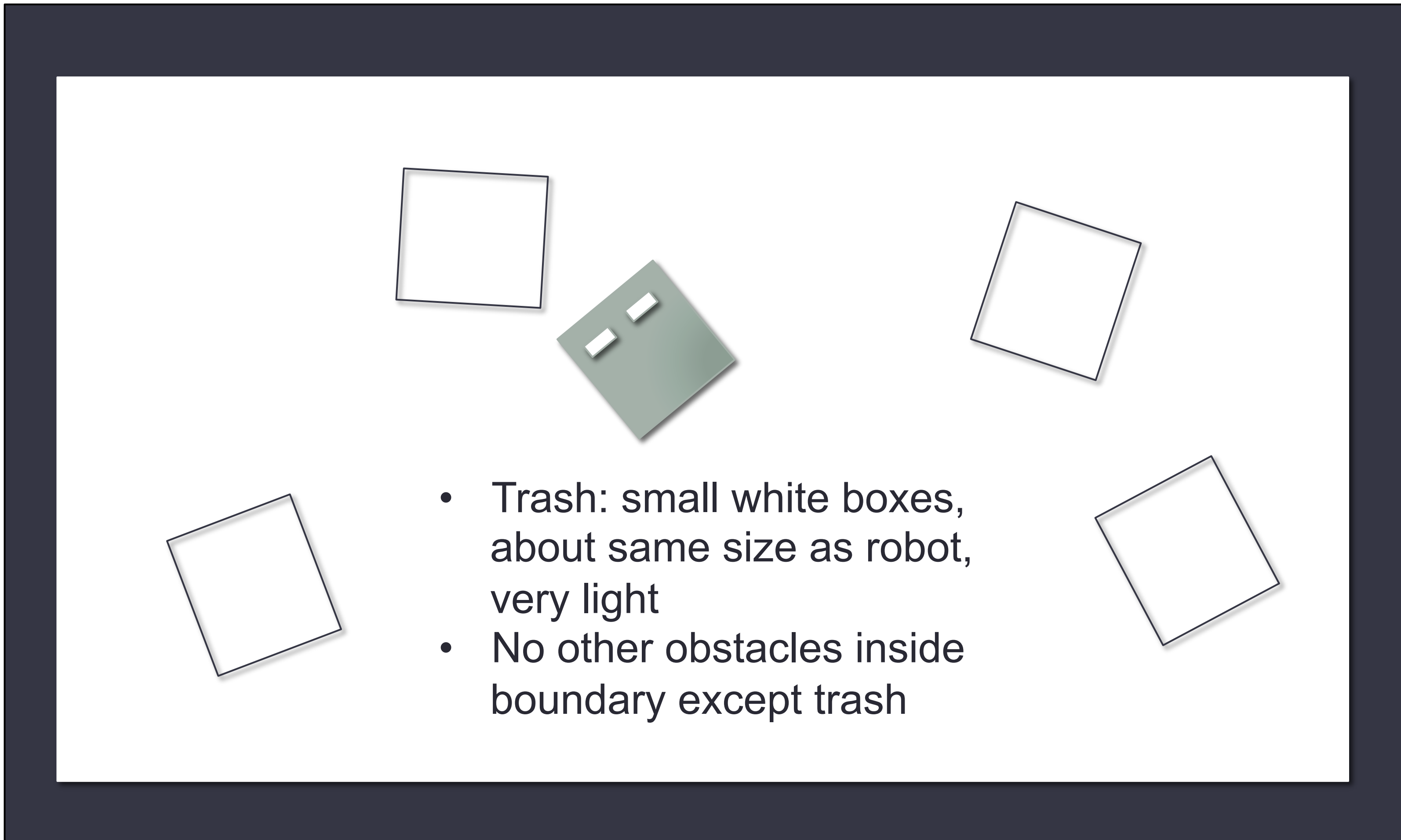


Turn-left, turn-right

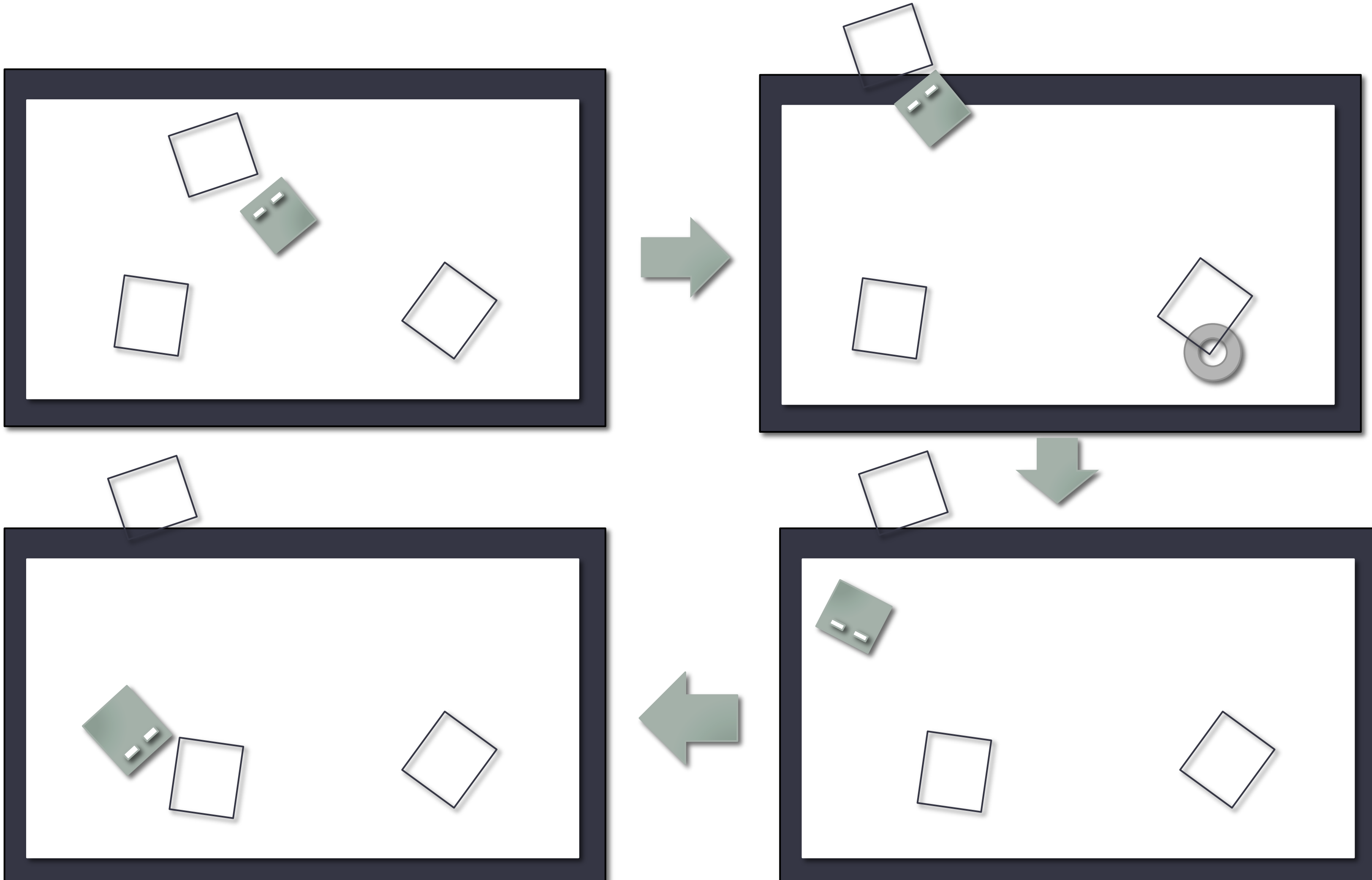
Two Robot Examples

- Obstacle Avoidance Example
- “Escape” Example

Home Work #2-2: “Cleaner” (Push Out “Trash”)



Clean Out Trash



HW Specification

- Push out trash to outside of boundary (black tape) – at least half of the “box” is outside of boundary
- Indicate (with sound or light) that track has been pushed out
- Quit (success condition) after pushing 3 pieces of trash out
- Assumptions:
  - No other object inside boundary except trash
  - Trash are small white boxes about the same size as the robot