

# Object Oriented Programming and State Machines

Goal: To model your progress through Course 6 at MIT using new subclasses of sm.SM

Steps:

1. State Transition Diagrams- Modeling Course 6
2. Object Oriented Programming and Coding Style- Modeling Progress
  - a. Transcript Object
  - b. State Machines
3. State Machine Combinators- Adding your HASS requirement
  - a. ListComprehensions/PureFunctionSM- Computing GPA
  - b. Cascade and Parallel Connections- Adding HASS
4. Further extensions- PE classes

## **Course Listing**

Course	Passing Grade	PreRequisite
6.01	C	None
6.02	D	C in 6.01
6.002	D	B in 6.02 or A-D in 6.042
6.042	D	C/D in 6.02
6.003	C	A/B in 6.002, A-C in 6.041
6.041	C	C/D in 6.002, D in 6.003
6.004	C	A-C in 6.003

Students failing to acquire minimum passing grade will repeat class before next class can be taken.

What does the state transition diagram look like?

**Outputs**

	A	B	C	D	F
6.01	“Done 6.01”	“Done 6.01”	“Done 6.01”		“Fail 6.01”
6.02	“Done 6.02”	“Done 6.02”	“Done 6.02”		“Fail 6.02”
6.002	“Done 6.002”	“Done 6.002”	“Done 6.002”		“Fail 6.002”
6.042	“Done 6.042”	“Done 6.042”	“Done 6.042”		“Fail 6.042”
6.003					
6.041	“Done 6.041”	“Done 6.041”	“Done 6.041”		“Fail 6.041”
6.004	“”Done”	“Done”	“Done”		“Fail 6.004”

**States**

	A	B	C	D	F
6.01	6.02		6.02	6.01	6.01
6.02	6.002		6.042	6.042	6.02
6.002	6.003		6.041	6.041	6.002
6.042	6.002		6.002	6.002	6.042
6.003					
6.041					
6.004	Done		Done	6.004	6.004

Not only do we carry with us current class, but also a record of everything we've taken

## **OOP- The Transcript**

specs:

a class that keeps track of all of your grades

starts empty upon entering MIT

ability to add class with its grade

ability to generate report, which prints each course taken and its grade

print method generates this report along with count of classes taken

class transcript:

```
def __init__(self):
```

```
def addCourse(self, course, grade):
```

```
def getReport(self):
```

```
def __str__(self):
```

## State Machines- The Degree Audit

```
course6degreeRequirements = ["6.01", "6.02", "6.002", "6.003", "6.004"]
```

```
class course6degreeAudit(sm.SM):
    #student starts out taking 6.01 with the complete list of courses to fulfill
    transcript = transcript()
    transcript.getReport()
    startState = course6degreeRequirements

    def getNextValues(self, state, inp):
        if inp == None:
            return (state, self.transcript)
        (currentCourse, grade) = inp
        remainingRequirements = state
        if currentCourse == "6.01":
            if grade == "A" or grade == "B" or grade == "C":
                currentCourse = "6.02"
                remainingRequirements.remove("6.01")
                self.transcript.addCourse("6.01", grade)
            elif currentCourse == "6.02":
                if grade == "A" or grade == "B":
                    currentCourse = "6.002"
                    print remainingRequirements
                    remainingRequirements.remove("6.02")
                    self.transcript.addCourse("6.02", grade)
                elif grade == "C" or grade == "D":
                    currentCourse = "6.042"
                    remainingRequirements.remove("6.02")
                    self.transcript.addCourse("6.02", grade)
            -----
            elif currentCourse == "6.004":
                if grade == "A" or grade == "B" or grade == "C":
                    remainingRequirements.remove("6.004")
                    self.transcript.addCourse("6.004", grade)
            newState = remainingRequirements

        self.transcript.getReport()
        return (newState, self.transcript)

    def done(self, state):
        remainingRequirements = state
        return len(remainingRequirements) == 0
```

```
>>> b= course6degreeAudit()
>>> b.transduce([(("6.01", "A"), ("6.02", "B"), ("6.002", "F"), ("6.002", "C"), ("6.041", "A"))])
Grade Report
A in 6.01
Grade Report
A in 6.01
B in 6.02
Grade Report
A in 6.01
B in 6.02
Grade Report
A in 6.01
B in 6.02
C in 6.002
```

## State Machine Combinators

### Non-Terminating

	Number Machines	Description	Input Vocabulary	Output Vocabulary
Cascade	2	Output of first machine is input to second	Input vocabulary of first machine	Output vocabulary of second machine
Parallel	2	Two machines run in parallel	Input vocabulary of first/second (same)	Pair of outputs from machine one, machine two
Parallel2	2	Two machines run in parallel	Pair of input to machine one, machine two	Pair of outputs from machine one, machine two
FeedbackSM	1	Output of machine fed back to input	None	Output vocabulary of machine
Feedback2	1	One input tied to output	Input vocabulary of machine	Output vocabulary of machine
FeedbackAdd	2	One machine on forward loop, second on feedback loop	Input vocabulary of machine one	Output vocabulary of machine one
Switch	2	Like Parallel, but input goes to only one machine	Input vocabulary of machine one or two	Pair of output of machines one and two
Multiplex	2	Same as switch, but we choose one of the two outputs	Input vocabulary of machine one or two	Output vocabulary of machine one or two

### Terminating

	Input	Description
Repeat	State machine, n	Repeats state machine n times
Sequence	List of state machines	Each machine in list is run to completion in sequence
RepeatUntil	State Machine, condition	Repeats state machine until condition is met

## Goal: To add HASS classes Requirement and Compute GPA

```
hassRequirements = ["spanish", "economics", "literature"]
```

How should we add HASS classes to our degree progress?

- Now we have two separate sets of requirements
- we now dont know the exact order of classes taken, what if we take Spanish between 6.01 and 6.02?
- we only want to feed our course6degreeAudit grades from course 6 classes!
- you can assume that students will not attempt to register for courses out of order

### Step One: HASS class SM

- takes as input a class and a grade
- maintains a transcript like course6degreeAudit
- if class is on list of required hass classes, it is removed if grade of A-D is achieved
- output is the transcript
- we are finished when all three required hass classes are completed

```
class hassDegreeAudit(sm.SM):
    startState = ?
    transcript = transcript()

    def getNextValues(self,state,inp):
        if inp == None:
            return (hassRequirements, self.transcript)
        (currentCourse, grade) = inp
        remainingRequirements = state

        ## Your Code here ##

        return ?

    def done(self, state):
        return len(state) == 0
```

## Step Two: Understanding Layout

How do we combine our `course6degreeAudit` and our `hassDegreeAudit`??

- input of combined SM will be course taken plus grade received
- course must be sent to correct state machine to be recorded
- we want to compute our combined GPA at the end !

Draw block diagram:

hint: you will need the two state machines we've built, plus two more !

## Step 3: Assigning Course to degree programming

```
class PureFunctionSM(sm.SM):
    def __init__(self, f):
        self.f = f
    def getNextValues(self, state, inp):
        return (state, self.f(inp))
```

Create a `PureFunctionSM` that will take as input (course, grade) and return a tuple of the either ((course, grade), None) or (None, (course, grade)) depending on whether it is a course 6 or hass class.

## Step Four: Calculating GPA !

Use a PureFunctionSM again to take output of two state machines (transcripts) and calculate GPA. Please attempt to use list comprehensions in doing so !

You may find the following helpful:

```
def convertToNumber(letter):
    if letter == "A":
        number = 5
    elif letter == "B":
        number = 4
    elif letter == "C":
        number = 3
    elif letter == "D":
        number = 2
    else:
        number = 0
    return number
```

```
def computeGPA((course6Transcript, hassTranscript)):
    ## Your code here ##
```

## Step 5: Combining

Write the code for a state machine that uses combinators and the parts we have built that takes in courses and their grades and outputs a GPA!

q =

```
>>> q.transduce([("6.01", "A"), ("spanish", "B")])
Grade Report
A in 6.01
```

```
Grade Report
B in spanish
```

```
[5.0, 4.5]
```

### **Further Extensions**

What if we wanted to add another category of classes with its own requirements (ie PE classes)?

Draw the block state machine diagram below?

Which new state machine combinator would we need? What existing state machine combinator is this similar to?