MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Fall Semester, 2007

Assignment 3, Issued: Tuesday, Sept. 18

To do this week

...in Tuesday software lab

1. Start writing code and test cases for the numbered questions in the software lab. Paste all your code, including your test cases, into the box provided in the "Software Lab" (Part 3.1) problem on the on-line Tutor. This will not be graded.

...before the start of lab on Thursday

1. Read the lecture notes.
2. Do the on-line Tutor problems for week 3 that are due on Thursday (Part 3.2).
3. Read through the entire description of Thursday's lab.

...in Thursday robot lab

1. Answer the numbered questions in the robot lab and demonstrate them to your LA.
2. Do the nanoquiz; it will be based on the material in the lecture notes and the on-line Tutor problems due on Thursday.

...before the start of lecture next Tuesday

1. Do the on-line Tutor problems for week 3 that are due on Tuesday (Part 3.3).
2. Do the lab writeup, providing written answers (including code and test cases) for **every** numbered question in this handout.

On Athena machines make sure you do:
`athrun 6.01 update`
so that you can get the `Desktop/6.01/lab3` directory which has the files mentioned in this handout.

- You need the file `fsm.py` for the software lab.
- You need the files `sequence.py` and `UtilityBrainTB.py` for the robot lab.

During software lab, if you are using your own laptop, download the files from the course Web site (on the Calendar page).

Object-Oriented Programming; Combining Sequential Behaviors

Tuesday's lecture, the Tuesday software lab, and the on-line problems cover Python's support for *object-oriented programming*. Thursday's lab applies the tools of object-oriented programming to develop a new system for defining and combining robot behaviors sequentially, rather than the utility-based combination we looked at last week.

Tuesday Software Lab: Practice

At the end of this lab, go to the on-line Tutor at `http://sicp.csail.mit.edu/6.01/fall07`, choose PS3, and paste all your code, including your test cases, into the box provided in the "Software Lab" problem. This will not be graded, it is simply a checkpoint for us to see how people are doing. Complete the assignment later and hand it in as part of your lab writeup. **Make sure that you include the test cases and results that you used to conclude that your code was working.**

This week, we'll get practice with object-oriented programming, and build some tools that will be useful in future labs.

Polynomial Arithmetic

We can represent a polynomial as a list of coefficients starting with the highest-order term. For example, the polynomial $x^4 - 7x^3 + 10x^2 - 4x + 6$ would be represented as the list `[1,-7,10,-4,6]`. Let's say that the *length* of a polynomial is the length of its list of coefficients.

Write a class called `Polynomial` that supports evaluation and addition on polynomials. You can structure it any way you'd like, but it should be correct and be beautiful. It should support, at least, an interaction like this:

```
>>> p1 = Polynomial([3, 2, 1])
>>> print p1
poly[3.0, 2.0, 1.0]
```

This represents the polynomial $3x^2 + 2x + 1$.

```
>>> p2 = Polynomial([100, 200])
>>> print p1.add(p2)
poly[3.0, 102.0, 201.0]
>>> print p1 + p2
poly[3.0, 102.0, 201.0]
>>> p1.val(1)
6.0
>>> p1.val(10)
321.0
```

The constructor accepts a list of coefficients, highest-order first.

Question 1. Define the basic parts of your class, with an `__init__` method and a `__str__` method (see the lecture notes for more information about this), so that if you do

```
>>> print Polynomial([3, 2, 1])
poly[3.0, 2.0, 1.0]
```

something nice and informative is printed out.

Question 2. Implement the `add` method for your class.

A cool thing about Python is that you can *overload* the arithmetic operators. So, for example, if you add the following method to your `Polynomial` class

```
def __add__(self, v):
    return self.add(v)
```

or, if you prefer,

```
def __add__(self, v):
    return Polynomial.add(self, v)
```

then you can do

```
>>> print Polynomial([3, 2, 1]) + Polynomial([100, 200])
poly[3.0, 102.0, 201.0]
```

Exactly what gets printed as a result of this statement depends on how you've defined your `__str__` procedure; this is just an example.

Question 3. Add the `__add__` method to your class.

A straightforward way to evaluate polynomials is to explicitly add up the terms $a_i x^i$. We can do this with list comprehension and `sum`. Hint: note that in a polynomial with k coefficients, the highest power of the variable is $k - 1$, for example, our example polynomial of length 3 has the highest power of x^2.

Question 4. Add the `val` method to your class, which evaluates the polynomial for the specified value.

Try to do things as simply as possible. Don't do anything twice.

Finite State Machines

This section builds on the FSM discussion in lecture. All the code from lecture can be found in the `fsm.py` file. There is additional written material covering the lecture presentation at the end of this handout.