MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Final Review – Python practice problems

(A) This problem is taken directly from nanoquiz 2. It is worth going over because many students had problems with it.

Main points:

- **Keep track of the exact *type* of objects you deal with in Python.**
- **Robot non-deterministic behavior: a function mapping *senors* → *set of actions***

Here are some procedures:

```
def a():
    return set([stop, go])

def b(sensors):
    return set([stop, go])

def c(limit):
    def f(sensors):
        if sensors[0][1] > limit:
            return set([stop, left, right])
        else:
            return set([go])
    return f
```

For each of the following Python expressions, indicate whether the **value** of the expression is a non-deterministic behavior (as described in the assignment handout) and, if not, briefly why not.

(1) `a`
(2) `a()`
(3) `b`
(4) `b(1.2)`
(5) `b([sonarDistances(), pose()])`
(6) `b(1.2)([sonarDistances(), pose()])`
(7) `c`
(8) `c(1.2)`
(9) `c([sonarDistances(), pose()])`
(10) `c(1.2)([sonarDistances(), pose()])`

(B) State machine practice: A "Smoothing state machine"

Main points:

- **Key in designing state machines: what is the *state*? → what information must the object store from step to step?**
- **Understand the basic methods for combining SMs, TSMs**

(a) Write a procedure `makeSSM(N)` that takes one argument, N, and returns a state machine which computes a "moving average" of the N latest input values.

How much storage does your solution use? How much computation does it require on each step? Are there alternative designs?

(b) Say you're given a `SSM` object which you cannot modify, and you're asked to create a state machine that computes the running *sum*. Design a simple solution using *SerialSM*. (don't worry about delays).

(c) Write a procedure that takes two arguments, N, M, and works like `makeSSM`, but the resulting SM terminates after completing M steps.

(C) Object-oriented programming

Main points:

- **Be comfortable with the general idea of OOP, classes, inheritance, etc.**
- **Understand some of Python's specifics for dealing with OOP. (__init__, __str__, __add__, etc)**

(a) Create a `Point` class for dealing with geometric coordinates of arbitrary dimension. It should handle the following:

```
p1 = Point([3,2])     # a point with 2-dimensions
p2 = Point([-4,1])
p3 = Point([3,2,1])   # a point with 3-dimensions

print p1     # Know how to customize string representation
p1+p2        # Know how to ''overload'' operators, such as ''+''

p1 + p3
ERROR: Can't add points of different dimension.

p1.distance(p2)   # return distance between points
p1.distance()     # Can you make the default be dist to origin?
```

(b) Create a subclass `PointTC`, which differs only in how distances are computed, here using the "taxi-cab metric", which in two dimensions is $d = |x_1 - x_2| + |y_1 - y_2|$.

(c) Using your `makeSSM` from the last question, can you transduce a list of `Point` objects to result in a list of `Point` objects along a "smoother" path? Why or why not? (Depending on your implementation, the answer to this question may involve knowing Python topics which we didn't really cover in this course. So, you don't need to able to solve this question, but it might be useful to think about what are the potential issues...)

(D) Practice with Procedures

Main points:

- **Know how to translate a description of a function into Python code.**

(a) We can represent n-dimensional vectors as Python lists: `[x1, x2,...,xn]`. Write a procedure that takes two vectors as inputs and returns their dot product.

(b) Write a procedure that calculates the angle (in radians) between two vectors.

(E) Abstracting Functions

Main points:

- **Be able to see the common aspects of different operations and abstract the procedures into more generic ones.**

(a) Write a procedure that calculates the arithmetic mean of a list of numbers.

(b) Write a procedure that calculates the geometric mean of a list of numbers.

(c) Now, abstract the operation of calculating a mean. Write a procedure `makeMean` that returns a procedure that takes a list and returns a mean of the list. The procedure should be able to deal with different kinds of means, e.g. arithmetic, geometric, harmonic, quadratic, etc.

```
>>> arithMean = makeMean(...)
>>> arithMean([1,2,3])
2.0
>>> geoMean = makeMean(...)
>>> geoMean([1,2,3])
1.8171205928321397
```

Hint: *Break the operation of calculating a mean down into parts: first, you operate on each term, then you combine those parts somehow, then you do something to that combination to get the final mean. For example, for the arithmetic mean, the operation on the terms is just the identity, the combination operation is addition, and the final operation on the combination is division by the number of terms.*