MATLAB Assignment 2

When working on this assignment you might want to take a look at MATLAB code that was developed by students who took ME451 in previous years. The students back then did not come up with identical solutions. Take a look at their solutions and develop your own.
2010: http://sbel.wisc.edu/Courses/ME451/2010/SimEngine2D/index.htm
2011: http://sbel.wisc.edu/Courses/ME451/2011/SimEngine2D/index.htm

Turning in your assignment: place all your files in a directory called "lastName_Matlab_02", zip that directory, and upload the resulting file "lastName_Matlab_02.zip" in the appropriate Dropbox Folder at Learn@UW.

Problem 1. Implement Matlab code that opens a file, called "input.acf" (`acf` stands for "analysis control file") and parses the JSON data below in order to generate all the information required to define a simulation.

```
{
  "simulation": "Kinematics",
  "tend": 20.0,
  "stepSize": 0.01,
  "outputSteps": 400
}
```

The ACF file that contains the text above indicates that your simEngine2D Matlab code will be expected to run a kinematics analysis, with step size of 0.01 seconds, end the analysis after 20 seconds and report information at 400 intermediate station points.

What do you need to do? Write a Matlab script that parses the file "input.acf" and stores the values 'Kinematics' as a string in the variable `simulation` and the numbers 20.0, 0.01, and 400 in the Matlab variables `tend`, `stepSize`, and `outputSteps`, respectively.

Notes:

- To load a JSON file, use the Matlab function `loadjson` provided by the package `jsonlab` which can be downloaded from the course website at
  http://sbel.wisc.edu/Courses/ME451/2014/MATLAB/jsonlab_0.9.0.zip
  Place this file in your working directory (or in your Matlab *userpath*, if you have that set up). Type `help loadjson` at the Matlab prompt for help on using the `loadjson` function.
- The JSON format does not provide for comments. However, we will keep the names of the *keys* in the *key-value* pairs as meaningful and verbose as needed to be self-explanatory.
- For more information on the JSON format, see www.json.org

Problem 2. Implement Matlab code that opens a file, called "model2.adm" (`adm` stands for "analysis data for the model") and parses the JSON data below in order to generate all the information required to fully characterize, for the purpose of 2D Kinematics and/or Dynamics analysis, a model consisting of 2 rigid bodies and 4 constraints (one of each of the following types: *AbsoluteX*, *AbsoluteY*, *AbsoluteAngle*, and *AbsoluteDistance*).

What do you need to do? The zip file that you submit should contain a function m-file, named `runme.m` which defines a function that takes two integer arguments. When invoked from the command line, for example using:

```
>> runme(bodyID, constraintID);
```

it should print out all information relevant for the body whose ID is `bodyID` and for the constraint whose ID is `constraintID`.

Notes:

- You can create the ADM file "model2.adm" yourself by copying and pasting the data at the end of this file or you can get it from the course website (`http://sbel.wisc.edu/Courses/ME451/2014/`)
- The model described in this ADM file is not meaningful (that is it does not describe a meaningful mechanism). It is used here only for the purpose of this assignment which focuses on parsing the ADM file.
- For now, store the information contained in the ADM file in whatever Matlab data structures you consider appropriate. As we progress with the class and with the Matlab assignments, you may have to revisit whatever decisions you make right now.
- The JSON parser function `loadjson` will return array data (for example `q0` and `qd0` in the definition of a body, or `sP1` in the definition of an *Absolute X* constraint) as row vectors. To make your life easier later on, I strongly suggest to store them as column vectors in the corresponding Matlab variables.
- When you write the code to parse the constraint specifications and you handle the `fun` property, keep in mind that this should be a function of time. To deal with it you'll have to recall what you did in the previous Matlab assignment to read in a string and convert that to a function that can later on be evaluated for various values of time t.
- All constraints will have the `name` and `id` properties defined (so that you can identify that particular constraint in the model). Similarly, they will all have the property `type`. The values for `type` will be reserved keywords (so far, we've seen `AbsoluteX`, `AbsoluteY`, `AbsoluteAngle`, and `AbsoluteDistance`). The remainder of the properties for a given constraint depend on its type.
- Besides the lists of bodies and constraints, an ADM file also defines two model-level properties: a `name` for the model itself (which may be used in post-processing when reporting analysis results) and `gravity` which provides the value of the gravitational acceleration vector.

```
{
    "name": "My model",
    "gravity": [0, -9.81],
    "bodies": [
        {
            "name": "first_body",
            "id": 1,
            "mass": 2,
            "jbar": 0.3,
```

```
            "q0": [2, 0, 0],
            "qd0": [0, 0, 0]
        },
        {
            "name": "second_body",
            "id": 2,
            "mass": 3,
            "jbar": 0.2,
            "q0": [7, 0, 0],
            "qd0": [0, 0, 0]
        }
    ],
    "constraints": [
        {
            "name": "abs_x",
            "id": 1,
            "type": "AbsoluteX",
            "body1": 1,
            "sP1": [-2, 0],
            "xGround": 4,
            "fun": "0.1*t + 1/9"
        },
        {
            "name": "abs_y",
            "id": 2,
            "type": "AbsoluteY",
            "body1": 1,
            "sP1": [-2, 0],
            "yGround": 3.5,
            "fun": "0"
        },
        {
            "name": "abs_angle",
            "id": 3,
            "type": "AbsoluteAngle",
            "body1": 2,
            "fun": "0.1 * t + 1/9"
        },
        {
            "name": "abs_dist",
            "id": 4,
            "type": "AbsoluteDistance",
            "body1": 2,
            "sP1": [2.1, 1.3],
            "Pground": [0.65, -1.3],
            "fun": "t^2 + 3*t + 1"
        }
    ]
}
```