Assignment 3 (100 points)

In this assignment, you will develop three Python scripts: one will simulate a non-preemptive EDF scheduler, one will simulate (preemptive) RM, and one will simulate (preemptive) DM. If desired, a visualizer is provided, as is a sample `EDF.py` solution from Assignment 2. The focus of Assignment 2 was on working out all the details that must be known by a scheduler at any given time to know which job to schedule. This included details like checking if the job had been released, checking if a job had completed all of its computation, and comparing deadline to see which job (of those valid to consider for that moment in time) had the earliest deadline. In contrast, the focus of Assignment 3 is building on this framework to make small changes to how the job to execute is chosen, and then comparing the different schedulers. Therefore, a significant portion of the assignment is coming up with task sets that highlight the differences and sharing your reasoning of which checks need to change between the implementations.

Part 1: Setting up your workspace.

In your home directory on the server, make a new directory called `HW3`. All of the files you intend to submit must be in this folder. Provided files are in `/home/shared/HW3/`. There is also a script posted that, if you comment in the relevant parts, can help speed your testing. To run the script, type:

```
./testing.sh
```

If it complains that you do not have the correct permissions, change the permissions:

```
chmod +x testing.sh
```

You should only need to change the permissions once.

Part 2: The assignment.

A - Creating testing data (36 points)

For each of the following, create a task set of periodic tasks which showcases the differences between preemptive and non-preemptive EDF. Pick and include a `max_t` that shows at least two complete job executions of each task under either scheduler. Make sure that at least 3 executions change between the two schedulers. Describe the differences in `README.txt`.

1. `test1.json` - preemptive EDF vs. non-preemptive EDF. Include five tasks in the task set for this comparison.
2. `test2.json` - preemptive EDF vs. preemptive RM. Include four tasks in the task set for this comparison.
3. `test3.json` - preemptive RM vs. preemptive DM. Include five tasks in the task set for this comparison.

Grading:

- For each test:
  - (2 points) Correct number of tasks.
  - (2 points) Readable and correct formatting of JSON. (That is, spacing within file allows easy reading of each task, and the file follows formatting of tasks expected by python scripts from the first two assignments.)
  - (4 points) A description in `README.txt` of how the schedules produced by the two schedulers are different. Point out at least three executions that happen at different times between the two schedulers.
  - (4 points) The tasks themselves, that they follow the instructions (are periodic, max_t chosen well, etc.) and should cause the described differences.

B - Non-Preemptive EDF (14 points)

Most implementations of preemptive EDF from Assignment 2 shared the following structure:

- Create all jobs of the given tasks that might execute in `t=0` to `t=max_t`.
- Start at time `t=0` and loop until `t=max_t`
  - Pick a minimum deadline.
  - Loop through all jobs to find the earliest deadline.
    - Update the minimum deadline if necessary. Only compare jobs that have been released at time `t`.
  - Add an execution of one time unit of the job with the minimum deadline found above. Do this by either adding a new execution or updating the end time of the previous execution if it was of the same job.
  - Update `t` with `t=t+1`.
- Output the tasks, executions, and `max_t` to the output file.

1. Based on the general structure of producing executions for a preemptive EDF scheduler (like those listed above or the steps that your implementation had), which portion will need to change to make it a non-preemptive EDF simulation? Explain what you will change and why in `README.txt`.
2. Implement a non-preemptive EDF schedule simulator in a file called `nonPreemptiveEDF.py`. This script should handle input and output in the same way that `EDF.py` does.

Grading:

- (4 points) Explanation in `README.txt` of what to change and why.
- (10 points) Implementation of `nonPreemptiveEDF.py`.

C - Rate Monotonic (RM) (20 points)

1. Based on the general structure of producing executions for a preemptive EDF scheduler (like those listed above or the steps that your implementation had), which portion will need to change to make it a preemptive RM simulation? Explain what you will change and why in `README.txt`.
2. Implement a RM schedule simulator in a file called `RM.py`. This script should handle input and output in the same way that `EDF.py` does. Feel free to implement this by using either your `EDF.py` or the provided sample implementation as a baseline.

Grading:

- (5 points) Explanation in `README.txt` of what to change and why.
- (15 points) Implementation of `RM.py`.

D - Deadline Monotonic (DM) (14 points)

1. Based on the general structure of producing executions for a preemptive RM scheduler, which portion will need to change to make it a preemptive DM simulation? Explain what you will change and why in `README.txt`.
2. Implement a DM schedule simulator in a file called `DM.py`. This script should handle input and output in the same way that `RM.py` does.

Grading:

- (4 points) Explanation in `README.txt` of what to change and why.
- (10 points) Implementation of `DM.py`.

E - Testing (16 points)

1. Test your new scheduling simulations with the task sets that you developed in the first part of this assignment. Produce an image for each schedule, naming it based on the test and the scheduler. This should result in the following files:

- `test1_EDF.pdf`
- `test1_np-EDF.pdf`
- `test2_EDF.pdf`
- `test2_RM.pdf`
- `test3_RM.pdf`
- `test3_DM.pdf`

2. Look at these files. Do they produce the differences you expected and noted in Part A? If not, find what went wrong and update either your task set or your implementation.

Grading:

- (6 points) Each file exists.
- (10 points) Schedules reflect the descriptions in `README.txt`.

Part 3: Submitting your assignment.

Your assignment should be in `/home/<yourCSlogin>/HW3`. I will collect homework from there and check the last time that each file was modified. This assignment is due on Oct. 8, 2018 at 9:05am EST. If the files have been edited after that time, I will assume that you have chosen to use one or more late days. If you would like to continue tweaking your solution, do so in a different folder. Make sure that the following files are in `/home/<yourCSlogin>/HW3`, especially if you were working in a different directory or on a different machine:

- `test1.json`
- `test2.json`
- `test3.json`
- `README.txt`
- `nonPreemptiveEDF.py`
- `RM.py`
- `DM.py`
- `test1_EDF.pdf`
- `test1_np-EDF.pdf`
- `test2_EDF.pdf`
- `test2_RM.pdf`
- `test3_RM.pdf`
- `test3_DM.pdf`

In order to grade your Python scripts, I will use your tests along with a set of tests that I develop. If any script fails one of the tests, I will look at the implementation to award partial credit, so make sure your code is readable (clear variable names, relevant comments, etc.).