Python Exercises III

Write your code for both exercises in a single Jupyter notebook.

Exercise 1: Light curve simulator

Variable stars show periodic changes in brightness over time. If you observe these stars at a regular cadence (like every 5 minutes), you can plot a light curve (brightness vs time), which will show a quasi-sinusoidal shape (for certain types of variable stars).

Write a function called `simulate_light_curve` that accepts the following arguments:

1. `period` (float): The period of the variable star (in days)
2. `amplitude` (float): The amplitude of the brightness variations
3. `duration` (float): The total time span of the simulation (in days)
4. `cadence` (float): The time interval between observations (in days)
5. `noise_level` (float, default=0.0): The standard deviation of Gaussian noise to add

Inside the function, first generate a `numpy` array of time values spanning the specified duration with the given cadence. Then use `np.sin()` and the given period and amplitude to generate simulated observations at each time value. If the noise level is not the default value of 0, generate Gaussian random noise with the specified standard deviation and add it to your simulated brightness measurements. Finally, return both the time array and the simulated brightness array.

Demonstrate that your function works by generating and plotting several different light curves. Use `matplotlib` to create and format your plots.

Exercise 2: Image processing issues

Download the data file `issues.txt` from the Courseworks assignment. This file is a log from an automated pipeline that runs every night to process data from the Condor Array Telescope in New Mexico. The pipeline processes images in 10-minute chunks, recording any issues with the images in the log file that you're working with. This file consists of three types of lines.

Type 1 lines (example shown below) list the date and time at the beginning of each 10-minute interval. (The T separates the date from the time, and the Z at the end can be ignored.) All lines of this type start with the > symbol.

```
>>>>>>>>>>>>>>> 2024-10-24T00:00:00Z <<<<<<<<<<<<<<<
```

Type 2 lines (example shown below) start with a string describing an issue that the processing algorithm encountered. (Note that these strings don't always start with "system"!) The string is

followed by a colon and a number that describes some aspect of the detected issue. One line is printed for each image that the processing algorithm rejects in a given 10-minute chunk. Sometimes, no images are rejected, so a given type 1 line might be followed by 0 type 2 lines.

```
system_autocorr_fwhm: 17.948531456139616
```

Type 3 lines (shown below) start with a number and end with “images satisfy the suitability criteria…” The number indicates how many “good” images that the algorithm identified in a given 10-minute interval. A type 3 line is printed for every time interval, regardless of the outcome. Note that if the number is 0, but no type 2 lines were printed, that means that no images were taken in that time interval.

```
18 images satisfy the suitability criteria...
```

Use Python’s built-in `open()` function to read `issues.txt` line-by-line and complete the following tasks:

1. Count the total number of images that satisfied the suitability criteria across the entire time period covered by the log. You might find the `.isdigit()` string method useful.

2. Create a dictionary where the keys are unique issues that the algorithm encountered (the strings printed at the beginning of type 2 lines) and the values are a count of the number of times the issue occurred over the entire time period covered by the log.

Hint: to skip lines when you’re looping through a file, you can create an `if` statement that catches the lines you want to skip, then put the reserved keyword `continue` as the only line in the body of the `if` statement. `continue` will immediately fast forward to the next iteration of the loop, skipping any code below it. Check out section 4.11.3 of the textbook to learn more.

Optional bonus challenge: Create a dictionary where the keys are a string representing the time and date (using the format that’s already in the log, YYYY-MM-DDT00:00:00, is fine) and the values are lists of unique issues that were encountered on that date.