Module 3 Homework: Running Times Analysis

Part 1 - Finding the missing element

You will create four functions in the file `FindMissing.py`. Each function should take a list as an input and return an integer. The input list of length `n` will be a permutation of all the integers from `0` to `n` (both included), with one of the integers missing. The functions should output the integer missing from the input list.

Desired behavior in an interactive shell:

```
>> find_missing([2, 5, 3, 1, 6, 0])
4
>> find_missing([3, 6, 4, 1, 5, 2])
0
>> find_missing([5, 1, 3, 0, 2, 4])
6
```

To test the functions, we will generate a randomized permutation with one element missing.

Use your code from the lab in the files `GenerateFigs.py` and `TimeFunctions.py`.

`find_missing_1`

For our first attempt, we will use the membership operator, `in`. The operator returns `True` if the specified element is present in an object (the list). There is also a negated version, `not in`. In our case, the missing element is unknown, so you need to iterate over all the integers and test each one of them for membership. Once the missing element is identified, the function should return it.

- Write a function `find_missing_1(L)` following the idea above.
- Modify `GenerateFigs.py` so that it:
  - Generates a figure with **21** data points evenly spaced between 0 and 1000 on the x-axis
  - Generates data to test your function
  - Includes labels for the x- and y-axes
  - Saves the figure as `find_missing_1.png`

`find_missing_2`

Another approach would be to sort the list and then iterate over the elements so as to identify the missing integer. We will spend a few weeks on sorting, but let's for now just use the sorting method available for a list `L`

```
L.sort()
```

However, sorting takes time. Does the time saved on searching for the missing element make up for the time spent sorting? In order to find out, we will plot the running times alongside those from our first attempt.

- Write a function `find_missing_2(L)` following the idea above.
- Modify `GenerateFigs.py` so that it:
  - Plots the running times of `find_missing_1` and `find_missing_2` in the same figure
  - Saves the figure as `find_missing_12.png`

`find_missing_3`

To do even better, we are going to use some math. Remember the formula for the sum of the first `n` numbers?

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

We will evaluate this formula and compare the result to the sum of the elements in the list. The two (different) values obtained can now be used to identify the missing element.

- Write a function `find_missing_3(L)` following the idea above.
- Modify `GenerateFigs.py` so that it:
  - Plots the running times of `find_missing_2` and `find_missing_3` in the same figure
  - Saves the figure as `find_missing_23.png`

`find_missing_4`

After we finish the sorting section of this course, we will introduce a technique whose beauty we will finally be able to understand: **hashing**. For now, we will demonstrate just how fast hashing is to whet our appetites. Sets use hashing to store items. You can convert a list `L` to a set using the command

```
set(L)
```

Because of its implementation, the set has much faster membership verification than the list.

- Write a function `find_missing_4(L)` following the idea above.
- Modify `GenerateFigs.py` so that it:
  - Plots the running times of `find_missing_3` and `find_missing_4` in the same figure
  - Saves the figure as `find_missing_34.png`

Part 2 - Creating high quality figures

`find_missing_3` and `_4` are so much faster than `_1` and `_2` that we need to use much larger collections to begin to compare them. We should present our results using two figures, so we can compare quickly and clearly discren how the functions compare to each other.

- Modify `GenerateFigs.py` to generate three figures:
  - `find_missing_1.png`: `find_missing_1`, ranging from 0 to 1000 items. Use 21 data points.
  - `find_missing_12.png`: `find_missing_1` and `_2`, ranging from 0 to 1000 items. Use 21 data points.

    - `find_missing_23.png`: `find_missing_2` and `_3`, ranging from 0 to 1,000,000 items. Use 21 data points.
    - `find_missing_34.png`: `find_missing_3` and `_4`, ranging from 0 to 1,000,000 items. Use 21 data points.

It is important that the plots you generate be quickly readable to a stranger, not just yourself. **Being able to quickly and clearly present data to a layperson is fundamental to being a good scientist.**

Make `find_missing_12`, `find_missing_23`, and `find_missing_34` better figures:

- Use different colors and styles for every dataset
- Change your axes scales so that numbers are between 0.1 and 1000. Note any changes in scale using either units or an explicit equation on that axis.

You will probably have to dig into `matplotlib` documentation (link) to figure out how to do all this. This is intentional: we cannot cover the entirety of `matplotlib` in this course, but if you learn how to use the documentation, you can figure out whatever you need down the road. As always, feel free to ask questions on Piazza if you get stuck.

Submitting

At a minimum, submit the following files:

- `FindMissing.py`
- `TimeFunctions.py`
- `GenerateFigs.py`
- `find_missing_1.png`
- `find_missing_12.png`
- `find_missing_23.png`
- `find_missing_34.png`

Students must submit to Mimir **individually** by the due date (typically, the second Wednesday after this module opens at 11:59 pm EST) to receive credit.

Grading

- 10 - `find_missing_1`
    - 5 - automatic test of functionality
    - 5 - manual test of correct implementation
- 10 - `find_missing_2`
    - 5 - automatic test of functionality
    - 5 - manual test of correct implementation
- 10 - `find_missing_3`
    - 5 - automatic test of functionality
    - 5 - manual test of correct implementation
- 10 - `find_missing_4`
    - 5 - automatic test of functionality
    - 5 - manual test of correct implementation
- 15 - `find_missing_1.png` manual grading

    - 5 - 21 data points
    - 5 - x-axis label
    - 5 - y-axis label
- 45 - `find_missing_12.png`, `find_missing_23.png`, and `find_missing_34.png` manual grading
    - 12.5 - 4 different colors
    - 12.5 - 4 different styles
    - 10 - 4 axis labels
    - 10 - 4 axis scales (i.e. the magnitudes of the numbers)

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on several assignments (particularly in the recursion module)