Module 3 Lab: Running Times and `matplotlib`

Hopefully, you have some abstract idea about modeling the running times of algorithms at this point. In this lab, we will attempt to find concrete data to validate these models. We have a few goals in this lab:

- Generate running time vs input size data to validate running time models
- Present data clearly and professionally
- Ensure we can download and install third party python packages (e.g. Matplotlib)

Installing and using Matplotlib is a friction point for a lot of students, as is modfiying plotting parameters. It also helps us meet two of the core outcomes for the Computer Science program at UConn ([link])(https://www.cse.uconn.edu/undergraduate/major-programs/computer-science/abet-cs/):

> *Practice and grow as computing professionals, conducting research and/or leading, designing, developing or maintaining projects in various technical areas of computer science.*

We want you to be leaders in software development, and being able to install and use third party packages is crucial to this.

> *Communicate effectively in a variety of professional contexts.*

This is a skill where a lot of students struggle - we will focus on some core data plotting guidelines to improve communication in this lab.

Timing Functions

Write a function `time_function(func, args)` that returns the number of seconds to run `func` with `args`.

`TimeFunctions.py` contains two functions that should have a running time ratio of ~10x to help test your function.

Starter code

```
if __name__ == '__main__':
    def test_func(L):
        for item in L:
            item *= 2

    L1 = [i for i in range(10**5)]
    t1 = time_function(test_func, L1)

    L2 = [i for i in range(10**6)]
    t2 = time_function(test_func, L2)
```

```
    print("t(L1) = {:.3g} ms".format(t1*1000))
    print("t(L2) = {:.3g} ms".format(t2*1000))
```

Expected behavior in terminal:

```
$ python3 TimeFunctions.py
t(L1) = 4.99 ms
t(L2) = 51.9 ms
```

When running code in terminal, remember:

- The `$` denotes a generic terminal prompt. You do not need to type it.
- You may need to use `python` or `python3` to run scripts depending on how Python is installed on your computer. The Mimir IDE requires `python3`.

Creating a simple plot

`matplolib` is widely used by the scientific community to generate plots with Python. Familiarity with this module will help make you a better data scientist.

We will use the `matplotlib` package in this assignment. You can find information on installing `matplotlib` here. Alternatively, you can work in the Mimir IDE (there is a button "Open in Mimir IDE" on the Mimir page for this assignment). Click here for documentation on the Mimir IDE.

Most of the plotting functionality of `matplotlib` is in the attribute `pyplot`, which is commonly imported with the alias `plt`. Below is starter code for a common three step process:

- create a figure
- add a scatter plot (data points) to that figure
- save that figure

```
from matplotlib import pyplot as plt     # import plotting functionality
plt.figure()                             # create a blank figure
plt.scatter(x, y, c='r', marker='x',     # add scatter plot
            label='has_duplicates_1')
plt.savefig('starter_fig.png')           # save figure
```

In the code above, `pyplot` generates a series of data points from the collections `x` and `y`, which must contain the same number of items `n`:

```
(x[0], y[0]), (x[1], y[1]), (x[2], y[2]), …, (x[n-1], y[n-1])
```

The starter code in `GenerateFigs.py` should generate the provided figure `starter_fig.png` when run after finishing Part 1 of this assignment.

Modify `GenerateFigs.py` so that it:

- Generates a figure with **21** data points evenly spaced between 0 and 1000 on the x-axis.

- Includes labels (with units) for the x- and y-axes.
- Saves the figure as `fig_1.png`

Checking for duplicates

Add several functions to `Duplicates.py`. Each function should take a list as an input and return a `bool`. Return `True` if the list contains any duplicate values; otherwise, return `False`.

has_duplicates_1

```
def has_duplicates_1(L):
    n = len(L)                          # number of items in list
    for i in range(n):                  # for each item
        for j in range(n):                  # compare it to every other it
            if i!= j and L[i] == L[j]:
                return True
    return False
```

This function is provided for you. It is the typical first pass approach at this problem: comparing every item in the list to every other item in the list.

has_duplicates_2

Unfortunately, `has_duplicates_1` performs many redundant comparisons.

- First outer loop: the item at index 0 is compared to the items at 0, 1, 2, 3, …, 9
- Second outer loop: the item at 1 is compared to items at 0, 1, 2, 3, …, 9
- Third outer loop: the item at 2 is compared to items at 0, 1, 2, 3, …, 9
- …
- Tenth outer loop: the item at 9 is comparted to items at 0, 1, 2, 3, …, 9

Can you spot the problem? By the time we get to the tenth loop, the element at index 9 has already been compared to the elements at indices 0, 1, 2, 3, 4, 5, 6, 7, and 8. Every comparison in that loop is redundant.

In the ninth loop, the element at index 8 has already been compared to elements at 1, 2, 3, 4, 5, 6, and 7. All but one comparison is redundant.

There is a common trick to eliminate these redundancies that cuts the number of comparisons roughly in half. Take a minute to see if you can figure it out. If not, check running time chapter of the textbook for inspiration.

- Implement a function `has_duplicates_2` using the trick hinted at above.

Creating high quality figures

Our goal in presenting data is to make it as easy to understand as possible. The default parameters in most plotting software (including matplotlib and excel) do not do a great job

at this, so we will require some adjustments. Some easy steps that will make your data easier to understand:

- Start your axes at or near 0 unless you have a really, really good reason not to. This will ensure that when one data point looks twice as big as another, it really is (see e.g. Truncated Graphs).

- Use "nice" numbers on your axes

    - use factors of 10 as your increments (1s, 10s, 100s, 200s, ...).
    - keep your numbers between 1 and 1000. Avoid smaller (e.g. 0.1) or bigger (e.g. 10000) numbers, chaning the scale if you have to.
    - This seems small, but it's huge. If you do this, it will take viewers less time and energy to internalize your data, making them that much more likely to listen to what you have to say.

- Use colors and shapes to differentiate different data sets on the same figure - this will make it easier for a viewer to quickly distinquish data. Avoid using red and green on the same plot.

Modify `GenerateFigs.py` to generate a high quality figure:

- plot results from `has_duplicates_1` and `_2` on the same figure
- Use 21 data points ranging from 0 to 1000 items for your x-axis
- Use different **colors** and **markers** for both datasets
- Change your axes' scales so that numbers are between 1 and 1000 (except for the starting 0)
    - You may have to manually scale all of the numbers in your x- and y- lists to do this. If you do, make sure to modify your units appropriately (e.g. if you multiply all your times by 10**6, and your times were in seconds, your resulting unit will be microseconds).
- save your figure as `dups.png`
- Labels the data sets, either through a legend or by adding unambiguous text blocks to the plot (see the example `dups.png` provided)

You will probably need 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 Discord if you get stuck.

A sample of the final `dups.png` is provided to illustrate the above guidelines. Note that the y-axis is in increments of 10, and the x-axis is in increments of 200.

Submitting

At a minimum, submit the following files:

- `TimeFunctions.py`
- `Duplicates.py`
- `GenerateFigs.py`
- `fig_1.png`

- `dups.png`

Students must submit to Mimir **individually** by the due date (typically, Sunday at 11:59 pm EST) to receive credit.

Grading

This lab is manually graded after the deadline.

- 5 - `time_function` is correct
- 5 - `has_duplicates_2` is correct
- 30 - `fig_1.png`
  - 10 - 21 data points
  - 10 - x-axis label
  - 10 - y-axis label
- 5 - `has_duplicates_2` is correct
- 60 - `dups.png`
  - 15 - 2 different marker colors
  - 15 - 2 different marker styles
  - 15 - 2 correct axis labels (including units)
  - 15 - 2 correct axis scales
    - values are correct
    - units are chosen to give an appropriate magnitude

Feedback

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

We check this feedback regularly, and it has resulted in many improvements.