Module 3 Lab: Running Times and `matplotlib`

Part 1 - Timing Functions

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

Desired 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`.

Part 2 - 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`. To create a figure, add a scatter plot, then save that figure, use the following commands:

```
from matplotlib import pyplot as plt
plt.figure() # creates a figure object to add curves on
plt.scatter(x, y, c='r', marker='x', label='has_duplicates_1') # adds scatter plot
to current figure
plt.savefig('starter_fig.png') # saves figure to the name provided
```

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 for the x- and y-axes. **You're a scientist. Your axes' labels should have units.**
- Saves the figure as `fig_1.png`

Part 3 - Checking for duplicates

Add several functions to `Duplicates.py`. Each function should take a list as an input and return a `bool`:

- List has duplicate values: return `True`
- List has no duplicate values: return `False`

`has_duplicates_1`

```
def has_duplicates_1(L):
    n = len(L)
    for i in range(n):
        for j in range(n):
            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.

Part 3 - Creating high quality figures

Modify `GenerateFigs.py` to generate a better figure:

- plot results `has_duplicates_1` and `_2` on the same figure
- Use 21 data points ranging from 0 to 1000 items
- Use different colors and markers for both datasets
- Change your axes scales so that numbers are between 0.1 and 1000. Note any changes in scale using either units in that axis's labels or an explicit equation on that axis.
- save your figure as `dups.png`

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 Piazza if you get stuck.

A sample of the final `dups.png` is provided to illustrate the above guidelines.

(Submission and grading instructions on next page)

Submitting

At a minimum, submit the following files:

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

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

Grading

- 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 - `has_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. 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)