Major 4/5
Replacement Assignment

Due: March 27, 2020 @ 11:59pm via Avenue ***(42 points available)***
Late submissions will not be accepted.
Grading: 13 % (this will replace the weighting of Major 4 and Major 5 labs and Minor 6 lab)

Objectives:

To apply the concepts taught in lecture and tutorial, including but not limited to:

- Algorithm design
- File Processing
- Matplotlib
- Functions
- Loops (for loops, while loops)
- Conditionals (if, elif, else)
- Boolean expressions
- Classes and Objects
- Exception Handling
- Testing and Debugging

Completion Instructions:

- This assignment is **to be completed individually**
- We have supplied you with a template for both your non-code and code answers (a word document template and a Python notebook (.ipynb) template file). **Use these documents to complete your assignment.**
- Python3 Code may be written and tested using Jupyter Notebooks or jhub.
- Be sure to read all the requirements/instructions!

Submission Instructions:

- **Submissions will be screened for plagiarism**
- Completed assignments must be submitted through Avenue by the due date.
  You must submit two documents:
  1. A PDF of your non-code answers (the filled-in student template titled **YourMACID_TakeHomeResponses.pdf**)
  2. The completed template python notebook, titled **YourMACID_TakeHomeCode.ipynb**

What to Know Beforehand

Yes, this assignment is a little longer and slightly more complex than a typical Major lab. But, keep in mind that you have a week to complete it, and you have access to all your notes, the course material (lectures/tutorials) and the textbook (woohoo!). So, take it one step at a time, consult the course material if necessary, and you'll be more than capable of completing it!

Good luck, you got this!

Problem 1: Estimating Nanoparticle Size (20 marks)

If you work in research and do anything related to nanoparticle synthesis – that is, the synthesis of nanoscale particles (on the order of 1 to ~100 nanometers (nm)) – you might encounter the challenge of determining your material's average particle size. Transmission Electron Microscopy (TEM) is one method of determining particle size. Once an image (or a set of images) is obtained, image processing algorithms can be applied to get an idea of the particle size distribution and therefore the average particle size. However, the problem is that TEM is not always available for this type of analysis.

An alternative estimation of nanoparticle size (or, more correctly, crystallite size), τ, is through the use of X-ray Diffraction (XRD – familiar from Materials 1M03 perhaps?) and the Scherrer Equation:

$$\tau = \frac{K\lambda}{\beta cos\theta}$$

Where K is a constant (K = 0.9), λ is the X-ray wavelength in nm (λ = 0.07107 nm for this problem), β is the peak width at half-maximum from a plot of intensity versus 2θ (Δ(2θ) in radians), and θ is the Bragg angle (the diffraction angle) of the peak in radians.

XRD gives a plot of intensity versus diffraction angle, which is primarily used to identify the composition of crystalline materials. However, according to the Scherrer equation above, XRD plots can also be used to estimate the average nanoparticle size, τ, based on the width of the peaks. Generally, broader (wider) peaks means smaller nanoparticles. In Figure 1 below, it is apparent that sample B) has smaller particle size than sample A) based on the wider peaks. On Figure 1B), the peaks are annotated, along with the peak widths at half maximum, β.

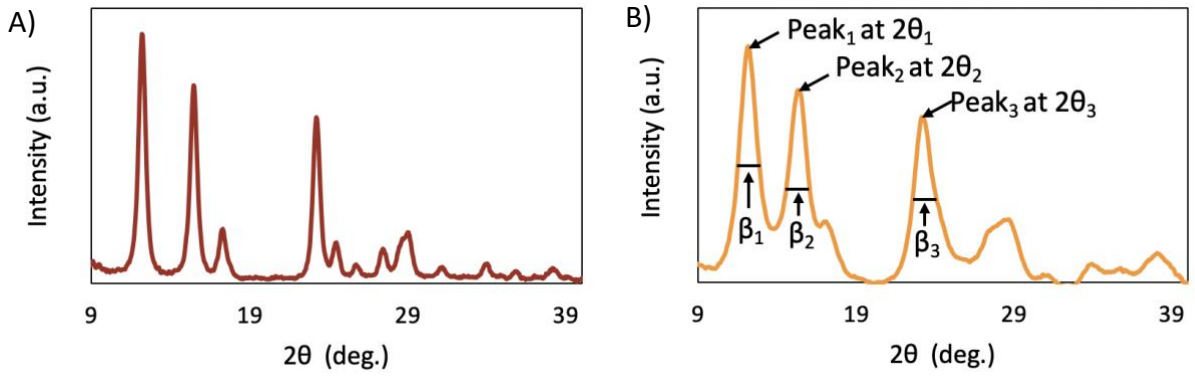

Figure 1. A) and B) both show example XRD intensity plotted against the Bragg angle, 2ϑ in degrees. Figure 1B) shows the annotated peaks and their widths at half maximum (β_i – note that this is shown in degrees though, not radians).

This calculation can be done by hand, but what if you are tasked with screening a bunch of nanoparticle materials? Well, then it would be ideal to write a script that takes in the XRD data and does this calculation automatically in order to save yourself lots of time! This is the challenge of Problem 1: implement a Python3 program that reads in the XRD data provided (this is a text file of 2θ angles (in degrees) and intensity pair values), that plots the graph similar to what is shown in Figure 1A) above using matplotlib, and that automatically estimates nanoparticle size based on the average of the τ values determined for each peak using the Scherrer Equation. Note that the provided file lists all 2θ angle (degrees) and intensity pairs on one line each, with the peak intensities annotated with an asterisk (*).

For instance, this could be a small subset of the file:

2θ (degrees)	Intensity (a.u.) (Peaks annotated with *)
12.14	95.97530555
12.16	98.27408102
12.18	99.76437208
12.2	100.0000000*
12.22	99.44131752
12.24	97.88960572
12.26	95.19694597

Note that the 2θ degree values and intensity values are separated by a tab character, and the peak intensities are annotated with asterisks at the end (*)

Hints: Note that when you are finding the peak width at half maximum, there may not be any data points exactly at the half maximum, so you may have to approximate the peak width at half maximum (a while loop might help you do this). Think about how you might approach this problem and be sure to explain it in your pseudocode!

Requirements:

Pseudocode (5 marks)

Write out your planned solution in pseudocode (doing so first will likely make implementing your solution in Python3 much easier!)

Code (3 marks)

Define a set of functions in Python3 that, together, may be used to accomplish the task defined in problem 1. You may define as many functions as appropriate, however you must have 1 main function called **XRD_Analysis(file)** that takes in the data text file, plots the Matplotlib figure and returns the estimated average particle size in nm as output. You will be marked on:

- **Legibility**: The TA's should be able to understand your code without spending hours reading it. For example, do not put your code in one very long line as this is hard for someone else reading your code to understand. **(1 mark)**
- **Comments**: Required for major parts of your code; your pseudocode may help here! **(1 mark)**
- **Variable Names**: Should be appropriate/meaningful **(1 mark)**
- Actual outputs from testing (see Testing below)

Testing (8 marks)

- Your solution should consist of a set of function definitions that are called by the overall function XRD_Analysis(). While you are implementing your solution, be sure to apply the principles of unit testing and test each individual function to make sure you're on the right track. ***For one of your function definitions***, document this unit testing by recording a normal, boundary and abnormal test case **(3 marks)**.
- Upon completion of your code, test your overall solution (XRD_Analysis() function) for the two data sets provided ("XRD_example1.txt" and "XRD_example2.txt"). Report your final results (the plot and the estimated particle size for each data set). **(4 marks)**
- Do your results make sense? Why or why not? **(1 mark)**

Reflection (4 marks)

If you were doing this for a real research project, what might you want to do to confirm whether or not your solution works as desired? What might you want to do to improve your solution? Maximum 2 paragraphs.

Problem 2: Creating a Date Class (22 marks)

In this problem you will be creating a class called **Date** which will do date related calculations. Design, implement, and test a program that satisfies the requirements below:

Requirements:

Code (11 marks)

Define the Python class called **Date** as outlined below:

1. The **__init__** method for **Date** takes the first formal parameter **self**, and takes three integers ***d, m,*** and ***y*** as input **in that order,** and stores each of them as instance variables (***d*** is the day, ***m*** is the month, and ***y*** is the year). You may choose to have additional instance variables if it is found to be useful to you.

2. The class contains accessors for ***d, m,*** and ***y.*** The names of the accessors are ***getDay(self), getMonth(self),*** and ***getYear(self)****, respectively.*

3. The class contains a method named ***Next(self)*** that takes only the formal parameter **self**. The method returns a **Date** object that is one day later than the current object. (for example, if the date was 31/12/2019 the next date should be 1/1/2020)

4. The class contains a method named ***Prev(self)*** that takes only the formal parameter **self**. The method returns a **Date** object that is one day earlier than the current object.

5. The class contains a method named ***isBefore(self, d)*** that takes in the formal parameter **self** and a **Date** object ***d***. The method returns the Boolean **True** if the current date is before ***d*** and the Boolean **False** otherwise.

6. The class contains a method named ***isAfter(self, d)*** that takes in the formal parameter **self** and a **Date** object ***d***. The method returns the Boolean **True** if the current date is after ***d*** and the Boolean **False** otherwise.

7. The class contains a method named ***isEqual(self, d)*** that takes in the formal parameter **self** and a **Date** object ***d***. The method returns the Boolean **True** if the current date is the same as ***d*** and the Boolean **False** otherwise.

8. The class contains a method named ***add_days(self, n)*** that takes in the formal parameter **self** and an integer ***n***. The method returns a **Date** object that is ***n*** days later than the date of the current object. (for example, if the date is 23/12/1998 and I add 15 days, the new date should be 7/1/1999)

9. The class contains a method named ***days_between(self, d)*** that takes in the formal parameter **self** and a **Date** object ***d***. The method returns an integer that is the number of days between the current date and ***d.*** (for example, the number of days between 19/03/2020 and 12/03/2020 is 7 days and the number of days between 12/03/2020 and 19/03/2020 is also 7 days)

Note: Make sure the spelling of the methods is the **exact same** as outlined in the requirements. **Hint**: Think about all of the possible scenarios for which these methods must work (we may have done something in Tutorial *(*cough*7*cough*)* that might help you out!). Remember that months have 30 or 31 days and that February can have either 28 or 29 days.

Code Legibility (4 Marks)

- **Legibility**: Same as part 1. **(1 mark)**
- **Comments**: Required for major parts of your code. **(1 mark)**
- **Variable Names**: Should be appropriate/meaningful **(1 mark)**
- **Exception Handling**: Have at least one try-except statement in at least one of your methods (Hint: think of how you would implement a try and except in a function) **(1 mark)**

Questions (4 Marks)

1. Does the formal parameter have to be called self? Explain.
2. What are the three main differences between a method and a function?
3. If you were to add one more method to this class, what would it be? What parameters would you pass in? And how would it make the implementation of the **Date** class easier?
4. Class instance variables can be referenced using two different ways: either using the accessor methods or direct referencing by using the instance variable reference (for example to get the day of the date you can use d.day or d.getDay()). Which one abides more towards the principle of information hiding? Explain. (This question might require some research from your end if you do not know what information hiding means)

Testing Plan (3 Marks)

- You should test all of your methods manually to ensure they function as expected. Record 3 test cases (one normal, one boundary, and one abnormal), ***each for a different method*** (one of ***Next***, ***Prev***, ***before, after, equal, add_days, days_between***). Do not record your tests for the constructor or accessor methods.
- Be sure to specify the objects on which you are testing the method, and any methods that you might use to compare your expected output to the actual output. Recall the format of a test case:

```
Test i for method j
Input/Method Call: method j
Expected Output: expected output for method j
Actual Output: actual output for method j
Result: Pass/Fail
```

Remember, submit a PDF of your non-code answers (the filled in student template renamed **YourMACID_TakeHomeResponses.pdf**) and the completed template python notebook, renamed **YourMACID_TakeHomeCode.ipynb**.

That's all Folks!