2 — PHY 494: Homework assignment (57 points total)

Due Thursday, Feb 2, 2017, 1:30pm.

Submission is now to your **private GitHub repository**. Follow the link provided to you by the instructor in order for the repository to be set up: It will have the name *ASU-CompMethodsPhysics-PHY494/assignments-2017*-YourGitHubUsername and will only be visible to you and the instructor/TA. Follow the instructions below to submit this (and all future) homework.

Read the following instructions carefully. Ask if anything is unclear.

1. `git clone` your assignment repository (change *YourGitHubUsername* to your GitHub username)

   ```
   repo="assignments-2017-YourGitHubUsername.git"
   git clone https://github.com/ASU-CompMethodsPhysics-PHY494/${repo}
   ```

2. run the script `./scripts/update.sh` (replace *YourGitHubUsername* with your GitHub username):

   ```
   cd ${repo}
   bash ./scripts/update.sh
   ```

   It should create three subdirectories[1] `assignment_02/Submission`, `assignment_02/Grade`, and `assignment_02/Work`.

3. You can try out code in the `assignment_02/Work` directory but you don't have to use it if you don't want to. Your grade with comments will appear in `assignment_02/Grade`.

4. Create your solution in `assignment_02/Submission`. Use Git to `git add` files and `git commit` changes.

   You can create a PDF, a text file or Jupyter notebook inside the `assignment_02/Submission` directory as well as Python code (if required). **Name your files `hw02.pdf` or `hw02.txt` or `hw02.ipynb`**, depending on how you format your work. Files with code (if requested) should be named exactly as required in the assignment.

5. When you are ready to submit your solution, do a final `git status` to check that you haven't forgotten anything, commit any uncommited changes, and `git push` to your GitHub repository. Check on *your* GitHub repository web page[2] that your files were properly submitted.

   You can push more updates up until the deadline. Changes after the deadline will not be taken into account for grading.

[1] If the script fails, file an issue in the Issue Tracker for PHY494-assignments-skeleton and just create the directories manually.

[2] `https://github.com/ASU-CompMethodsPhysics-PHY494/assignments-2017-`*`YourGitHubUsername`*

Homeworks must be legible and intelligible or may otherwise be returned ungraded with 0 points.

This assignment contains **bonus problems**. A bonus problem is optional. If you do it you get additional points that count towards this homework's total, although you can't get more than the maximum number of points. If you don't do it you can still get full points. Bonus problems and bonus points are indicated with an asterisk "*".

2.1 Version control with Git (9 points)

Keep your answers short, one or two sentences should be sufficient for questions (b)–(d).

(a) *Briefly* explain what a version control system such as Git does and how it can help you. (For your answer, it is sufficient to focus on three different aspects out of many — choose the ones that *you* find most important.) [**3 points**]

(b) Explain the difference between `git init` and `git clone`. [**2 points**]

(c) Explain the difference between `git add` and `git commit`. [**2 points**]

(d) Explain the difference between `git commit` and `git push` [**2 points**]

2.2 Your GitHub account (10 points)

As part of the last lesson you should have set up your own GitHub account on https://github.com (if you have not done it yet, do it now!). What is your **GitHub username**?

- Write down your GitHub username. [**10 points**]
- Take the survey PHY 494: Your GitHub account[3] if you have not done so already.

2.3 Simple coordinate manipulation in Python (11 points)

We can represent the cartesian coordinates $\mathbf{r}_i = (x_i, y_i, z_i)$ for four particles as a list of lists `positions`:

```
positions = \
    [[0.0, 0.0, 0.0], [1.34234, 1.34234, 0.0], \
     [1.34234, 0.0,  1.34234], [0.0, 1.34234, 1.34234]]
```

For the following, do not import any additional modules: *only use pure Python.*

(a) How do you access the coordinates of the second particle and what is the output? [**1 points**]

[3] In case the link to the survey is not clickable: got to https://goo.gl/forms/AZMtF6c60FBdCCkt1. You must be logged in with your ASU account. Log in to https://my.asu.edu first and then go to the survey.

(b) How do you access the y-coordinate of the second particle and what is the output? **[1 points]**

(c) Write Python code to translate all particles by a vector $\mathbf{t} = (1.34234, -1.34234, -1.34234)$,

```
t = [1.34234, -1.34234, -1.34234]
```

Show your code and the translated coordinates. **[3 points]**

(d) Make your solution of (c) a function `translate(coordinates, t)`, which translates all coordinates in the argument `coordinates` (a list of N lists of length 3) by the translation vector in `t`. The function should return the translated coordinates.

Show the code and the function applied to (1) the input `positions` and `t` from above and (2) for `positions2 = [[1.5, -1.5, 3], [-1.5, -1.5, -3]]` and `t = [-1.5, 1.5, 3]`. **[6 points]**

2.4 NumPy arrays (11 points)

Work through the NumPy tutorial.[4] Do the examples while you read it.

(a) How do NumPy array operations such as $+$, $-$, $*$, $/$ ... differ from linear algebra operations (i.e. scalar product, vector/matrix multiplication, ...)? **[2 points]**

(b) Given the three arrays

```python
import numpy as np

sx = np.array([[0, 1], [1, 0]])
sy = np.array([[0, -1j],[1j, 0]])
sz = np.array([[1, 0], [0, -1]])
```

(i) What is the result of `sx * sy * sz`? Explain what happened. **[2 points]**

(ii) Use `np.dot()` to multiply the three arrays (like $\sigma_x \cdot \sigma_y \cdot \sigma_z$). Show the code that you are running and the final result. Explain what happened. **[2 points]**

(iii) Compute the "commutator" $[\sigma_x, \sigma_y] := \sigma_x \cdot \sigma_y - \sigma_y \sigma_x$ and show that it equals $2i\sigma_z$.[5] Show your code and the results. **[3 points]**

(iv) Given a "state vector"

$$\mathbf{v} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix}$$

[4] Some stuff such as the `ix_()` function is fairly esoteric for beginners but almost everything else is what you should be familiar with for your daily work with arrays.

[5] These are the Pauli matrices that describe the three components of the spin operator for a spin 1/2 particle, $\hat{\mathbf{S}} = \frac{\hbar}{2}\boldsymbol{\sigma}$. The fact that any two components of the spin operator do *not* commute is a fundamental aspect of quantum mechanics.

calculate the "expectation value" $\mathbf{v}^\dagger \cdot \sigma_y \cdot \mathbf{v}$ (i.e., the multiplication of the hermitian conjugate of the vector, $\mathbf{v}^\dagger$ with the matrix σ_y and the vector $\mathbf{v}$ itself) using NumPy. [6] Show your calculation and your result.[7]) [**2 points**]

2.5 Coordinate manipulation with NumPy (16 points)

We can represent the cartesian coordinates $\mathbf{r}_i = (x_i, y_i, z_i)$ for four particles as a numpy array `positions`:

```
import numpy as np
positions = np.array(\
    [[0.0, 0.0, 0.0], [1.34234, 1.34234, 0.0], \
     [1.34234, 0.0,  1.34234], [0.0, 1.34234, 1.34234]])
t = np.array([1.34234, -1.34234, -1.34234])
```

and `t` will be a translation vector. *For the following use NumPy.*

(a) What is the *shape* of the array `positions` and what is its *dimension*? [**1 points**]

(b) What is the *shape* of the array `t` and what is its *dimension*? [**1 points**]

(c) How do you access the coordinates of the second particle in `positions` and what is the output? [**1 points**]

(d) For the second particle:

  (i) How do you access its y-coordinate and what is the output? [**2 points**]

  (ii) What type of object is this output, what is its *shape* and its *dimension*? [**2 points**]

(e) Write Python code to translate all particles by a vector $\mathbf{t} = (1.34234, -1.34234, -1.34234)$,

```
t = np.array([1.34234, -1.34234, -1.34234])
```

Show your code and the translated coordinates. [**3 points**]

(f) Make your solution of (e) a function `translate(coordinates, t)`, which translates all coordinates in the argument `coordinates` (an `np.array` of shape `(N, 3)`) by the translation vector in `t`. The function should return the translated coordinates as a numpy array.

[6] The *hermitian conjugate* $\mathbf{v}^\dagger = (v_1^*, v_2^*)$ is `v.conjugate().T` where `v.T` is shorthand for `v.transpose()`. It turns out that you don't need the transposition when you use `np.dot()` but I include it here for conceptual clarity. (Including `transpose()` comes at a minor performance penalty — check with `%timeit` if you are curious.)

[7] Note for anyone having done PHY 315 (Quantum Mechanics II) that here you are calculating the quantum mechanical expectation value of the y-component of a spin $\frac{1}{2}$ particle in an eigenstate of the operator of the y-component of the spin ($\sigma_y \mathbf{v} = -\mathbf{v}$) and because $\mathbf{v}$ is normalized, you should get the eigenvalue as the expectation value.

Show the code and the function applied to (1) the input `positions` and `t` from above and (2) for `positions2 = np.array([[1.5, -1.5, 3], [-1.5, -1.5, -3]])` and `t = np.array([-1.5, 1.5, 3])`. [**6 points**]

2.6 BONUS: File processing in Python (15* bonus points)

The standard way to open a file in Python and to process it line by line is the code pattern

```
with open(filename) as inputfile:
   for line in inputfile:
       line = line.strip()   # strip trailing/leading whitespace
       if not line:
          continue           # skip empty lines
       # now do something with a line
       # E.g., split into fields on whitespace
       fields = line.split()
       # access data as fields[0], fields[1], ...
       x = float(fields[0])  # convert text to a float
       y = float(fields[1])
       # ...
print("Processed file ", filename)
```

In brief:

1. A file is opened for reading with `open(filename)`, which returns a *file object* (here assigned to the variable `inputfile`). The `with` statement is a very convenient way to make sure that the file is always being closed at the end: when the `with`-block exits (here at the `print` statement), `inputfile.close()` is called implicitly[8].

2. We *iterate* over all lines in the file (similar to what we did for lists) in a `for`-loop.

3. Remove leading and trailing white space with the `strip()` method of a string (`line` is a string). If you want to keep all white space, do not use `strip()`.

---

[8] If you were not to use `with`, your code would look like

```
inputfile = open(filename)
for line in inputfile:
   # ...
inputfile.close()
print("Processed file ", filename)
```

but with the disadvantage that when something goes wrong during the `for`-loop, your file will never be closed, which exhausts system resources. When open a file for *writing* (`open(filename, 'w')`) you will *corrupt the file* when you are not closing it properly. The `with` statement guarantees that the file will *always be closed, no matter what else happens*. Use the `with` statement!

4. Skip empty lines: note that an empty string evaluates to `False` and thus can be used directly in the `if` statement. The `continue` statement then starts the next iteration in the loop.

5. Start processing the line. Often you know the structure of the file (e.g. a data file with 3 columns, separated by white space) so you typically split into fields (the string's `split()` method produces a list). Select the fields as needed.

6. As an example, fields 0 and 1 are assumed to represent floating point numbers. `fields[0]` contains a string but using `float(fields[0])` it can be converted ("cast") to a Python float. Similarly, integer numbers can be cast with `int()`.

Use the above information to write a Python program that reads the file

`PHY494-resources/01_shell/data/starships.csv`

splits lines on commas[9], and prints out the names and cost (in credits, "CR") of all starships that cost more than 100 million CR.[10]

Show your code and your output. [**bonus +15***]

---

[9] "csv" stands for "comma separated values" and is a common file format for tabular data.

[10] Hint: Turn all `"unknown"` entries into 0 and then cast numbers to floats.