4 — PHY 494: Homework assignment (36 points total)

Due Thursday, Feb 7, 2019, 5pm.

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-2019*-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-2019-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_04/Submission`, `assignment_04/Grade`, and `assignment_04/Work`.

3. You can try out code in the `assignment_04/Work` directory but you don't have to use it if you don't want to. Your grade with comments will appear in `assignment_04/Grade`.

4. Create your solution in `assignment_04/Submission`. Use Git to `git add` files and `git commit` changes.

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

You can create a PDF, a text file or Jupyter notebook inside the `assignment_04/Submission` directory as well as Python code (if required). **Name your files `hw04.pdf` or `hw04.txt` or `hw04.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.

Homeworks must be legible and intelligible and on-time or may be returned ungraded with 0 points.

Bonus problems 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 "*".

Included code and tests The homework comes with starter code in the `Submission` directory. Edit and submit code as directed in the problems. The directory also includes files `test_hw4.py` and `test_bugs.py`. You can use these tests to check if your solutions are correct:

```
pytest
```

(If you solved all coding problems, you should see "112 passed". Otherwise you will be informed which problems failed.)

[2] `https://github.com/ASU-CompMethodsPhysics-PHY494/assignments-2019-`*`YourGitHubUsername`*

4.1 Counting Vowels (11 points)

Given a string `s`, count how often each of the 6 vowel letters in the English alphabet (A, E, I, O, U, Y – we include Y here) occurs. You can ignore case by converting the string to lowercase with `s.lower()`.

(a) Write a function `count_vowels(s)` and put it in a file `problem1.py`. It should take a string `s` as input and return a list (let's call it `counts`) with 6 elements, where `count[0]` is the count for letter A, `count[1]` for E etc. [3] **[10 points]**

(b) Apply your function to the string

```
s = """'But I don't want to go among mad people,' Alice remarked.
'Oh, you can't help that,' said the Cat, 'we're all mad here. I'm
mad. You're mad.' 'How do you know I'm mad?' said Alice.  'You
must be,' said the Cat, 'or you wouldn't have come here.'"""
```

and report the counts in the variable `counts` in the same file. **[1 points]**

4.2 Factorial Fun (15 points)

The factorial function is defined for integers $n > 0$ by

$$n! = n \cdot (n-1) \cdot (n-2) \ldots 2 \cdot 1 = \prod_{k=1}^{n} k \tag{1}$$

and $0! = 1$ is *defined* for consistency.

The double factorial is defined by

$$n!! = \begin{cases} n \cdot (n-2) \cdot (n-4) \cdot \cdots \cdot 5 \cdot 3 \cdot 1, & n > 0 \text{ odd} \\ n \cdot (n-2) \cdot (n-4) \cdot \cdots \cdot 6 \cdot 4 \cdot 2, & n > 0 \text{ even} \\ 1, & n = 0, -1 \end{cases}. \tag{2}$$

[3] Hint: you can iterate through a string like a list using `for letter in s:` and then analyze the letter.

(a) Find a function in the `math` standard library to calculate $n!$ (Eq. 1).

(i) Write a function `factorial_math(n)` that computes $n!$ for $n \geqslant 0$. This function *should* use the function from `math`. Put the function `factorial_math(n)` in a file `problem2a.py`. **[2 points]**

(ii) Show results for the integers $n = 0, 1, 2, 3, ..., 20$. **[1 points]**

(b) Write a function to calculate the factorial $n!$ (Eq. 1):

(i) Create a function `factorial(n)` that computes $n!$ for $n \geqslant 0$. This function should *not* use any modules (i.e., neither `math` nor `numpy`). Put the function in a file `problem2b.py`. **[4 points]**

(ii) Show results for the integers $n = 0, 1, 2, 3, ..., 20$. **[1 points]**

(c) Write a function to calculate the double factorial $n!!$ (Eq. 2):

(i) Create a function `double_factorial(n)` that computes the double factorial Eq. 2 for an integer `n`. The function should only return the value of $n!!$. This function should *not* use any modules (i.e., neither `math` nor `numpy`). Put the function in a file `problem2c.py`. **[6 points]**

(ii) Show results for the integers $n = 0, 1, 2, 3, ..., 20$. **[1 points]**

4.3 Squash the bug (10 points)

Three files `bug_a.py`, `bug_b.py`, and `bug_c.py` are include. Each contains at least one Python bug. Fix them and commit your fixed files.[4]

[4] You will also see a file `test_bugs.py`. It contains *tests* that check your code. Your instructors will *run these tests on your code*. You can run them yourself with the `pytest` command,

```
pytest -v test_bugs.py
```

If everything is correct, you should see something like `===== 36 passed in 0.36 seconds =====`. If tests fail then you can correct your code until you get the tests to pass.

(a) Fix `bug_a.py` and commit the fixed file. The code should run, assign the correct value to the variable `value`, and print the correct value. **[3 points]**

(b) Fix `bug_b.py` and commit the fixed file. The code should add two vectors $\mathbf{a} = (12.3, 3.90, 4.5)$ and $\mathbf{b} = (1.3, 0.91, -3.3)$ and print the value of the new vector $\mathbf{c} = 5\mathbf{a} - 3\mathbf{b}$ and assign the value to the variable `c`. **[3 points]**

(c) Fix `bug_c.py` and commit the fixed file. You should correctly implement the 2D sinc function

$$\operatorname{sinc}(x, y) := \frac{\sin x}{x} \frac{\sin y}{y}$$

(where x and y are given in radians). The function should be correct for arbitrary input.[5] **[4 points]**

4.4 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, 'r') 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])
```

[5] Hint: Especially consider the edge and corner cases.

```
12          # ...
13  print("Processed file ", filename)
```

In brief:

1. A file is opened for reading with `open(filename, 'r')`, which returns a *file object* (here assigned to the variable `inputfile`).[6] 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()` function), `inputfile.close()` is called implicitly[7].

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()`.

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

[6] Opening for writing uses the `'w'` argument and one can write a line to the file with `fileobject.write(...)`.

[7] 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!

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 `evaluate_ships.py` that reads the file starships.csv (which is also provided as part of the homework). Fields in this file are separated by *commas*.[8] The meaning of the fields (or columns) is

```
name,model,vehicle_class,max_atmospheric_speed,cost_in_credits,length
```

i.e, the first column contains the *name*, the second column the *vehicle class* etc. Split the lines on commas and print out the names and cost (in credits, "CR") of all starships that cost more than 100 million CR.[9]

Submit your code `evaluate_ships.py` and your output in a file `starship_costs.dat`. **[bonus +15*]**

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

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