2 — PHY 494: Homework assignment (60 points total)

Due Thursday, Jan 28, 2016, 1:30pm.
Submit a PDF or text file through Blackboard (name it `lastname_firstname_hw2.pdf`). Homeworks must be legible 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)

(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 it down here **and** take the survey PHY 494: Your GitHub account[1] **[10 points]**

2.3 Python control statements (10 points)

For each of the following control statements, briefly state (a) the syntax, (b) the purpose, (c) a short example of Python code using the statement.

Not all of these statements were discussed in class—you will have to learn about them on your own (see the Resources for Python given in the Introduction to Python).

(a) `for` loop **[2 points]**

(b) `while` loop **[2 points]**

(c) `if` statement **[2 points]**

(d) `break` statement **[2 points]**

(e) `continue` statement **[2 points]**

[1] In case the link to the sruvey is not clickable: got to http://goo.gl/forms/ue1A178tbb. You must be logged in with your ASU account. Log in to https://my.asu.edu first and then go to the survey.

2.4 Python Lists and Strings (20 points)

Lists and strings share some similarities but also have important differences. Let's look at them. (Type code in the Python interpreter (e.g., in a Jupyter notebook or ipython or python)).

```
bag = ["guide", "towel", "tea", 42]
ga = "Four␣score␣and␣seven␣years␣ago"
```

(Note that spaces are shown explicitly in the second string with the symbol "␣"—just type a space.)

(a) How do you have to slice `bag` in order to get `['towel', 'tea']`? **[1 points]**

(b) What does `bag[::-1]` do?

How do you slice bag in order to get `['tea', 'towel']`? **[2 points]**

(c) Strings can also be sliced. How do you have to slice `ga` to get

- "Four"
- "seven"

[2 points]

(d) You can access elements of a list in a variety of ways:

(i) Explain what

```
bag[0] = 'book'
```

does? (Hint: print `bag`!) **[1 points]**

(ii) Create two new variables:

```
mybag = bag
yourbag = bag[:]
```

and use them:

```
mybag[3] = "mice"
yourbag.append("money")
```

What is the content of `bag`, `mybag`, `yourbag`? **[2 points]**

(iii) From your observations, explain how the assignment `x = a` differs from `y = a[:]`? **[3 points]**

(e) Try

```
ga[:4] = "Three"
```

(i) Describe what happens?[2] **[1 points]**

(ii) How would you construct the string `"Three score and seven years ago"` from `ga` and the string `"Three"`? **[1 points]**

(f) What do the commands

```
ga.split()
a, b, c = ga.split()[:3]
list([1,2,3])
list(ga)
```

do? You can show the output but you need to explain in your own words what is happening. **[4 points]**

(g) Nested lists: Given the list

```
bags = [['salt', 'pepper'], ['pen', 'eraser', 'ruler']]
```

how do you have to index `bags` to get

(i) `['salt', 'pepper']` **[1 points]**

(ii) `'pepper'` **[1 points]**

(iii) `'ruler'` **[1 points]**

2.5 Simple coordinate manipulation in Python (11 points)

We 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]]
```

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

(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]**

[2] Note that strings are "immutable" objects in Python whereas lists are "mutable".

(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]**

2.6 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[3].
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()`.

[3] 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[4], and prints out the names and cost (in credits, "CR") of all starships that cost more than 100 million CR.[5]

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

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

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