Lesson 3: Strings and lists in action

After completing Lesson 1 and Lesson 2 of this course in *"Engineering Computations,"* here we have a full example using all that you've learned.

You may be wondering why we're dedicating the first lessons of the course to playing around with strings and lists. *"Engineering computations involve numbers, formulas and equations!"*, you may be thinking. The reason is that this course assumes no programming experience at all, so we want to get everyone used to Python first, without adding the extra complexity of number-crunching. The idea is to get acquainted with programming constructs first, applying them to situations that involve no mathematics ...for now!

1 Play with the MAE bulletin

We are going to play with text from a file containing a copy of the MAE Bulletin for 2017-2018. We'll create different lists to allow us to locate the title, credits and description of a course based on the course code.

The data file for this example should be located in a folder named `data`, two levels above the location of this lesson—if you copied the course materials as they were stored. If you have the data tile elsewhere, you should edit the full path below.

We'll start by reading the data file into the Jupyter notebook, then we'll clean the data a bit, and finally work out ways to play with it.

2 Read data from a file

We know that we have a data file, and we'd like to read its contents into the Jupyter notebook. Usually, it's a good idea to first peek into the file, to see what its contents look like. This also gives us a chance to teach you a pretty neat trick.

Recall that code cells in a Jupyter notebook can handle any valid **IPython** statement. Well, IPython is able to do a bit more than just Python: it can also run any system command. If you know some Unix, this can be very useful—for example, you could list all the files in the working directory (your location in the computer's file system).

To execute a system (a.k.a., shell) command, you prepend `!`—a "bang." The command we need is `head`: it prints out the first few lines of a file.

Our data folder is found two directories up from the lessons (this Jupyter notebook): in Unix, going *up* a directory is indicated by two dots; so we need to have `../../data/` before the file name,`mae_bulletin.txt`, as part of the path.

Let's call `head` with a bang:

```
In [1]: !head ../../data/mae_bulletin.txt
```

```
MAE 1004. Engineering Drawing and Computer Graphics. 0-3 Credits.

Introduction to technical drawing, including use of instruments, lettering,
geometric construction, sketching, orthographic projection, section view,
dimensioning, tolerancing, and pictorial drawing. Introduction to computer
graphics, including topics covered in manual drawing and computer-aided
drafting.   (Fall and spring).

MAE 2117. Engineering Computations. 3 Credits.

Numerical methods for engineering applications. Round-off errors and discretization
errors. Methods for solving systems of linear equations, root finding, curve fitting,
numerical Fourier transform, and data approximation. Numerical differentiation and
integration and numerical solution of differential equations. Computer applications.
Prerequisite: MATH 1232. (Fall, Every Year).

MAE 2124. Linear Systems Analysis for Robotics. 3 Credits.
```

That looks good! The next step is to *open the file* and save its contents into a Python variable that we can use.

The `open()` function with the file name as a *string* argument (note the quotes) returns a Python file object. We have several options next, and you should definitely read the documentation about reading and writing files.

If you use the `read()` file method, you get a (big) string with all the file's contents. If you use instead the `readlines()` method, you get a list of strings, where each string contains one line in the file. Yet another option is to use the `list()` function to create a list of lines from the file contents.

```
In [2]: mae_bulletin_file = open('../../data/mae_bulletin.txt')

In [3]: mae_bulletin_text = mae_bulletin_file.readlines()

In [4]: type(mae_bulletin_text)

Out[4]: list

In [5]: len(mae_bulletin_text)

Out[5]: 431
```

3 Cleaning and organizing text data

When manipulating text data, one of the typical actions is to get rid of extra white spaces and lines. Here, we'll remove the spaces at the beginning and end of each line.

Note that there are also some blank lines: we'll skip them. The goal is to get two new lists: one with the course ID line, and another with the course descriptions.

Study the following block of code:

```
In [6]: courses = []
        descriptions = []

        for line in mae_bulletin_text:
            line = line.strip()          #Remove white spaces
            if line == '':               #Skip the empty lines
                continue
            elif line.startswith('MAE'):
                courses.append(line)     #Save lines that start with MAE in list
            else:
                descriptions.append(line)  #Save descriptions in other list
```

Be sure to also visit the documentation for string methods, to get a glimpse of all the things you can do with Python! Here, we used the `strip()` method to get rid of leadign and trailing spaces, and we also used `startswith()` to identify the course ID lines.

Let's check what we ended up with by printing a few items in each list:

```
In [7]: #print first 5 elements of courses
        print(courses[0:5])
```

```
['MAE 1004. Engineering Drawing and Computer Graphics. 0-3 Credits.', 'MAE 2117.
Engineering Computations. 3 Credits.', 'MAE 2124. Linear Systems Analysis for
Robotics. 3 Credits.', 'MAE 2131. Thermodynamics. 3 Credits.', 'MAE 2170. History
and Impact of the US Patent System. 3 Credits.']
```

```
In [8]: #print first 3 elements of descriptions
        print(descriptions[0:3])
```

```
['Introduction to technical drawing, including use of instruments, lettering,
geometric construction, sketching, orthographic projection, section view,
dimensioning, tolerancing, and pictorial drawing. Introduction to computer graphics,
including topics covered in manual drawing and computer-aided drafting.   (Fall and
spring).', 'Numerical methods for engineering applications. Round-off errors and
discretization errors. Methods for solving systems of linear equations, root finding,
curve fitting, numerical Fourier transform, and data approximation. Numerical
differentiation and integration and numerical solution of differential equations.
Computer applications. Prerequisite: MATH 1232. (Fall, Every Year).', 'Properties of
linear systems. Mathematical modeling of dynamic systems. State space, state variables,
and their selection. Linearization of non-linear behavior. Matrix functions. Solution
of state equations in the time domain and using transformations. System stability and
frequency response.']
```

We should also check that both lists are the of the same length!

```
In [9]: print(len(courses))
        print(len(descriptions))
```

```
108
108
```

4 Separate `courses` list into course id, title, and credits

We may want to have the information of the course id, title, and credits in separate lists. Here's how we could do that:

```
In [10]: course_id = []
         course_title = []
         course_credits = []

         for course in courses:
             course_info = course.split('. ')
             course_id.append(course_info[0])
             course_title.append(course_info[1])
             course_credits.append(course_info[2])
```

Note that we split using the string `'. '` (period + space) to avoid having an extra white space at the beginning of each string in the lists for course title and credits.

Let's print out the first elements of the new lists.

```
In [11]: print(course_id[0:5])
```

```
['MAE 1004', 'MAE 2117', 'MAE 2124', 'MAE 2131', 'MAE 2170']
```

```
In [12]: print(course_title[0:5])
```

```
['Engineering Drawing and Computer Graphics', 'Engineering Computations',
'Linear Systems Analysis for Robotics', 'Thermodynamics', 'History and
Impact of the US Patent System']
```

```
In [13]: print(course_credits[0:5])
```

```
['0-3 Credits.', '3 Credits.', '3 Credits.', '3 Credits.', '3 Credits.']
```

5 Tracking course information

The lists we have created are aligned: that is each item on the same index location corresponds to the same course. So by finding the location of a course id, we can access all the other information.

We use the `index()` method to find the index of the course id and track the rest of the info. Try it out!

```
In [14]: course_id.index('MAE 3190')
```

```
Out[14]: 17
```

```
In [15]: print(course_id[17])
         print(course_title[17])
         print(course_credits[17])
         print(descriptions[17])
```

```
MAE 3190
Analysis and Synthesis of Mechanisms
3 Credits.
Kinematics and dynamics of mechanisms. Displacements, velocities, and accelerations
in linkage, cam, and gear systems by analytical, graphical, and computer methods.
Synthesis of linkages to meet prescribed performance requirements. Prerequisite:
APSC 2058.  (Fall).
```

6 How many courses have prerequisites?

Here's an idea: we could look for all the courses that have prerequisites using a `for` statement. In this case, we'll iterate over the *index* of the elements in the list `descriptions`.

It gives us a chance to introduce to you a most helpful Python object: `range`: it creates a sequence of numbers in arithmetic progression to iterate over. With a single argument, `range(N)` will create a sequence of length `N` starting at zero: `0, 1, 2, ..., N-1`.

```
In [16]: for i in range(4):
             print(i)
```

```
0
1
2
3
```

The funny thing with `range` is that it's created on-the-fly, when iterating over it. So it's not really a list, although for most practical purposes, it behaves like one.

A typical way to use it is with an argument that's output by the `len()` function. Study this block of code:

```
In [17]: course_with_pre = []

         for i in range(len(descriptions)):
             if 'Prerequisite' in descriptions[i]:
                 course_with_pre.append(course_id[i])
```

Now we have a list named `course_with_pre` that contains the id of all the courses that have prerequisites.

```
In [18]: print(course_with_pre)
```

```
['MAE 2117', 'MAE 2131', 'MAE 3120', 'MAE 3126', 'MAE 3128', 'MAE 3134', 'MAE 3145',
'MAE 3155', 'MAE 3162', 'MAE 3166W', 'MAE 3167W', 'MAE 3187', 'MAE 3190', 'MAE 3191',
'MAE 3192', 'MAE 3193', 'MAE 3195', 'MAE 3197', 'MAE 4129', 'MAE 4149', 'MAE 4157',
'MAE 4163', 'MAE 4168', 'MAE 4172', 'MAE 4182', 'MAE 4193', 'MAE 4194', 'MAE 4198',
'MAE 4199', 'MAE 6201', 'MAE 6207', 'MAE 6220', 'MAE 6221', 'MAE 6222', 'MAE 6223',
'MAE 6224', 'MAE 6225', 'MAE 6226', 'MAE 6227', 'MAE 6228', 'MAE 6229', 'MAE 6230',
'MAE 6231', 'MAE 6232', 'MAE 6233', 'MAE 6234', 'MAE 6237', 'MAE 6238', 'MAE 6239',
'MAE 6240', 'MAE 6241', 'MAE 6243', 'MAE 6244', 'MAE 6245', 'MAE 6246', 'MAE 6247',
'MAE 6249', 'MAE 6251', 'MAE 6252', 'MAE 6253', 'MAE 6254', 'MAE 6255', 'MAE 6257',
'MAE 6258', 'MAE 6260', 'MAE 6261', 'MAE 6262', 'MAE 6270', 'MAE 6271', 'MAE 6274',
'MAE 6276', 'MAE 6280', 'MAE 6281', 'MAE 6282', 'MAE 6283', 'MAE 6284', 'MAE 6286',
'MAE 6287', 'MAE 6288', 'MAE 6290', 'MAE 6291', 'MAE 6292', 'MAE 8350', 'MAE 8351',
'MAE 8352']
```

Exercise:

1. Save in a new list named `course_with_cor` all the courses that have a corequisite, and print out the list.
2. Using a `for` statement and `if-elif-else` statements, separate the courses that are offered in the Fall semester, those offered in the Spring semester, those offered in both semesters, and those that don't specify a semester. Create 4 lists: `fall_and_spring`, `fall`, `spring` and `not_spec`.

To check your answers uncomment the following lines by deleting the # symbol and running the cell. If there is no output you got it right!

```
In [19]:
```

7 What we've learned

- System commands in a code cell start with a bang (`!`).
- Opening a text file and saving its contents into a string or list variable.
- Cleaning up text data using string methods.
- Manipulating text into lists.