**CS-1004, Introduction to Programming for Non-majors, A-term 2016**

# Lab 5 — String Manipulation

Due: at 11:59 pm on the day of your lab session

## Objectives

- To work with strings
- To learn to use the *Python* documentation
- (Optionally) to run a *Python* program from a command line

**Note:** Don't worry if you can't finish the entire lab exercise. Submit as much as you've completed to *Canvas* before you leave the lab. Finish the rest of the lab on your own time.

Before you start, please sign the attendance sheet. Also, open an edit window in IDLE to create a file named **`Lab5.py`**. Develop all of the functions of this lab in this file, including any test code.

## Splitting a string

Consider a string representing the time of day — for example, **`'10:55:39'`** which means 39 seconds after 10:55 AM. In this exercise, you will first pick apart a string such as this and then convert hours to minutes and minutes to seconds to get the time in seconds after midnight.

To do this, you will split a string into one or more substrings using the **`split()`** string method.

1. To learn about the **`split()`** method, select the **`Help > Python Docs`** menu command of IDLE. The *Python Documentation* window should open. The panel at the left of the window contains four tabs — *Contents, Index, Search,* and *Favorites*. Under *Index*, type **`split`**. The index will scroll to the "split" entries in the index. Select "(str method)" to bring up the documentation page for the string method named **`split`**. Read and study this description.[1]
2. Write a function called **`timeToSeconds(time)`** that takes a string of the form

   **`hh:mm:ss`**

   and converts it into the number of seconds since midnight. You may assume that $0 \le$ **`hh`** $< 24$, that $0 \le$ **`mm`** $< 60$, and that $0 \le$ **`ss`** $< 60$. Your function must return an integer value.
3. Write second function called **`secondsToTime()`** that converts the number of seconds since midnight into a time string of the form accepted by part 2 above. Your function must return this string.

---

[1] A brief description of the **`split()`** method can also be found on page 136 of the textbook.

4. Test both functions against each other. Write a test program that generates a wide variety of times in the form of seconds after midnight, calls **`secondsToTime()`** for each one, and then calls **`timeToSeconds()`** for each result. The test program should print its original number and then the output of each function on one line, so that you can verify that it works.

   **Note:** There are *86400* seconds in a day. Your test program may use a random number generator (p. 271), or it may cycle thru the integers at intervals of a prime number. Make sure that your test program gets good coverage of the hours, minutes, and seconds.

## Parse a date

In this exercise, you will use the string method **`find()`** to help you to parse a date.

5. Learn about the **`find()`** string method in the *Python* documentation.[2]

6. Write a function called **`parseDate(dateString)`** that returns a list of three integers representing the month, day, and year of the date represented by the **`dateString`** argument.

   You may assume that the date string contains exactly the following:– a month name spelled correctly and capitalized in English) followed by a space, followed by the day (an integer), followed by another space, followed by the year (also an integer).

   Although it is possible to convert month names to numbers using a long sequence of **`if`** and **`elif`** statements, in this Lab, you must use the **`find()`** string method.

   To do this, set up a constant string of the form

   ```
   JanFebMarApr…
   ```

   containing the first three letters of the names of the months, concatenated together in order.

   When presented with an input date, slice off the first three letters of the input month, and then use the **`find()`** string method to find the position of those first three letters in your constant string. Finally, work from that position to determine the month number.

   Write a test function to prompt the user for input dates and confirm that they are correct.

7. If you still have time, modify your **`parseDate()`** function to ignore the case of the input string. Call the modified function **`parseDate2()`**. Find and use the appropriate string methods to deal with the case of the string.

If you wish to try the extra credit part of Homework #5, read on. Otherwise, you are done for this lab. Be sure to save your **`Lab5.py`** file and submit it to *Canvas* under the assignment *Lab5*.

## Extra Credit: Accessing Python thru a Command Prompt

A *Command Prompt* (also called a *Command Shell* or simply a *Shell*) is a tool for running computer programs from commands written as text. Commands in command shells are "imperative" in the sense

---

[2] The **`find()`** method for strings is mentioned briefly in Table 5.2 on page 140.

that each command is an order for the computer to "do" something to or with a set of arguments. A typical command has the form

```
commandName arg1 arg2 arg3 ...
```

where **commandName** is the name of a file containing an executable program, and **arg1**, **arg2**, **arg3**, etc., are *argument* strings separated by whitespace that control what the command operates on. These argument strings may be file names or other strings.

In this lab, we will build up the ability to run *Python* from a command shell so that it prints the argument strings.

8. Open a command prompt. (In Windows 7, this can be found in *Start Menu* > *Accessories* > *Command Prompt*). In Windows 8, you may have to search for it. (On the Macintosh, it is called *Terminal* and can be found in the */Applications/Utilities* folder.)

   Type the command **python**. (On the Mac, type **python3** because the command **python** would invoke *Python 2.7*, which is already built into all Macintosh and Linux systems.)

   You should now see a *Python* shell, but without the supporting infrastructure of IDLE. Type a few *Python* expressions or statements to confirm that this works.

   If you were successful in running **python** from the command line, skip forward the next step.

   However, if Windows complains that **python** is not recognized as an internal or external command, you probably have to update the **PATH** environment variable. See the Appendix at the end of this document for information on how to set it.

   When you are satisfied that you can run **python** from the command prompt (or Terminal window in the Macintosh), type the **exit()** function to exit *Python* and return to the command prompt.

9. Type or copy-and-paste the following function to the end of **Lab5.py**:–

```
import sys
def printArgv():
    for arg in sys.argv:
        print(arg, end=' ')
    print('\n')

if __name__ == '__main__':
    printArgv()
```

   This will cause the **printArgv()** function to be called before the window is displayed whenever **Lab5.py** is run. Try it to see what is printed in the shell.

10. Finally, let's try to invoke **Lab5.py** directly from a command prompt and let it display the arguments from the command line.

    Open a *Command Prompt* (i.e., a *Terminal* on the Macintosh).

    Use the **cd** command **c**hange **d**irectory to the directory/folder where you stored **Lab5.py** and **graphics.py**. If you need to list a directory, use the **dir** command in Windows or the **ls** command in Macintosh.

    Once you are in that directory/folder, type the following command on one line:–

```
python Lab5.py These are some arguments.
```

You may substitute any string for "**`These are some arguments.`**" On the Macintosh, be sure to use **`python3`** instead of **`python`**.

Notice that the arguments (including **`Lab5.py`**, which is the zeroth argument) are displayed in your command prompt. There is no *Python* shell to be seen. Notice also that the graphics window has been opened and is waiting for you to click it to close.

11. Congratulations! You have successfully run a *Python* program from a command prompt.

    You now have enough knowledge to implement a command line for the extra credit part of Homework #5. Remember, **`sys.argv`** is just a list of strings. It contains exactly the strings that you typed on the command line (not including **`python`** itself).

That's all for today! Be sure to save your **`Lab5.py`** file and submit it to *Canvas* under the assignment *Lab5*.

## Appendix — Setting the Environment Variable *Path*

In Windows 7 or Windows 8, open the *Control Panel* and select *System*. The *System* control panel window will open. In the left panel of this window, select *Advanced System Settings*. This will bring up the *System Properties* dialog. Select the *Advanced* tab, and click on the *Environment Variables* ... button at the bottom. This will open the *Environment Variables* dialog shown in Figure 1 below.

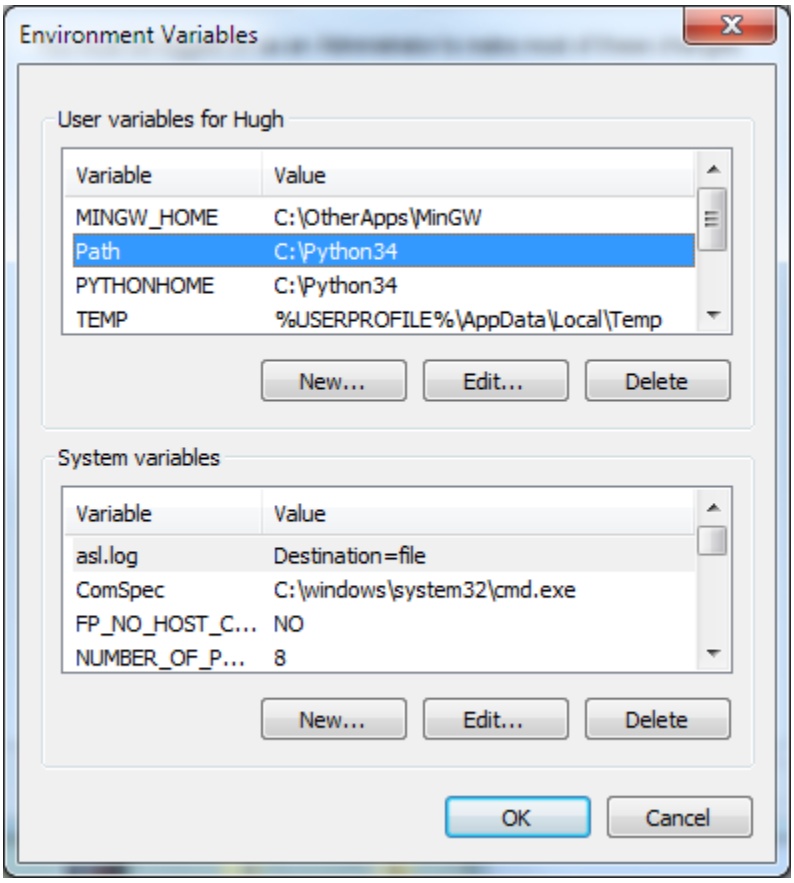


*Figure 1*

In the upper panel, click *New* …. This will bring up another dialog in which you can define a new *environment variable*. The name of this new variable must be "**`Path`**" and the value must be "**`C:\Python35`**" (or wherever your copy of *Python 3.5* is installed).

Click **`OK`**. You should now have an entry in the upper panel resembling the highlighted line in Figure 1. Close the Environment Variable and Control Panel windows, return to Step 8 and try the **`python`** command again.[3]

[3] *Environment variables* are variables that operating systems keep in order to record useful information and locations where useful stuff is kept. The system-wide environment variables are shown in the lower panel of Figure 1, and the user-specific environment variables are shown in the panel window. The *Path* environment variable is a list of directories, separated by semi-colons, that name the directories where the operating system searches for named programs to run. By adding *C:\Python35*, you have just told the system to also look there for programs to run. The **`python`** program is located in that directory (actually, it is named **`python.exe`**.)