# CSE 101: Computer Science Principles (Fall 2019)

## Lab #8

**Assignment Due: Saturday, November 16, 2019, by 11:59 PM**

### Assignment Objectives

By the end of this assignment you should be able to develop original Python functions to solve simple programming problems involving strings, dictionaries, and files.

### Getting Started

This assignment requires you to write Python code to solve several computational problems. To help you get started, we will give you a basic starter file for each problem. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct (**note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!**). You need to complete (fill in the bodies of) these functions for the assignments. **Do not, under any circumstances, change the names of the functions or their parameter lists.** The automated grading system will be looking for functions with those exact names and parameter lists; if you change any function headers, the grading program will reject your solution and mark it as incorrect.

### Directions

Solve each of the following problems to the best of your ability. We have provided a code skeleton for each of the programming problems. The automated grading system will execute your solution to each programming problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points. This assignment contains 3 problems, and is worth a total of 30 points. **Note that not every problem may be worth the same number of points!**

- ⚠ Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work. **If you leave this information out, you may not receive credit for your work!**
- ⚠ Submit your work as a set of individual files (one per problem). **DO NOT** zip or otherwise compress your files, and **DO NOT** place all of your work in a single file. If you do so, we may not be able to grade your work and you will receive a failing grade for this assignment!
- ⚠ Every function **MUST** use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).
- ⚠ Every function must explicitly ***return*** its final answer; the grading program will ignore anything that your code prints out. Along those lines, do **NOT** use `input()` anywhere within your functions (or anywhere before the `if __name__ == "__main__":` statement); your functions should get all of their input from their parameters. Programs that crash will likely receive a failing grade, so test your code thoroughly **with Python 3.7.4 or later** before submitting it.
- ⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your completed work as directed by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or that is submitted before you have signed the course's Academic Honesty agreement.
- ⚠ **ALL** of the material that you submit (for each problem) **MUST** be your own work! You may not receive assistance from or share any materials with anyone else, except for the instructor and the (current) course TAs.

## Part I: Extracting Hostnames (10 points)

(Place your answer to this problem in the "hostname.py" file)

**Function Summary**

**Parameters:** A string representing a single, complete URL.

**Return Type:** A string that contains **ONLY** the next-to-last section of the hostname from the function parameter.

A *Uniform Resource Locator* (URL) is a unique address for a document on the World-Wide Web. A URL has the following form:

*scheme://hostname/path/filename*

- *scheme* is a protocol such as `http`, `ftp`, `irc` or `file`
- *hostname* consists of 1 or more alphanumeric strings (with length 1 or more) followed by single periods, followed by a *top-level domain* (TLD) like `com`, `edu`, or `net`
- *path* is optional, and consists of 1 or more alphanumeric strings (with length 1 or more), each followed by a forward slash
- *filename* is also optional, and is an alphanumeric string (of length 1 or more)

**URL Examples**

```
irc://foo.b25.com/readme.txt
file://bar.edu/path/to/file.html
ftp://12q.net/
```

Complete the `hostname()` function, which takes a single string argument representing a URL and returns a string that corresponds to the *next-to-last* section of the hostname. For example, given the URL "http://www.example.com/", the function would return the string "example". Given the URL "ftp://this.is.a.long.name.net/path/to/some/file.php", the function would return the string "name". While the *path* and *filename* sections of the URL are optional, you may assume that the full hostname is always followed by a single forward slash ("/").

The easiest way to solve this problem is by using `find()` and `rfind()` to strategically and systematically remove pieces of the original string.

**Examples:**

| Function Call | Return Value |
|---|---|
| `hostname("irc://foo.com/")` | `foo` |
| `hostname("http://i.am.a.hostname.edu/blah/blah/")` | `hostname` |
| `hostname("ftp://some-like.it-hot.net/something/filename.pdf")` | `it-hot` |

## Part II: Indexing a Text File (10 points)

(Place your answer to this problem in the "index.py" file)

**Function Summary**

**Parameters:** A string representing the name of a plain text file to be processed.

**Return Type:** A dictionary whose keys are the unique words in the text file; each key's value is a ***list*** of integers.

Search engines like Google and Bing work by consulting an index of the words found in each Web page that they examine. This index identifies the document(s) that contain a particular word (like "dog" or "cat"), as well as the word's position(s) in those documents. For example, Google might record that the Web page "pets.html" contains the word "cat" at positions 22, 35, 100, and 209. Later, when a user searches for the word "cat", Google will return "pets.html" as a hit for that search term.

Complete the `index()` function. This function takes one argument: a string representing the name of a plain text file whose contents are to be indexed. `index()` returns a dictionary whose keys are the unique words in the text file; each key's value is a ***list*** of integer positions where that word appears in the original text. **The input file may contain multiple lines, but you may assume that every word is separated by a space, and that the file does not contain any external punctuation like periods or commas. If a word appears in varying cases, treat all of those instances as copies of the same (lowercase) word (for example, "Cat", "CAT", and "cat" all count as "cat").**

After processing the entire source file, `index()` returns a dictionary that represents the index of word locations.

**NOTE:** In order for your code to work properly, your data file **MUST** be located in the same directory as "index.py".

For example, a three-line file containing the text

```
Archimedes anticipated modern calculus and analysis by applying
concepts of infinitesimals and the method of exhaustion to derive and rigorously
prove a range of geometrical theorems including the area of a circle
```

would return a dictionary like

```
{"a":[21, 30], "analysis":[5], "and":[4, 11, 18], "anticipated":[1], "applying":[7],
 "archimedes":[0], "area":[28], "by":[6], "calculus":[3], "circle":[31],
 "concepts":[8], "derive":[17], "exhaustion":[15], "geometrical":[24],
 "including":[26], "infinitesimals":[10], "method":[13], "modern":[2],
 "of":[9, 14, 23, 29], "prove":[20], "range":[22], "rigorously":[19], "the":[12, 27],
 "theorems":[25], "to":[16]}
```

The exact order of the keys in your dictionary may vary; this will not affect the grading of your submission.

We have provided three files that you can use to test your code. The "Part II Sample Output.txt" file contains copies of the dictionaries that `index()` should produce for those files.

## Part III: A Multiple-Delimiter `split()` Function (10 points)

(Place your answer to this problem in the "multisplit.py" file)

**Function Summary**

**Parameters:** This function takes three parameters: a string to process (split), a list of strings representing delimiters, and a Boolean value that indicates whether the delimiters should be included in the final result.

**Return Type:** This function returns a list of strings.

We've already seen Python's `split()` method, which divides a string into parts using a single delimiter string. For this problem, we will develop a new function, named `multisplit()`, that can divide a string into parts based on multiple delimiters; each (single-character) delimiter will create a break in the string. `multisplit()` takes three arguments:

- a string to process
- a list of single-character strings representing valid delimiters
- a Boolean value indicating whether the function should also return the delimiters.

The function returns a list of string pieces that may include the delimiter characters (based on the value of the third argument).

For example, given the string `"when in the course of human events"` and the delimiter list `["e", "t", "u"]` (assume that the third argument is `False`), `multisplit()` will divide the first string at each character from the delimiter list, producing `["wh", "n in ", "h", " co", "rs", " of h", "man ", "v", "n", "s"]` — every 'e', 't', and 'u' character has been removed and used to break the original string into smaller pieces (if the third argument was `True`, this list would also include the delimiters: `["wh", "e", "n in ", "t", "h", "e", " co", "u", "rs", "e", " of h", "u", "man ", "e", "v", "e", "n", "t", "s"]`).

**NOTE:** If delimiters are included in the result, consecutive delimiter characters should be combined into a single substring.

**HINT 1:** Use two strings to store (respectively) any valid (non-delimiter) characters that you have seen since the last delimiter, and any delimiters that have been seen since the last valid character. As you encounter delimiter and non-delimiter characters, you will need to append these strings to your results list and/or reset them to the empty string from time to time.

**HINT 2:** When you finish processing the input, be sure to add any "leftover" characters to your list before returning it.

**Examples:**

| Function Call | Return Value |
|---|---|
| `multisplit("repetition", [], False)` | `["repetition"]` |
| `multisplit("repetition", ["e"], False)` | `["r", "p", "tition"]` |
| `multisplit("i am the very model of a modern major-general", ["p"], False)` | `["i am the very model of a modern major-general"]` |
| `multisplit("why is the sun yellow and not blue", ["m", "x", "z"], False)` | `["why is the sun yellow and not blue"]` |
| `multisplit("vampires and werewolves are mainly nocturnal creatures", ["a", "m", "n", "x", "z"], False)` | `["v", "pires ", "d werewolves ", "re ", "i", "ly ", "octur", "l cre", "tures"]` |
| `multisplit("repetition", [], True)` | `["repetition"]` |
| `multisplit("repetition", ["e"], True)` | `["r", "e", "p", "e", "tition"]` |
| `multisplit("i am the very model of a modern major-general", ["p"], True)` | `["i am the very model of a modern major-general"]` |
| `multisplit("why is the sun yellow and not blue", ["m", "x", "z"], True)` | `["why is the sun yellow and not blue"]` |
| `multisplit("vampires and werewolves are mainly nocturnal creatures", ["a", "m", "n", "x", "z"], True)` | `["v", "am", "pires ", "an", "d werewolves ", "a", "re ", "ma", "i", "n", "ly ", "n", "octur", "na", "l cre", "a", "tures"]` |