**CISC 487 / 680 SU 2019**

**Applied Artificial Intelligence International Boot Camp**

**Lab Booklet #03**

**Lab Title: Procedural Programming & Search Algorithms**

### Instructors: Prof. Shalaan PhD, Prof. Grey M.S., Prof. Chu M.S.

### TA's: Aditya Syal, Bao Pham, Zach Pelkey

# Objectives

In order to be able to effectively program applications in Python, you need to be able to create sub-programs, also known as subroutines, procedures, or functions, to capture a behavior in a single place. This process will also help with code readability, modularity, and reuse. We are also going to be practicing recursion, or the ability of a procedure to call itself. This functionality is core to understanding programming languages and being a computer scientist.

To this end, we are going to look at some common search algorithms and ask you to write functions that will implement these algorithms. As we are going to be working extensively with data stored in file formats such as spreadsheet or delimited formats, we are incorporating these file types into our lab.

After completing this lab, you will be able to:

* Create Python procedures
* Understand the principles of code reuse and modular design
* Understand variable scoping in Python
* Be able to write recursive functions
* Understand the uses for, requirements of, and differences between Sequential and Binary Search

# Resources

While the lab will explain everything that you need to know, you may want to consult the following sources for additional knowledge:

<https://en.wikipedia.org/wiki/Comma-separated_values>

<https://www.tutorialspoint.com/python3/string_split.htm>

<https://en.wikipedia.org/wiki/Linear_search>

<https://en.wikipedia.org/wiki/Binary_search_algorithm>

<https://www.shmoop.com/computer-science/recursion/head-tail.html>

# Deliverables

For this lab, you will need to submit:

* This lab book, completed as instructed.
* Python code files (.py), named and coded as instructed. In this lab, you will need to submit:
  + Search.py

# Instructions & Questions

# Part 1: File Handling

In order to complete this lab, you will need to be able to read and search a comma-separated values, or CSV, file. In a CSV file, rows of data are separated by new line characters but values within a row are (obviously) separated by commas.

In order to read from a file, you will need three different commands:

* <identifier> = open(<file-name>, "r")
  + This command opens a file for reading and stores the reference in your <identifier>.
  + Note that <file-name> can be a relative or absolute path to the appropriate file.
* <identifier> = open(<file-name>, "r", encoding="utf-8-sig")
  + **NOTE:** this line is for Mac users.
* <list> = <identifier>.readlines()
  + This command reads the contents of the file referenced by <identifier> into memory.
  + <list> will contain a list of strings where each string represents the entire contents of a row.
  + Keep in mind that each row will contain all characters including the associated new line control character (\n) at the end of the line.
* <identifier>.close()
  + This command closes the connection to the file.
  + When you have found and retrieved the appropriate data, it is best practice to then close your file connection.

Finally, there is a string command that might also come in handy:

* <list> = <string>.split(<delimiter>)
  + This command splits <string> into a list of component elements based on the <delimiter>.

# Part 2: Linear Search

Linear Search is an elementary search technique. When searching for an element in a list, Linear Search attempts to find the element by starting at one end of the list and checking each element until the desired element is found or until the search reaches the end of the list.

Part 1 of the lab you will need to download and reference the file data.csv.

1. Given a list of 256 elements, what is the best-case scenario for how many elements of the list Linear Search will have to check before it finds the desired element? What is worst-case scenario for the number of elements to check? What is the average case?

**Ans: Eg=[3,9,20,40,54,60,74,79,83,87,91,93,97,99]**

**Best Case: find the element at the 0th index itself.**

**Lets, say we are we are finding 3, 3 itself is the first element. In asymptotic notation it is O(1)**

**Worst Case:The element does not exist in the array. In asymptotic notation it is O(n).**

**Avearge Case: Element can be found anywhere between 1st index and last Index. In asymptotic notation it is O(n)**

1. Create a file called Search.py and inside create a function called IterativeLinear. The function will search a given row of data.csv using an iterative implementation of Linear Search that you have programmed from scratch. The function will expect the following parameters IN THIS ORDER:

* The 0-indexed row number in the csv file to be searched
* The element to be searched for

This function should retrieve the row of data from data.csv specified by the first parameter and try to find the element specified by the second parameter. You may assume that searched for parameter and the elements of the list to be comparable. The function should return the index of the element in the row or -1 if the element doesn’t exist.

1. Alter Search.py. Inside create a function called RecursiveLinear. The function will search a given row of data.csv using a recursive implementation of Linear Search that you have programmed from scratch. The function will expect the following parameters IN THIS ORDER:

* The 0-indexed row number in the csv file to be searched
* The element to be searched for

This function should retrieve the row of data from data.csv specified by the first parameter and try to find the element specified by the second parameter. You may assume that searched for parameter and the elements of the list to be comparable. The function should return the index of the element in the row or -1 if the element doesn’t exist.

# Part 3: Binary Search

Binary Search is a more complex search technique. When searching for an element in a list, Binary Search attempts to find the element by starting in the middle of the list. If the desired element is less than the element checked in the list, the search algorithm then checks the element in the first half of the list. If the desired element is greater than the element checked in the list, the search algorithm then checks the element in the second half of the list. This subdivision and checking until the element is found or cannot be found.

Part 2 of the lab you will need to download and reference the file data2.csv.

1. Binary Search cannot be attempted on every list. What is required of the list for Binary Search to be successful? Why?

**ANS: Binary cannot work on all lists. The List should be sorted. Middle element should be found in the first iteration itself.**

1. Given a list of 256 elements, what is the best-case scenario for how many elements of the list Binary Search will have to check before it finds the desired element? What is worst-case scenario for the number of elements to check? What is the average case?

**Ans: Best case: The element is found in the middle. In asymptotic notation it is O(1).**

**Worst case - O (log n) comparsions**

**In the worst case, the item X does not exist in the array A at all.**

**Average case- O (log n)**

**not at the 1st iteration itself.**

1. Alter Search.py. Inside create a function called IterativeBinary. The function will search a given row of data.csv using an iterative implementation of Binary Search that you have programmed from scratch. The function will expect the following parameters IN THIS ORDER:

* The 0-indexed row number in the csv file to be searched
* The element to be searched for

This function should retrieve the row of data from data2.csv specified by the first parameter and try to find the element specified by the second parameter. You may assume that searched for parameter and the elements of the list to be comparable. The function should return the index of the element in the row or -1 if the element doesn’t exist.

1. Alter Search.py. Inside create a function called RecursiveBinary. The function will search a given row of data.csv using an recursive implementation of Binary Search that you have programmed from scratch. The function will expect the following parameters IN THIS ORDER:

* The 0-indexed row number in the csv file to be searched
* The element to be searched for

This function should retrieve the row of data from data2.csv specified by the first parameter and try to find the element specified by the second parameter. You may assume that searched for parameter and the elements of the list to be comparable. The function should return the index of the element in the row or -1 if the element doesn’t exist.

# Part 3: Head vs. Tail Recursion (Bonus)

You have created two different recursive functions: RecursiveLinear and RecursiveBinary. However, recursive functions can be programmed in one of two different styles.

Head recursion is a style of recursion where the calculation is done on the way up the call stack. As each recursive call is made, the value being calculated is generated within the parameter list of that recursive call. This means that when a base case is reached, the functions simply return the already calculated value.

Tail recursion is a style of recursion where the calculation is done on the way back down the call stack. As each recursive call is made, the value being calculated is generated in the return statement. The recursive call simply passes on the information necessary for the recursive call to function properly. This means that when a base case is reached, a basic value is returned. Then, as each function’s context is restored down the call stack, the necessary value is then calculated.

These two styles are simply complementary; there is no meaningful performance difference between the two. However, some solutions lend themselves to one style over another. Similarly, some developers simply prefer one style over another.

1. **BONUS (2½%)**: In your implementation of RecursiveLinear, did you use head or tail recursion? Explain in detail.

**ANS: Tail recursion have recursive call in the last statement. It is tail recursion.**

1. **BONUS (2½%)**: In your implementation of RecursiveBinary, did you use head or tail recursion? Explain in detail.

**ANS: It is tail recursion**

1. **BONUS (10%)**: Alter Search.py. Inside create a function called RecursiveLinear2. In Question 8, you identified your implementation of RecursiveLinear as either head or tail recursion. For this question, you need to implement this function using the other style of recursion. All other requirements are identical to Question 3, above.
2. **BONUS (10%)**: Alter Search.py. Inside create a function called RecursiveBinary2. In Question 9, you identified your implementation of RecursiveBinary as either head or tail recursion. For this question, you need to implement this function using the other style of recursion. All other requirements are identical to Question 7, above.

\*\*JUst Attempted, but errors\*\*\*