# Software for Digital Innovation (CIS4044-N) Tutorial 4: File I/O

## Before You Start

Ensure you have completed all tasks from Tutorial 3. Any issues seek help from your tutors.

**Hint:** In Python, you can search within strings in the same way you can search within a list, using the in keyword.

>> "sand" in "sandwich"

True

>> "abc" in "abacus"

False

Attempt to complete this set of tasks before your next tutorial.

## Introduction

This session aims to familiarise you with writing to, reading from, appending to and deleting from files.

## Question 1: Sandwich Menu

Write an interactive sandwich menu for a restaurant.

1. First, in Notepad, create a file that contains the names of different types of sandwiches, one per line (e.g. “Tuna Mayo”). Call this file menu.txt.
2. Write a program that asks the user for the name of a sandwich, then politely tells them whether or not there is a sandwich with that name in menu.txt.

## Question 2: Smart Sandwich Menu

You are tasked with making the interactive menu from the previous question smarter!

1. Extend your answer to the previous question to not only search for a complete string match but search *within* each item to see if the ingredient is present.
2. For example, a search for “Tuna” should return “Tuna Mayo”, “Tuna Sweetcorn” and any other sandwich with tuna in its name.
3. Make the search ignore uppercase and lowercase letters (read about Python’s case functions lower() and upper() here: <https://www.geeksforgeeks.org/isupper-islower-lower-upper-python-applications/>)
4. See this week’s hint (above) for a pointer on how to achieve this.

## Question 3: Loading Cars

Write an application to load information about cars from a file and search that information.

1. Download cars\_exercise.txt from Blackboard and open it in Notepad. Notice its structure:
   1. 3 lines describe one car - line 1 is make, line 2 is model and line 3 is year.
   2. There is information about 3 cars present in the file.
2. Now, write a script to load these 3 cars as dictionaries from the file. Each dictionary should have 3 keys: make, model and year.
3. Now, print out the make, model and year information for the oldest and newest car.

## Question 4: Splitting the Car Database

Write a program which splits the car database from question 3 into smaller files.

1. Write a script that will read in the car database used in question 3 and write out 3 files named as make\_model\_year.txt containing information about one car only. For example, mitsubishi\_outlander\_2015.txt should contain:

Mitsubishi

Outlander

2015

1. Now add another car to the original file and run your program again. The program should still work as expected, producing 4 files this time instead of 3. If it does not do this, modify it so that it does.

## Question 5: Removing the Years

Write a program that will remove all years from the car database used in questions 3 and 4.

1. Write a script that will read in the car database used in questions 3 and 4 and write it out again with all years removed.

## Question 6: Moving the Years

Write a program that will combine the year associated with a car with its model number:

1. Adapt your answer to question 5 to prepend the year to the model number instead of deleting it from the file completely. The resulting file should look like this:

Hyundai

2006 Tucson

Mitsubishi

2015 Outlander

Ford

1998 Focus

## Extension Activity: A More Sophisticated App

Write a car management app for a car showroom.

**Note:** This activity is designed to be challenging! If it seems difficult, that’s because it is! Don’t give up and approach it with curiosity, rather than dread. It’s worth it I promise!

1. Produce an app which first reads all cars from cars\_exercise.txt used in previous exercises into a list of dictionaries as in question 3, then asks the user if they’d like to add or remove a car.
2. If the user specifies that they’d like to add a car, ask them for the make, model and year of the car and add it to the list of dictionaries, then write the list of dictionaries back to the text file, formatted in the same way as the original file (3 lines per car: make, then model, then year). **Be careful here not to lose data!**
3. If the user specifies that they’d like to remove a car, ask them to specify a make, model and year. Check the list of dictionaries for a car that fulfils these criteria, ignoring letter case (i.e. in a case-insensitive way—see question 2).
4. If one is found, ask the user if they’re sure they want to delete it. If they choose yes, remove the car in question from the list of dictionaries and write out the file again. If they choose no, skip to point 6.
5. If one is not found, tell the user that the app could not find a car that satisfies the criteria.
6. Loop back around to point 1 and ask the user if they’d like to add or delete a car again. This should continue until the user types ‘exit’.

## Document History

Revision 0 (15-Oct-20): This is the initial version of the 2020/21 exercise.