★DS

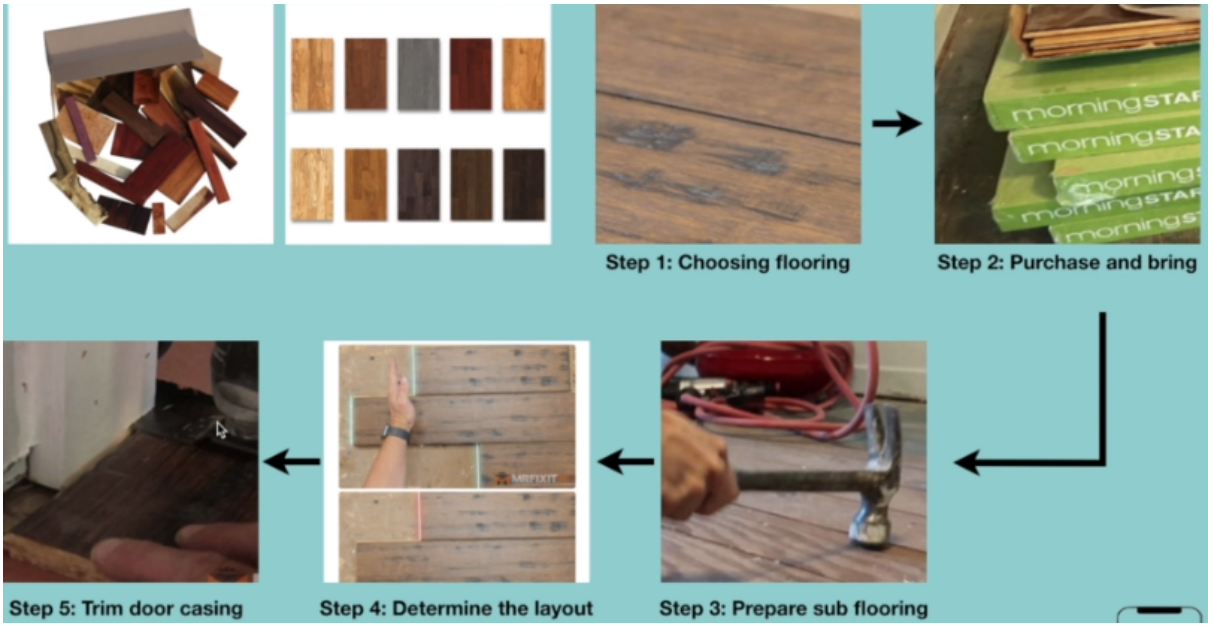
Different ways of organizing data in your computer to be used later in an effective way! (how do they organized n group together and stored during program execution!)
All softwares ------> data-driven

In every-day life(get a concert ticket!----> queue—->first in first out)

Book—-> return back to library —> not able to carry

★ Algorithms:

Set of tasks to accomplish a task! (task flooring!)



★ ALGO in CS:

A set of rules to accomplish a task!

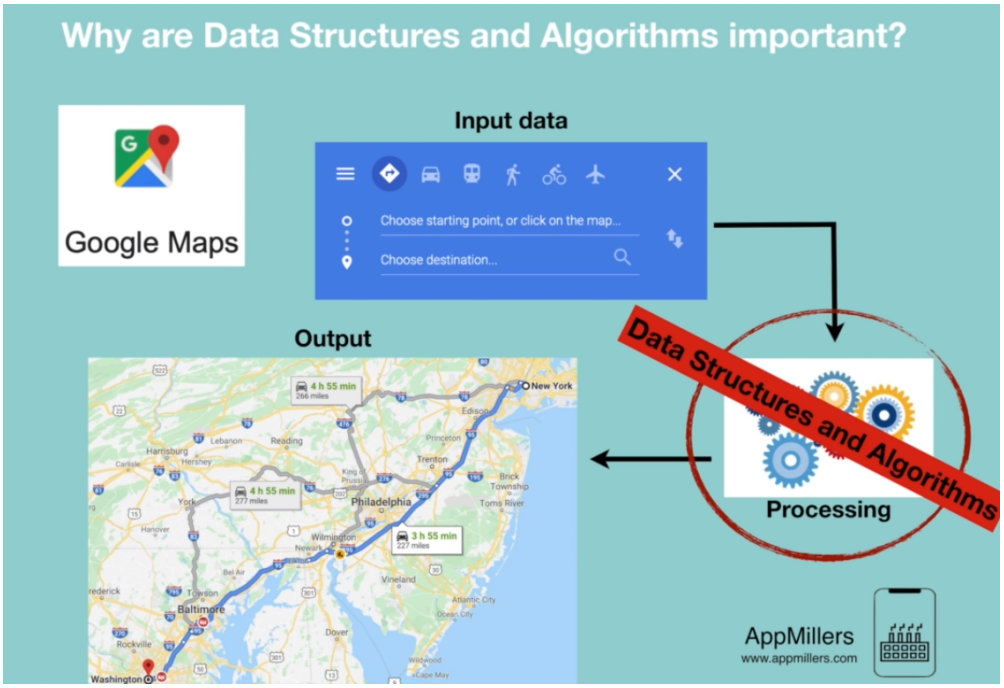
Allow us to write memory efficient programs!

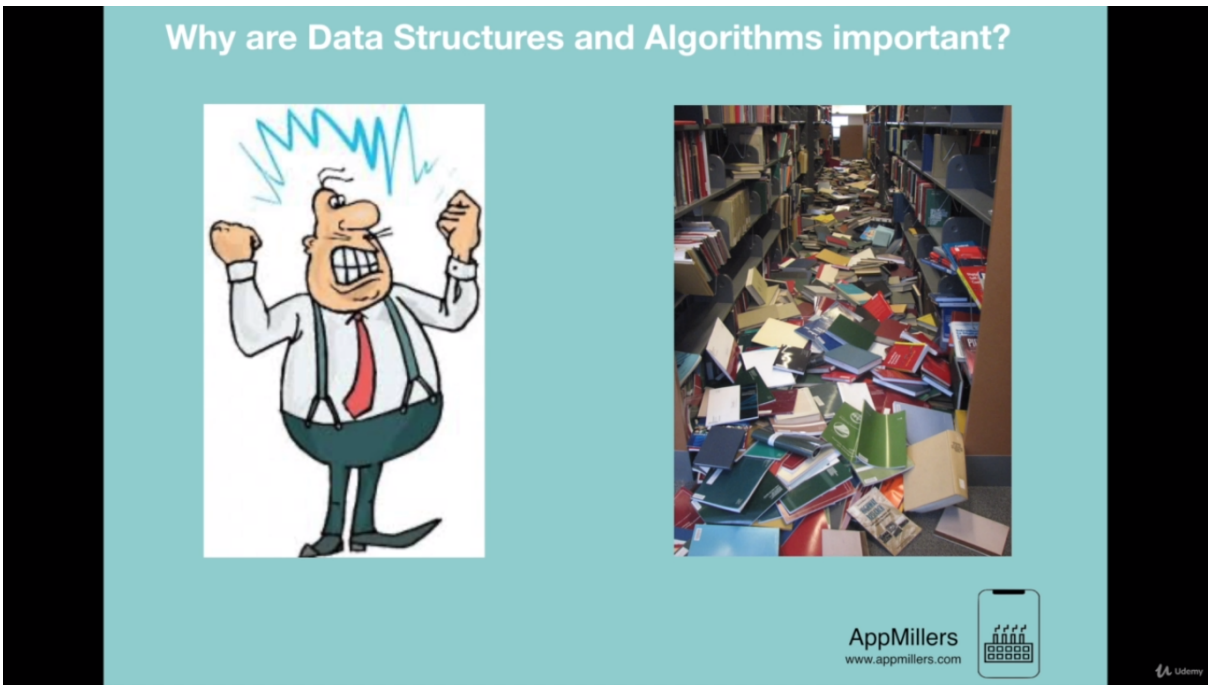
Input data —–> calculation —–--> stop when find the answer!

Important factors:

1)**correctness:** the algorithm doing things in a correct way!

2)**efficiency:** do it in an efficient way!





Here:

- ➔ Books: Data
- ➔ DS: Arranging the books
- ➔ Algo: The way we find a book!

★ Recursion Algo: function calling itself recursively!

Important factors in Recursion:

- ➢ **Performing the same operation multiple time in smaller input of the similar type!**
- ➢ **Base condition to stop the recursion loop!**

```
def openRussionDoll(doll):
    if doll==1: #when we reach to the end!
        print('all dolls are opened!')
    else:
        openRussionDoll(doll-1) #call the function on smaller pieces!
```

☐ When to use recursion:

1. If ya can divide problem to similar sub-problem
2. Any key in the Question so that we note as a Recursion candidate?
    - Design an Algo to compute nth
    - Write a code to list n…
    - Implement a method to compute all…
3. In Trees n Graphs!
4. Used in many Algo; Divide n Conquer/ Dynamic/ Greedy

☐ How Recursion Works?

1. A method calls itself w/ smaller values
2. Exit from infinite loop
3. Stack memory is used for managing Recursive calls! So every time the Recursive method calls itself, the system stores it in stack for the system to come back bc there are some executions left statements left after calling itself, so this means the system somehow should remember the deploy where to stop to call different parameter base on the stop condition!
4. Any Problem that can be solved by Recursion can also be Solved by Iteration!

☐ When to Use/Avoid Recursion?

Use:

1. When it can be easily divided to similar sub-problems!
2. Always remember it brings extra overheads(both time----> for push/pop outta stack + Space---->for storing in stack!)
3. When we wanna a quick solution! (factorial/fibonacci)
4. If the input is pretty small or become smaller
5. Traverse of tree
6. When use memoization

Avoid:

1. time/space complexity matter to us!
2. Memory limited(for exp. Mobile application, low memory)

3. **Recursion is slow!**

How to write a Recursive Function?

- **Recursive Case – flow**
- **Stop Condition**
- **Unintentional Case!**

NOTE: in Recursion since we are doing the problems for smaller values in each iteration, then in most cases the condition would go for 0 or 1!

Sample interview Q in Recursion:

- **Sum of digit of an int number**
- **POWER**
- **GCD**
- **Convert Decimal to Binary**