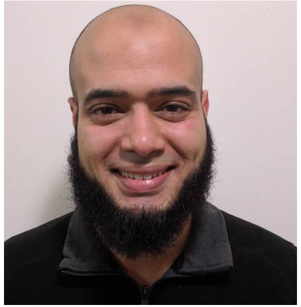
Data Structures

Data Structure: **What and Why**

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

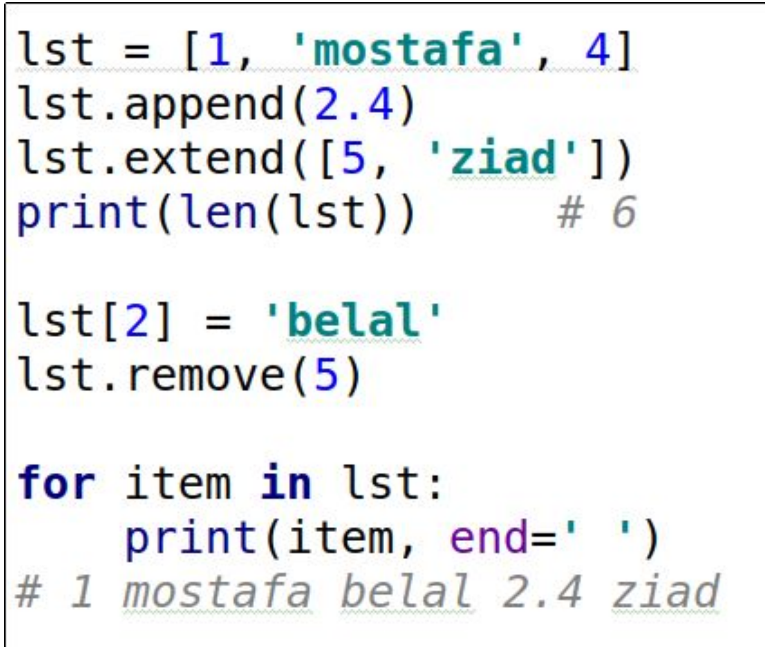


A Data Structure (DS)

- A data-structure is just a **class**!
- This means it combines **two components**
  - **Organization of data**
  - **operations** on the data
- During a Python programming course, you should have already studied many built-in data structures many **built-in data** structures
  - List, Tuple, Set and Dictionary (please review well)

The **List** Data Structure

- What are the allowed methods?
  - A lot!
  - Create a list
  - Append item
  - Extend with items
  - Delete an item
  - Access item
- The same for the other data-structures
  - E.g. Tuple, Set and Dictionary
  - A lot of support!



```
lst = [1, 'mostafa', 4]
lst.append(2.4)
lst.extend([5, 'ziad'])
print(len(lst))     # 6

lst[2] = 'belal'
lst.remove(5)

for item in lst:
    print(item, end=' ')
# 1 mostafa belal 2.4 ziad
```

Common Questions

- **Why do we need data structures?**

- **Why are built-in data structures supported?**

- **Why do we study data structures if we have built-in ones?**

Why do we need data structures?

- If you have implemented code in a 500+ line project, you have probably met scenarios where you need complex ways to both manage your data, and perform operations on it
- In a hospital or restaurant system, you need to maintain the queue of people
  - Who came first (first in first out principle)
  - What their order is
  - Other relevant statistics: when was the person served? How much was the bill? etc...
- With just basic int/float/character, we are very constrained!
- Data structures (aka classes), provide such great flexibility!

Why Built-in Data Structures?

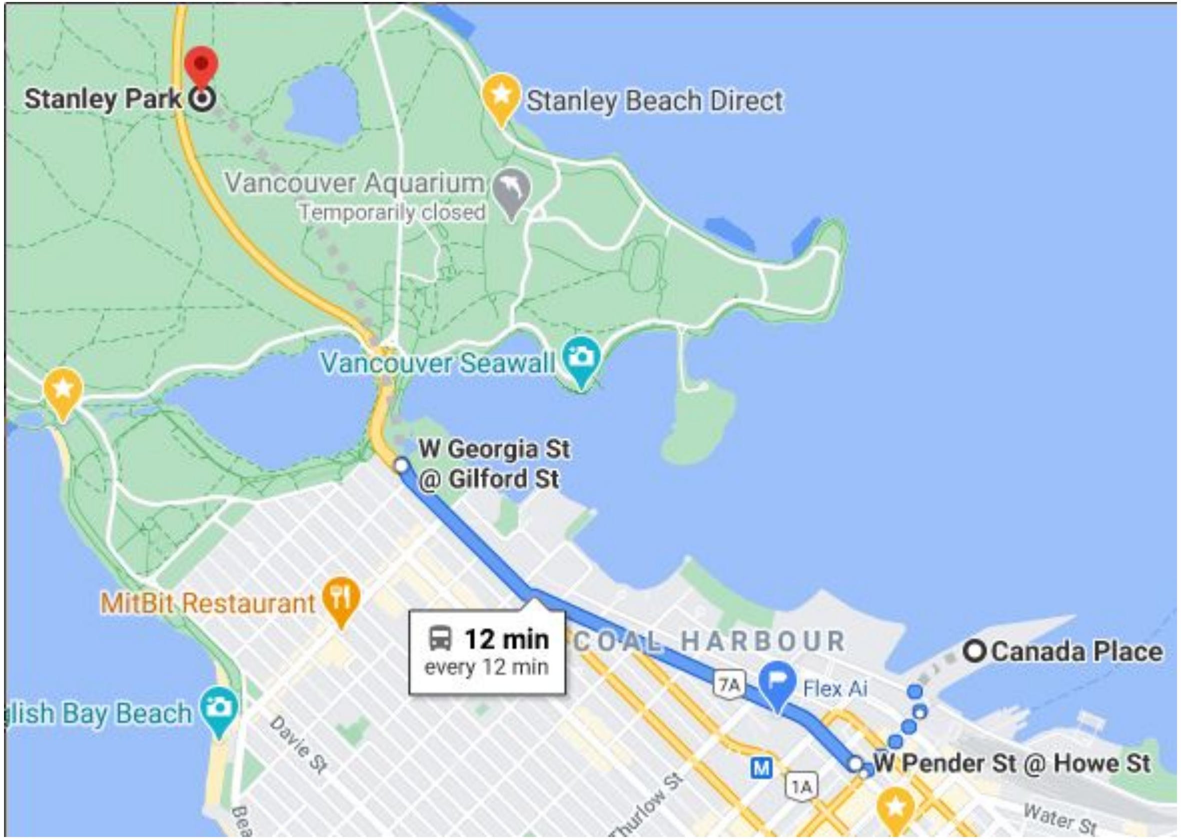
- Throughout the history of software engineering, certain types of **data+operations** are common and repetitive
    - Some are basic, such as what are now known as the List, Stack, and Queue data structures
        - E.g. A queue data structure is useful for queues in *restaurants and hospitals*
    - Some are advanced, such as Hash-Table and AVL Trees
- It's time-consuming and repetitive to **re-**implement these from scratch
    - Many languages implement them: STL in C++, Collections in Java/C#, etc
- During this course, we will learn the **inner details** of these built-in DS

Why not just use built-in DS?

- Using built-in data structures as a **black box** without understanding their details is a big risk in real projects
    - Typically you won’t realize the **time/memory** order
    - Typically you will use them **improperly** as you don’t recognize their differences
- These built-in data structures were built to serve **only** common scenarios
- What if we face **new** scenarios relevant to our project?
- Example: search for a specific word in a 1 billion word article
        - List is very slow,
        - We can invent new data structures
            - **For example: Trie or Suffix Tree** data structures are much faster!
- Example: Google Maps

Google Maps

- We use maps to navigate from one place to another
- We need a **data structure** to represent **points and streets**!
  - We have a lot of data!
  - Constraints: time/date/car
- We need **efficient** functions to find **optimal paths** (*algorithms area*)



Why not just use built-in DS?

- What if we face **new** scenarios relevant to our project?
- We need to **create** a **new data** structure to provide flexibility in dealing with something very centered around this data
  - A **user defined data-type**: your own class, which serves a specific purpose
- To be effective, you need several skills!
  - E.g. what are the different data organization **perspectives** that we may use?
    - What are the pros/cons of them?
    - The time/memory differences?
    - The tradeoffs?
- Side effect
  - Your thinking skills are improved
  - This is good for problem-solving & algorithms courses

Data Organization Perspective

- From one purpose to another, we may arrange the same **data** in different ways
  - There are many ways to implement a data structure
- When we have huge amounts of data (think of the billions of users on Facebook), things become much more complex and critical!
  - E.g. Search the engines for scientific or social purposes
  - E.g. If a storm hits a specific list of locations, how many homes will face a power outage?
  - E.g. We are in a war, and want to destroy the **minimum number of bridges** in a city to disconnect 2 points of our enemy? Rockets are expensive!

Data Structure Efficiency

- Assume we have N (10000) employees
- Is a loop over these N employees as fast as 3 nested loops?
  - No, it seems like 10000 operations vs 10^12 operations!
  - It looks like the first is efficient, but the 2nd is not!
- So how to measure the efficiency of a function?
  - The **complexity** (asymptotic) **analysis** in the algorithms field answers that.
  - The efficiency can be for **time and memory (space)**
  - For the same problem, we may arrange data in a way that is so **fast** in computations, or we could look for one which is optimal in terms of **memory** efficiency
    - You might be lucky, and find a DS that is both time and memory efficient!
  - On mobiles: you may target a memory efficient approach
  - On real-time services: you may target a time efficient approach

Normal class vs Data structure

- It seems a data structure is simply a class with data and methods!
- Why do we give it a special name? Because of how we perceive it
- Data structure is very **centered around data** to provide specific functionalities
    - It is mainly about the data. Specific functionalities are **driven** and tailored to the data
    - Every data structure **arranges & stores** data in a specific way to support a specific use case
    - Queue data structure orders that data to follow: **First in First Out order**, like restaurants
- A normal class is **centered around functionalities**. I want a Student class, in which I can not only store student information and add students, but also assign each student grades, as well as mark and print student assignments
    - Employee, Payroll, Question, Answer, Email, etc. All are classes of **business** logic
- Eventually, both have data + operations, but **different views**

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."