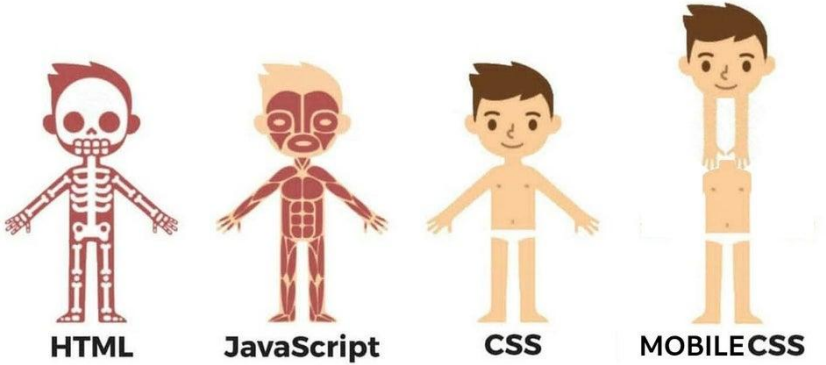
COMP10001

WORKSHOP #9

Shevon Mendis
shevonm@unimelb.edu.au

HTML

- **H**yper-**T**ext **M**arkup **L**anguage
  - **Markup languages** are designed for the processing, definition and presentation of text. The **language** specifies code for formatting, both the layout and style, within a text file.
- When you load a web page, your browser downloads the HTML file and renders that on the screen, usually with some styling (CSS)
- HTML is used to define the **structure** of a web page
  - Tags are used to specify the formatting



HTML

- Entities: special characters that need to be written in a special syntax:
  - `&< entity >;`
    - `>` → `>`
    - `<` → `<`
    - `&` → `&`
- Static page: Where the information on the page remains the same until manually changed- every user that loads this page will see the exact same thing all the time. Eg. *most* portfolio sites
- Dynamic page: Where the page's content is different for different users. Dynamic websites often make use of databases to support this tailored user experience. Eg. Grok, Youtube, Google

Further info: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-web-pages/

Algorithms

- An algorithm is a set of steps for solving an instance of a particular problem type.
- As the size of our inputs increase, the efficiency of our algorithms becomes important and hence we study how to write them well
- Programming is simply learning how to write code: how to use correct grammar and structure when writing in a programming language so that a computer can understand what we intend to communicate. Studying algorithms is learning about good code: elegant ways of solving problems and how to use more advanced vocabulary to write more powerful coding sentences.

Judging Algorithms

- We judge algorithms on 2 factors:
  - Correctness
    - Does the algorithm produce the right answer for all inputs?
  - Efficiency
    - How much time and memory does the program take?

Exact vs Approximate

- An exact approach calculates a solution with a guarantee of correctness
- An approximate approach estimates the solution so may not be as correct
  - If the problem is too complex to calculate, it may be more feasible to use an approximate approach. Eg. Geo-location

Brute Force

- Also known as Generate & Test
- Exact approach
- Finds the solution by enumerating every possible answer and testing them one-by-one. This requires answers to be within a calculable range that allows the program to run to completion: if there are billions of options, it may take years to do this, rendering a brute-force approach useless to that problem
- Example:
  - Password cracking by generating all the possible combinations of characters

Heuristic Search

- Also known as Greedy Search
- Approximate approach
- Finds a solution by using a more efficient approximate method than one which is completely correct. The answer may not be optimal but is often “close enough” especially when the alternative is a significantly slower algorithm.
- Example:
  - When finding the shortest distance from a source to a target, instead of evaluating where every single path will take you, you can pick the next (intermediary) point as the one that has the closest straight-line distance to the target

Simulation

- Approximate approach
- Finds a solution by generating a lot of data to predict an overall trend
- Examples:
  - Simulating a game of chance to test whether it's worth adding to a casino. Many rounds of results can be generated and combined to find whether, on average, money was won or lost. In the real world, the outcome may be different, but by running simulations we can find the most likely outcome
  - Weather simulations/ predictions based on past data

Divide & Conquer

- Exact approach
- Finds a solution by dividing the problem into a set of smaller sub-problems which can be more easily solved, then combining the solutions of those sub-problems to find the answer of the overall problem.
- Examples:
  - Binary search
  - Sorting algorithms (eg. Quicksort, Mergesort)