

# Advanced Programming

Ramaseshan Ramachandran

- ① Introduction
  - Expectations
  - Logistics
- ② Algorithmic Patterns
  - Simple Patterns
  - Divide and Conquer
  - Greedy Algorithms
  - Dynamic Programming
  - Backtracking

Some more patterns

- ③ Programming Paradigms
  - Procedural Programing
  - OOP
  - Functional Programming
  - Declarative Programming
  - Logic Programming
  - Event-Driven Programming,
  - Multiprocessing, and Multi threading

- ▶ Course Information
- ▶ Introduction to Programming paradigms
- ▶ Introduction to Basic Programming in Python
- ▶ Data structure to represent knowledge and its practical use cases
- ▶ Object oriented programming
- ▶ Functional Programming
- ▶ Analysis of Algorithms - Time and space complexities

# EXPECTATIONS!

---

- ▶ The course does not require any background in programming, but requires considerable dedication and hard work
- ▶ Attend all the classes
- ▶ Submit assignments on time. There will be no extensions
- ▶ Learn more about your machine
- ▶ Practice, practice..., practice<sup>∞</sup>

# COURSE LOGISTICS

---

- ▶ Grades are based on quizzes, mid semester exam, assignments and End semester Exams
- ▶ There will be at least 6 quizzes
- ▶ There will be at least 8 assignments
- ▶ Quizzes and Mid/End semester exams - 60%
- ▶ Assignments - 40%
- ▶ Submit all assignments using [Colab](#)

## Books

- ▶ [1] *Introduction to Algorithms, Third Edition*
- ▶ [2] *Algorithm Design*
- ▶ [3] *Algorithms*
- ▶ [Python official Documentation for 3.12.1](#)

# Algorithmic Patterns

# SIMPLE PATTERNS

---

- ▶ An algorithmic pattern describes a core solution strategy for solving a problem
- ▶ There are/may be several solutions to the same problem
- ▶ It is important to examine the problem and choose the right technique
- ▶ Simple Patterns
  - ▶ Sequential
  - ▶ Iterative
  - ▶ Recursive
- ▶ Brute Force

# DIVIDE AND CONQUER

---

Break down a problem into smaller, self-similar subproblems - solve them iteratively or using recursive approach

- ▶ Merge sort
- ▶ Quick sort
- ▶ Matrix Multiplication
- ▶ Fast Fourier Transform
- ▶ Nearest Neighbors



Make locally optimal choices at each step, hoping to reach a global optimum

- ▶ Dijkstra's shortest path algorithm
- ▶ Beam search

Breaking the problem into multiple overlapping subproblems

- ▶ Levenshtein distance
- ▶ Travelling Saleman

- ▶ Explore possible solutions by building a solution incrementally
- ▶ Backtrack (undo choices) if a solution path doesn't lead to a valid solution
  - ▶ N-Queens problem - placing 8 queens in a chess board in a way that no two queens attack each other
  - ▶ Sudoku solver item Route planing

## MORE ALGORITHMIC PATTERNS

---

- ▶ Hill climbing - Finding an approximate solution to a problem by generating candidate solutions that are better solutions over the previous candidate solutions
  - ▶ Machine Learning algorithms, many image processing algorithms
- ▶ Branch and bound
  - ▶ Making a round trip by visiting  $N$  cities - find the shortest path that visits all cities with the minimum total distance
  - ▶ - Assembly line of manufacturing
- ▶ Monte Carlo
- ▶ Particle Swarm Optimization
- ▶ Las Vegas

# Programming Paradigms

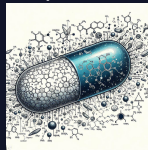
- ▶ Breaks the programs into multiple procedures/subroutines
- ▶ Uses statements that change a program's state, like assignments and control flow structures (if-else, while, for loops)

# OBJECT ORIENTED PROGRAMMING

---

- ▶ Abstraction
  - ▶ Hide the internal working of the function - we will what it does but will not know how it does
- ▶ Encapsulation
  - ▶ Placing data and methods in a single unit - restricting and or protecting internal variables from direct access to prevent the unauthorized use of data
- ▶ Inheritance
  - ▶ Eliminate redundancy
- ▶ Polymorphisms
  - ▶ Taking multiple forms of actions using a single interface to the outside world

Examples - Banking application, capsule, car



Function as First class citizen - Treats computation as the evaluation of mathematical functions

- ▶ Uses functions to transform inputs into outputs without modifying external data - Avoids side-effects

## Example

Lisp, Haskell, Clojure, Scala



# DECLARATIVE PROGRAMMING

---

Specifies the desired output, not the exactly how it should be achieved

Example

Structured Query Language

Expresses problems as logical statements and relationship

## Example

Prolog - Uses rules and inference mechanisms to derive solutions

# EVENT-DRIVEN PROGRAMMING

---

- ▶ Program flow driven by events (user actions, system signals, messages)
- ▶ Structure:
  - ▶ Program idle until an event occurs
  - ▶ Event loop monitors for events and triggers event handlers
  - ▶ Event handlers execute specific code blocks
- ▶ Use cases: GUI applications, web servers, game development, network communication, embedded systems

- ▶ Key concept: Executing multiple processes concurrently
- ▶ Structure:
  - ▶ Each process has independent memory space and resources
  - ▶ Processes communicate through IPC mechanisms
- ▶ Use cases: CPU-intensive tasks, parallel computations, independent tasks

- ▶ Key concept: Executing multiple threads within a single process
- ▶ Structure:
  - ▶ Threads share memory space and resources
  - ▶ Threads communicate and synchronize more easily than processes
- ▶ Use cases: I/O-bound tasks, improving GUI responsiveness, web servers

## REFERENCES

---

- [1] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.
- [2] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison Wesley, 2006.
- [3] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. 1st ed. USA: McGraw-Hill, Inc., 2006. ISBN: 0073523402.