DSA105 - FUNDAMENTALS OF ALGORITHMS AND PROGRAMMING

W1-W2

Data Science Before the AI Revolution: A Historical Context

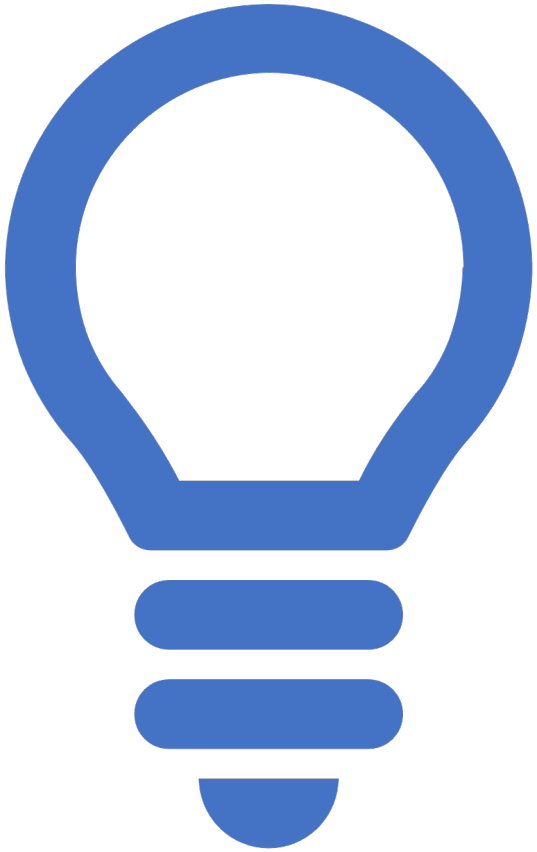
- Before the modern AI revolution transformed data science with large language models and automated machine learning, data scientists worked with a fundamentally different toolkit and mindset.

The Four Types of Early Data Scientists

- 1. Data Businesspeople
- 2. Data Creatives
- 3. Data Developers
- 4. Data Researchers

Data Businesspeople

- **Profile**: The strategic translators between data and business decisions
- **Primary Focus**: Converting statistical insights into business value and profit
- **Typical Role**: Director of Analytics at Fortune 100 companies
- **Key Skills**:
  - Business strategy and management
  - Statistical analysis and visualization
  - Presenting to executives ("translating p-values into profits")
- **Educational Background**: Often Industrial Engineering + MBA
- **Work Reality**: More time in meetings than desired, but crucial for organizational impact
- **Tools**: Traditional statistical software, business intelligence platforms
- **Challenge**: Bridging the gap between technical analysis and business strategy



Data Creatives

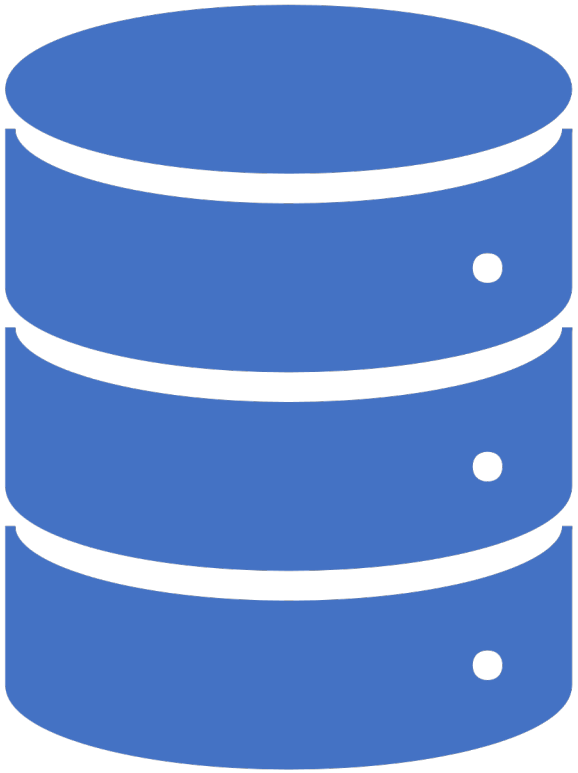
- **Profile**: The versatile innovators and prototype builders
- **Primary Focus**: Applying diverse tools creatively across the entire data pipeline
- **Typical Role**: Interactive visualization developer, hackathon participant
- **Key Skills**:
  - Data visualization and storytelling
  - Web development (JavaScript, HTML, CSS)
  - Prototyping and experimentation
  - Open source contribution
- **Educational Background**: Economics with Computer Science minor
- **Work Reality**: Multi-faceted projects, strong community involvement
- **Tools**: D3.js, Python plotting libraries, web-based visualization
- **Philosophy**: "Jack of all trades" approach to data problems

Data Developers

- **Profile**: The technical architects of data systems
- **Primary Focus**: Building scalable, maintainable machine learning systems
- **Typical Role**: ML Engineer at consulting firms, production system developer
- **Key Skills**:
  - Machine learning algorithms (SVM, neural networks)
  - Big data technologies (Hadoop, distributed computing)
  - Software engineering and system architecture
- **Educational Background**: Master's in Computer Science
- **Work Reality**: Writing production-level code, optimizing algorithms
- **Tools**: Scala, Java, Hadoop ecosystem, mathematical libraries
- **Challenge**: Balancing algorithmic sophistication with system reliability

Data Researchers

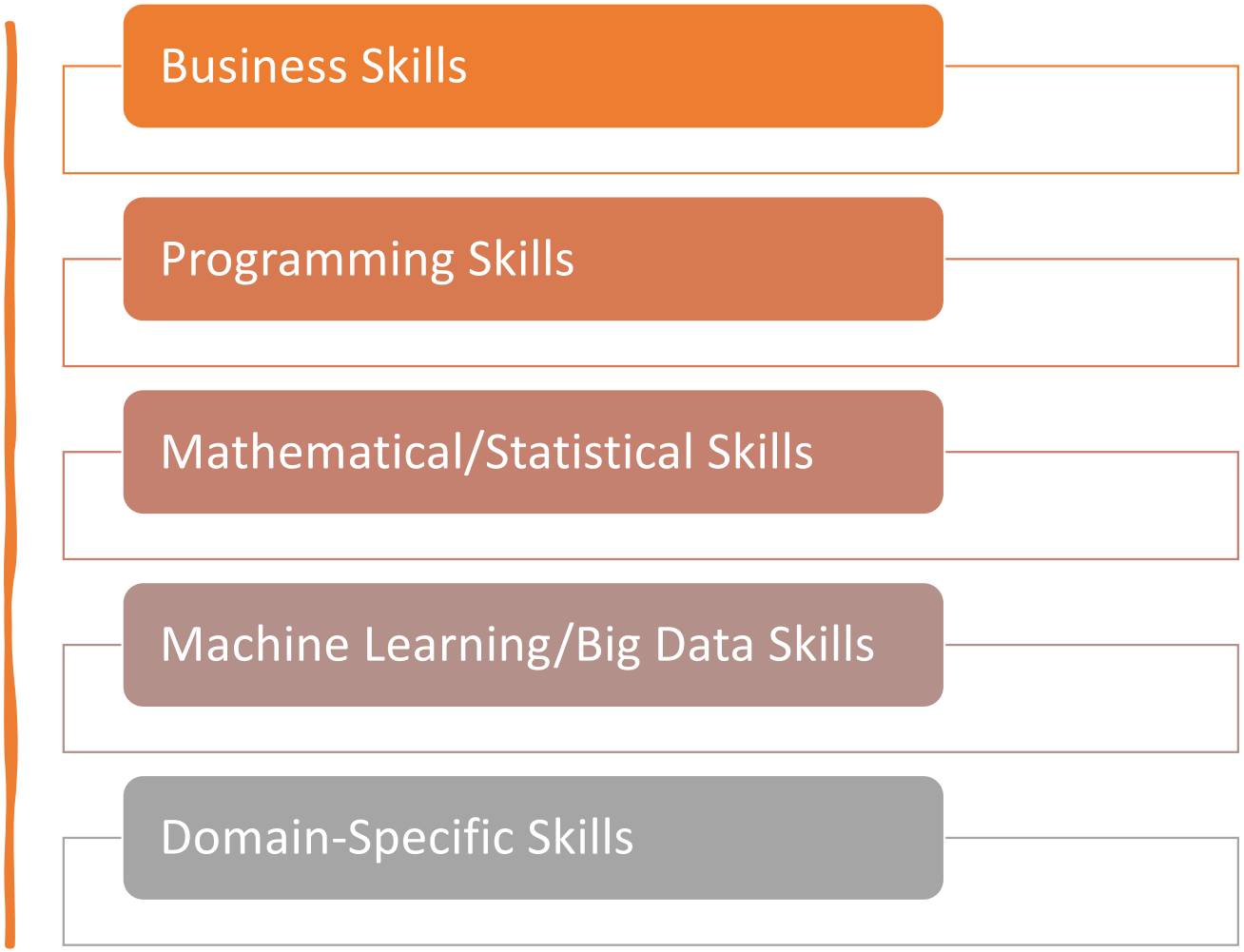
- **Profile**: The scientific method applied to business problems
- **Primary Focus**: Rigorous statistical analysis and hypothesis testing
- **Typical Role**: Scientists transitioning from academia to industry
- **Key Skills**:
  - Advanced statistical methods
  - Experimental design
  - Academic research methodology
  - Scientific communication
- **Educational Background**: PhD in physical or social sciences
- **Work Reality**: Applying academic rigor to business questions
- **Tools**: R, SAS, statistical modeling packages
- **Transition**: Moving from "publish or perish" to "insights that drive decisions"



The Complete Data Pipeline (Pre-AI)

- **Data Extraction**: Writing custom scripts to gather data from various sources
- **Data Cleaning**: Manual data quality assessment and correction
- **Data Integration**: Combining datasets from different systems and formats
- **Statistical Analysis**: Applying mathematical models and hypothesis testing
- **Visualization**: Creating charts and graphs to communicate findings
- **Interpretation**: Drawing business insights from statistical results
- **Tool Building**: Developing reusable scripts and analytical frameworks

Skills Required in the Pre-AI Era



Business Skills

- Management and leadership
- Product development
- Business strategy

Programming Skills

- Systems administration
- Back-end development
- Front-end development

Mathematical/Statistical Skills

- Classical statistics (ANOVA, regression)
- Bayesian methods
- Optimization techniques
- Algorithms and computational complexity

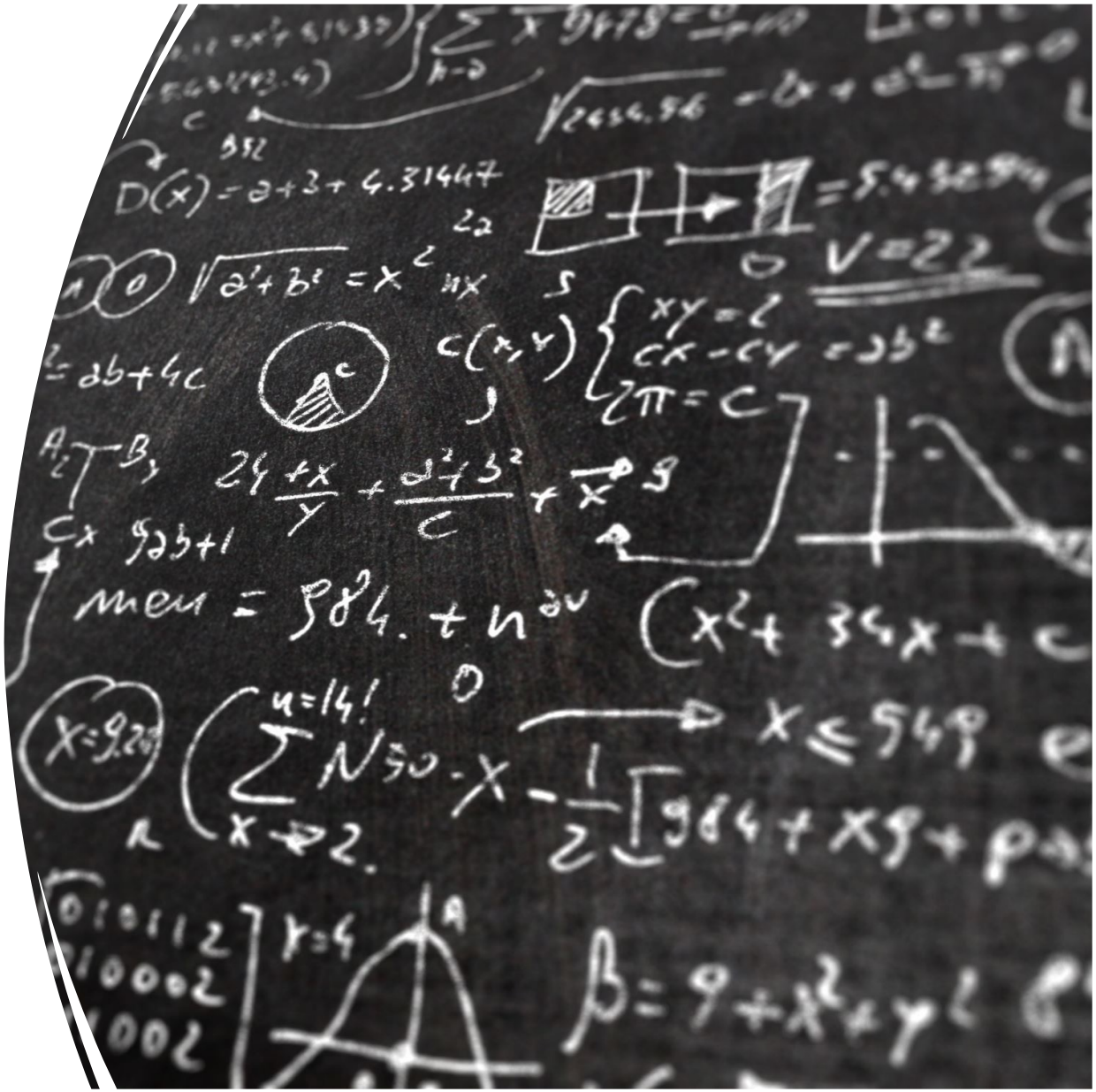
Machine Learning/Big Data Skills

- Decision trees and clustering
- Neural networks (basic)
- Distributed computing
- Unstructured data processing

Domain-Specific Skills

- Spatial statistics
- Time series analysis
- Survey methodology

Problem-Solving Stages, Introduction to Algorithms, and Flowcharts



The Problem-Solving Process

- **1.1 Why Problem-Solving Skills Matter in Programming**
- Programming is fundamentally about **solving problems**. Before writing any code, successful programmers follow a systematic approach to understand and break down problems.
- **Real-World Example**: Netflix Recommendation System
- **Problem**: How does Netflix know what movies you might like?
- **Solution**: Not magic, but a step-by-step algorithmic process
- **Key Point**: Complex systems start with clear problem-solving methodology

The Five-Stage Problem-Solving Process

1. UNDERSTAND the Problem
2. PLAN the Solution
3. IMPLEMENT the Solution
4. TEST the Solution
5. DEBUG and REFINE

Stage 1: UNDERSTAND the Problem

- **Questions to Ask:**
- What exactly needs to be solved?
- What information do I have? (inputs)
- What result do I need? (outputs)
- Are there any constraints or limitations?
- **Example Problem**: Calculate student grades
- **Input**: Test scores, homework scores, participation
- **Output**: Final letter grade (A, B, C, D, F)
- **Constraints**: Grading scale (90-100 = A, 80-89 = B, etc.)

Stage 2: PLAN the Solution

- **Key Activities:**
- Break the problem into smaller sub-problems
- Identify the steps needed
- Consider different approaches
- Choose the most suitable method
- **Example Planning**: Student grade calculation
    1. Collect all scores
    2. Calculate weighted average
    3. Compare average to grading scale
    4. Assign letter grade

Stage 3: IMPLEMENT the Solution

- **What This Means:**
- Write the actual code (we'll learn this in coming weeks)
- Follow your plan step-by-step
- Start simple, then add complexity

Stage 4: TEST the Solution

- **Testing Strategies:**
- Use sample data with known results
- Test edge cases (very high/low values)
- Check for errors and unexpected behavior

Stage 5: DEBUG and REFINE

Common Activities:

- Fix errors found during testing
- Improve efficiency
- Make code more readable
- Add error handling

Problem-Solving Strategies

- Divide and Conquer
- Work Backwards
- Pattern Recognition

Divide and Conquer

Break large problems into smaller, manageable pieces

Example: Planning a trip

Large problem: "Plan a vacation"

Smaller pieces: Choose destination → Book flights → Find accommodation → Plan activities

Work Backwards

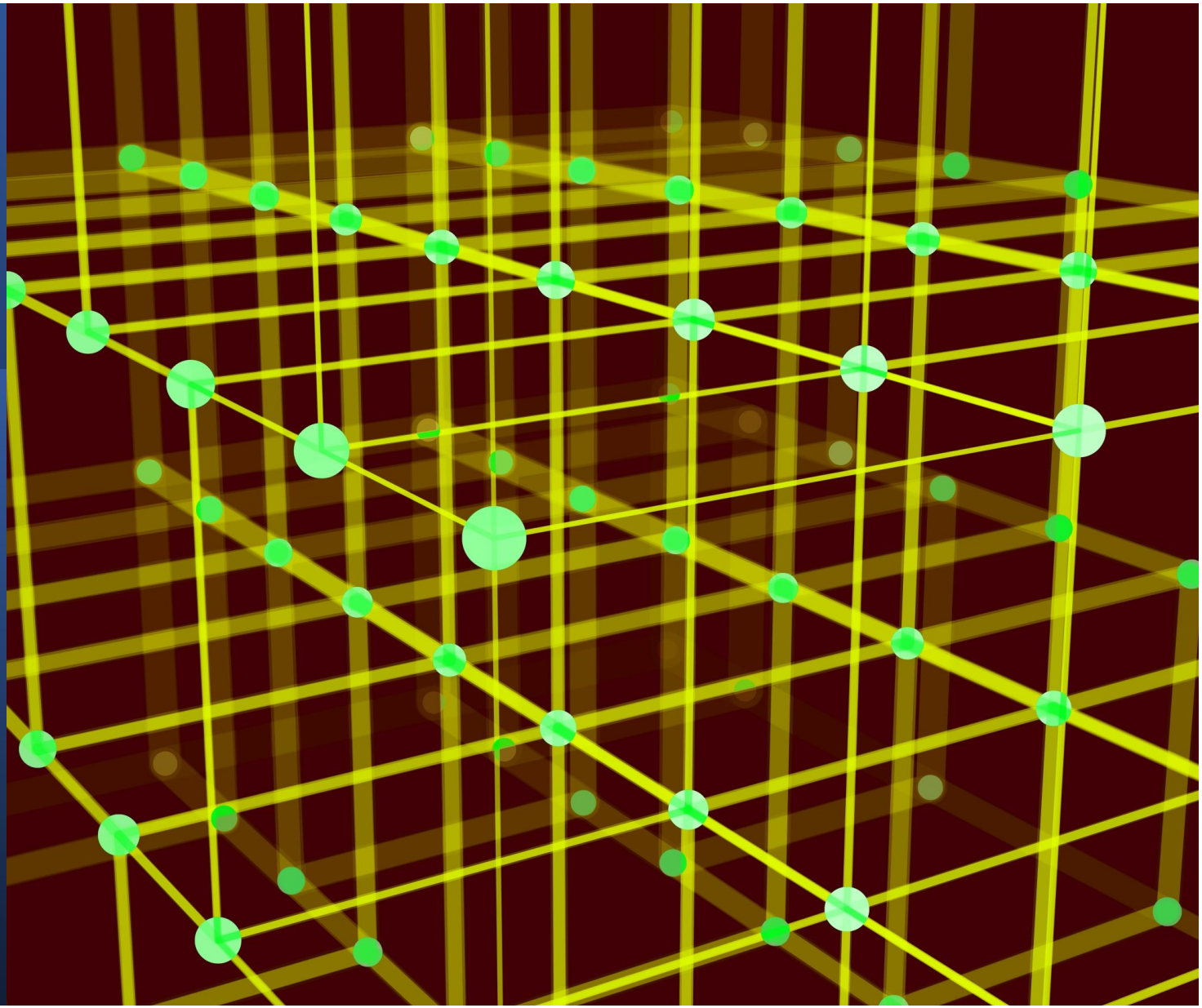
Start with the desired outcome and work backwards to the starting point

Example: Baking a cake for 6 PM dinner

- 6:00 PM: Serve cake
- 5:30 PM: Decorate cake
- 4:30 PM: Cake finishes baking
- 3:30 PM: Start baking
- 3:00 PM: Begin preparation

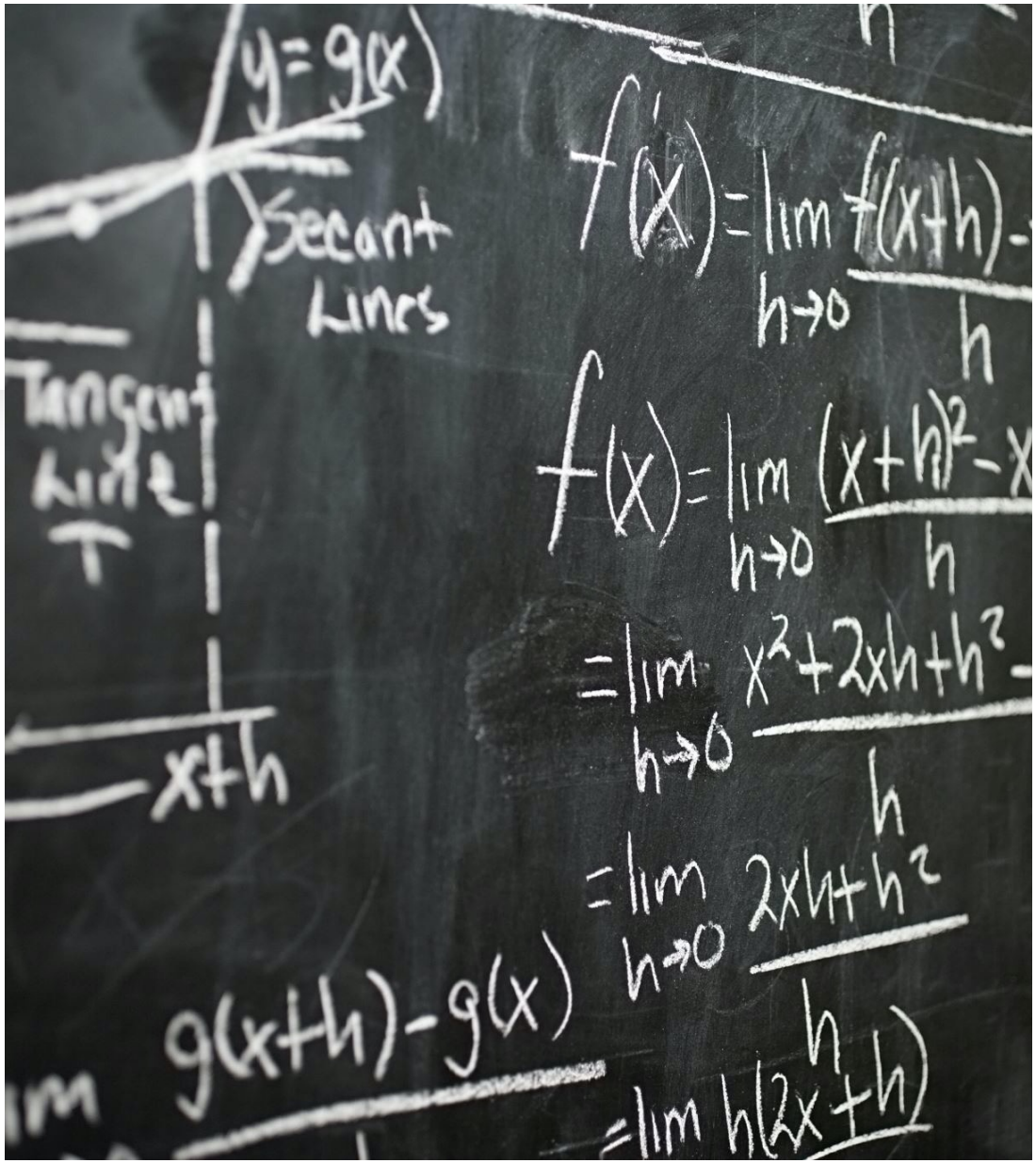
Introduction to Algorithms

An algorithm is a step-by-step procedure for solving a problem or completing a task.



What is an Algorithm?

- **Definition**: An algorithm is a step-by-step procedure for solving a problem or completing a task.
- **Key Characteristics:**
  - **Clear and unambiguous**: Each step must be precisely defined
  - **Finite**: Must have a definite beginning and end
  - **Effective**: Each step must be achievable
  - **Input/Output**: Takes input and produces output



Algorithm Characteristics

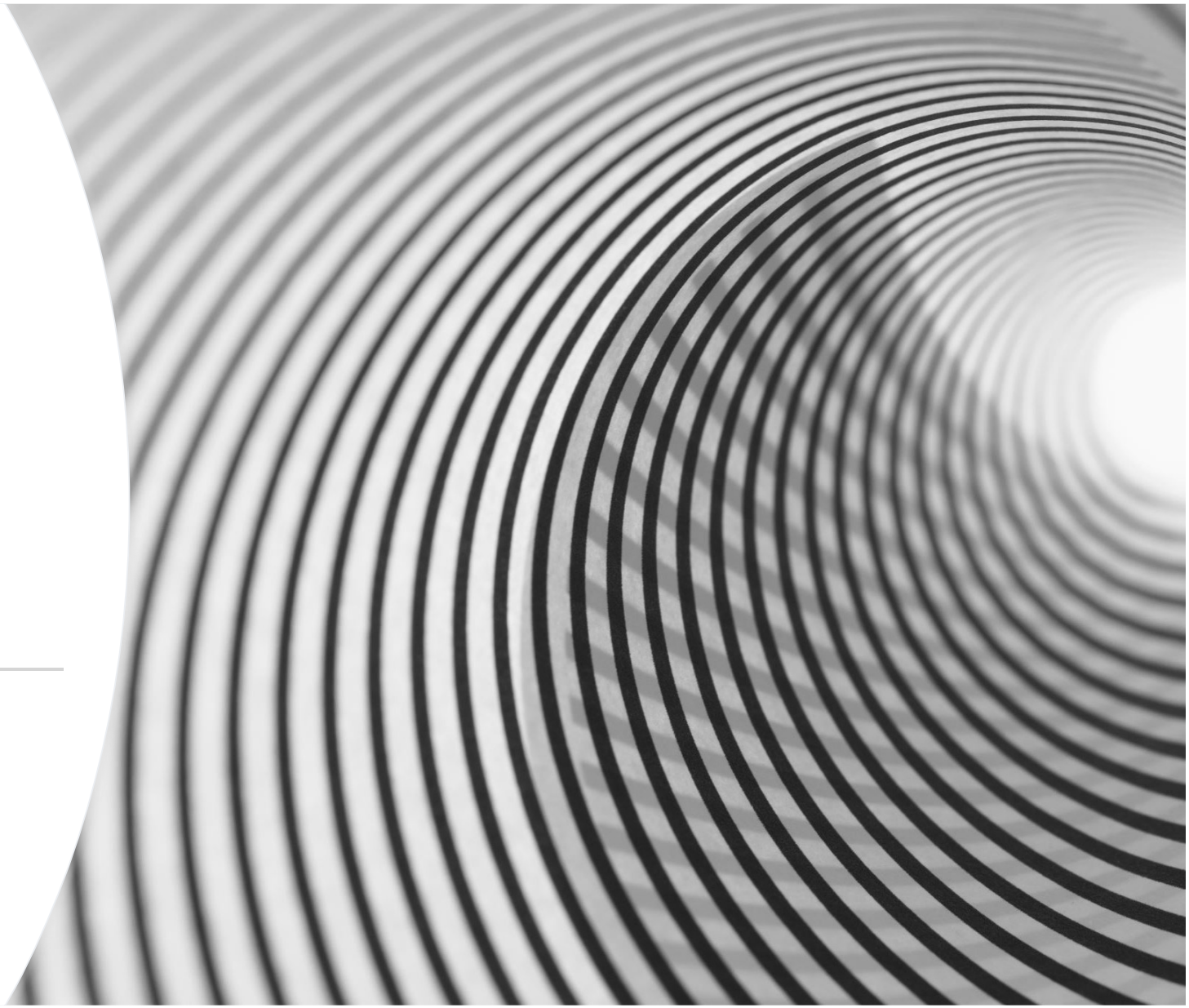
Precision Matters

Consider All Scenarios

Types of Algorithm Structures

- **Sequential (Linear)**
  - Steps performed one after another
  - Step 1
  - Step 2
  - Step 3
- **Conditional (Decision-Making)**
  - Different paths based on conditions
  - IF condition THEN do A
  - ELSE do B
- **Repetitive (Loops)**
  - Repeat steps multiple times
  - WHILE condition is true, keep doing X
  - FOR each item in list, do Y

Flowcharts - Visualizing Algorithms



What are Flowcharts?

- **Definition**: Visual representations of algorithms using standardized symbols and connecting arrows.
- **Benefits:**
- Make algorithms easier to understand
- Help identify logical errors
- Facilitate communication between team members
- Serve as documentation

Flowchart Symbols

Basic Symbols

Oval: Start/End (Terminal)

Rectangle: Process/Action

Diamond: Decision/Question

Parallelogram: Input/Output

Circle: Connector

Arrow: Flow direction

Flowchart Best Practices

Flow Direction: Top to bottom, left to right

One Entry/Exit: Each flowchart should have one start and one end

Clear Labels: Use descriptive text in symbols

Avoid Crossing Lines: Keep flowlines clean and uncluttered

Consistent Size: Keep symbols roughly the same size