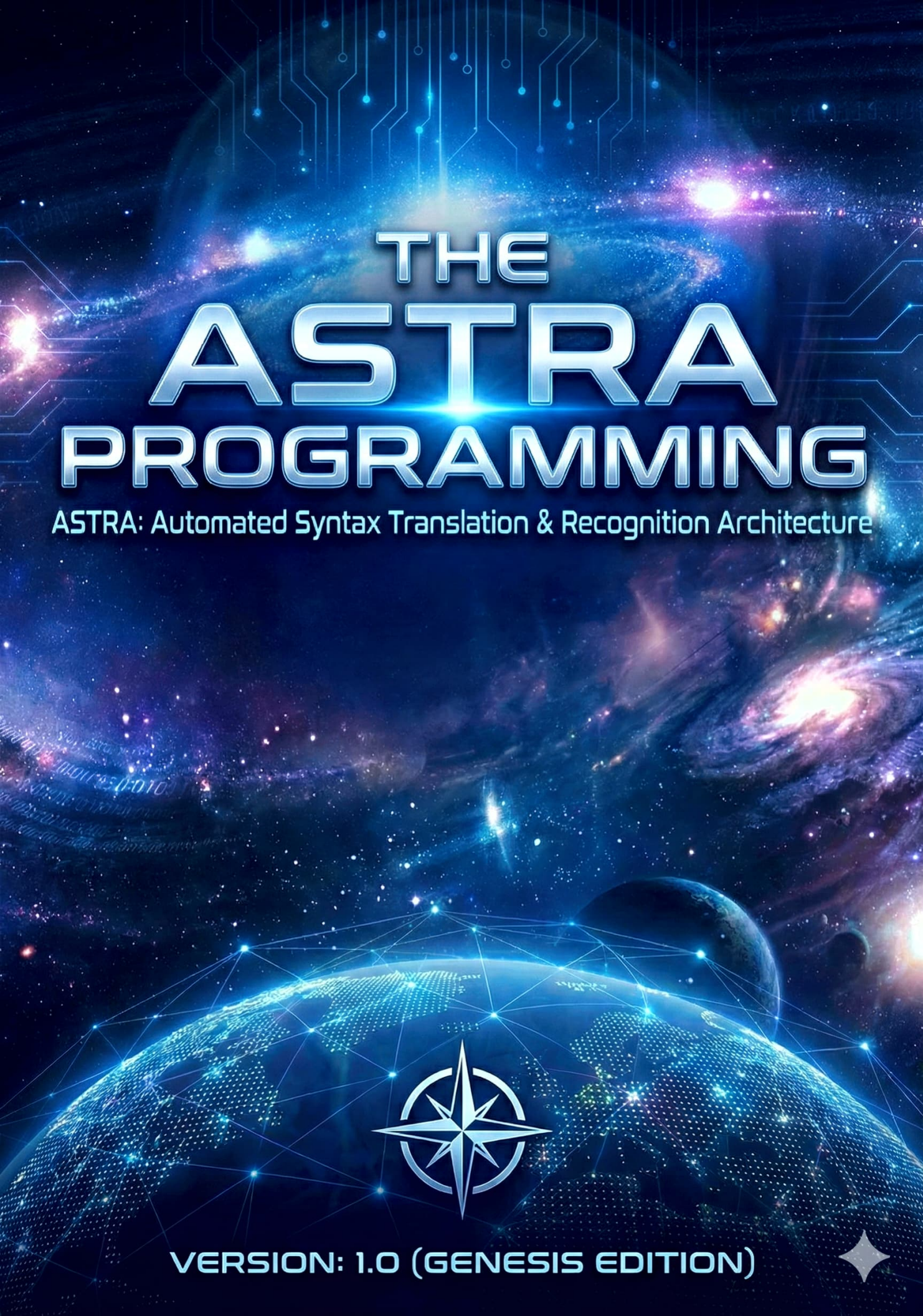

THE
ASTRA
PROGRAMMING
ASTRA: Automated Syntax Translation & Recognition Architecture
VERSION: 1.0 (GENESIS EDITION)

INSPIRING WORDS ABOUT ASTRA PROGRAMMING

“Astra empowers everyone to turn ideas into reality through simple, natural language commands. A true revolution in coding.”

“Finally, a programming language that bridges the gap between human thought and machine execution. Elegant and powerful.”

“The 'If you can say it, you can code it' motto is not just a slogan; it's the core philosophy of Astra. Transformative.”

LEGAL & COPYRIGHT NOTICE

By Syed Ismail Nasser ©
B.E Robotics and Automation

ASTRA PROGRAMMING – THE DEFINITIVE GUIDE

Automated Syntax Translation and Recognition Architecture

Version: 1.0 (Genesis Edition)

Official Reference Manual

Author: Syed Ismail Nasser

For Students, Educators, Software Engineers, and AI Practitioners

---

About This Book

Programming languages shape how we think about computation. Astra represents a fundamental rethinking of this relationship—designed not for machine efficiency, but for human cognition, collaborative development, and AI-assisted programming.

This publication-grade technical guide transforms the Astra Language Specification into a comprehensive learning resource featuring:

- Conceptual foundations grounded in cognitive science
- Visual execution models with detailed diagrams
- Worked examples with complete outputs
- Professional case studies from data science, AI, and systems programming
- Progressive exercises from beginner to advanced

By the final chapter, we will understand how Astra achieves Python-level expressiveness while maintaining the readability of structured English.

Dedication

For my beloved parents,

MoqthiarUnissaNasser and Syed Nasser H.

whose love, guidance, and sacrifices shaped the person I am today.

Acknowledgements

I am deeply grateful to everyone who has supported me on this journey.

To my mentors, teachers, and colleagues who fuelled my curiosity and guided my steps into the world of robotics, data science, and artificial intelligence; Thank you for your wisdom and encouragement. To my friends and peers, who believed in me, challenged my thinking, and pushed me to grow; your support has been invaluable. To the communities of learners, professionals, and innovators I've been fortunate to engage with; your conversations and insights have shaped the reflections in this book. And finally, to my family, for their unwavering love, sacrifice, and belief in me.

This book is not just mine; it is the result of every hand that lifted me, every word that inspired me, and every challenge that made me stronger.

TABLE OF CONTENTS

PART II: COMPUTATION & LOGIC

Chapter 5: Operations & State Mutation

- Why Astra Forbids Inline Expressions
- Explicit Mutation Commands
- Type-Aware Operations
- Preventing Common Errors

Chapter 6: Control Structures

- The Sandwich Architecture
- Conditional Logic (If/Else/Else if)
- Iteration Models (Repeat/Loop/Iterator)
- Flow Control (Stop/Skip)

Chapter 7: Error Handling & Safety

- Attempt-Recover-Always Model
- Failure Flow Visualization
- Resource Safety
- Nested Error Handling

PART III: STRUCTURING PROGRAMS

Chapter 8: Functions as Reusable Intent

- Functions as Recipes
- Call Stack Visualization
- Input/Output Flow
- Return Values

- The Await Mechanism
- Event-Driven Programming
- Concurrency Models

PART V: DATA SCIENCE & MACHINE LEARNING

Chapter 14: Data Tables & Analysis

- Tabular Data Model
- Loading and Inspecting Datasets
- Data Cleaning Pipeline
- Column Operations

Chapter 15: Data Visualization

- The Visualization Pipeline
- Chart Types and Use Cases
- Axes and Relationships
- Visual Analysis

Chapter 16: Machine Learning Fundamentals

- ML as a Pipeline
- Training Predictors
- Model Evaluation
- Making Predictions

Chapter 17: Neural Networks & Deep Learning

- Layered Architecture
- Computer Vision
- Natural Language Processing

APPENDICES

PART I: FOUNDATIONS OF ASTRA

Chapter 1: The Philosophy of Astra

Programming as Intent

1.1 Why Astra Exists

Programming languages are tools for translating human intent into machine execution. For over six decades, most languages have optimized for compiler efficiency, hardware constraints, and mathematical notation—accumulating symbols, implicit rules, and historical complexity.

Astra begins from a different premise:

> *If code reads like intent, correctness becomes easier to achieve.*

This is not simplification for its own sake. It is engineering discipline applied to the fundamental question: **How should humans communicate computational logic?**

Astra is designed so that programs can be:

- **Read aloud** without loss of meaning
- **Explained visually** before execution
- **Generated and verified** reliably by AI systems
- **Understood by domain experts** without programming background

1.2 The Four Design Laws

Astra is governed by four non-negotiable design principles that distinguish it from traditional languages:

Law 1: One Action Per Line

Each line expresses exactly one state change or decision point.

Rationale: Cognitive load increases exponentially when multiple operations are compressed into single statements. By enforcing one-action-per-line, Astra makes program flow traceable and debuggable.

Law 2: No Symbolic Operators

Symbols like `+`, `==`, `{}`, and `[]` are replaced with explicit English commands.

Rationale: Symbols require memorization and create ambiguity. Words carry semantic meaning that remains consistent across contexts.

Law 3: Explicit Block Boundaries

Every logical block has visible start and end markers.

Rationale: Indentation-based scope (Python) and brace-matching (`{}` in C-family languages) both create opportunities for error. Explicit boundaries eliminate ambiguity for humans, parsers, and AI.

Law 4: Intent Before Optimization

Readable correctness precedes clever compactness.

Rationale: Premature optimization causes more bugs than it prevents. Astra prioritizes understanding over brevity, trusting modern compilers to handle performance.

1.3 Comparative Analysis

Let's examine how these laws change program structure:

Traditional Expression-Based Style

```python
x = x + 1
```

This single line conceals multiple operations:

1. Read current value of `x`
2. Add `1` to that value
3. Store result back to `x`
4. Discard previous value

Astra Intent-Based Style

```astra
Add 1 to x
```

Analysis:

- The action is explicit and singular
- The target of modification is clear
- The magnitude of change is stated
- No temporary variables are implied

Output Result:

(State updated: x = 11, assuming x was 10)

1.4 Mental Models: Expression vs. Intent

Expression-Oriented Execution Model

```
[Read x] → [Compute x+1] → [Reassign x] → [Continue]
```

This requires understanding:

- Variable dereferencing
- Arithmetic evaluation
- Assignment semantics
- Memory management

Astra Intent Execution Model

```
[Command: Add 1 to x] → [State Change] → [Continue]
```

This requires understanding:

- What changed (x)
- How it changed (+1)
- Nothing else

The reduction is intentional.

Astra removes **how** in favor of **what.**

1.5 Professional Applications

In professional software systems, this philosophy aligns with:

1. State Machines

Production systems model behavior as discrete state transitions. Astra's one-action-per-line maps directly to state change notation.

2. Workflow Engines

Business process automation requires readable, auditable logic. Astra programs can be reviewed by non-programmers.

3. Distributed Systems

When operations span network boundaries, explicit actions prevent race conditions and make transaction boundaries clear.

4. AI Code Generation

Language models excel at generating Astra because the syntax matches their training on natural language. The explicit structure reduces hallucination.

1.6 Cognitive Science Foundation

Research in program comprehension shows that developers spend 70% of their time **reading** code, not writing it. The cognitive cost of reading is determined by:

- **Working memory load:** How many concepts must be held simultaneously
- **Semantic distance:** Gap between code syntax and problem domain
- **Ambiguity resolution:** Mental effort to determine what code does

Astra's design minimizes all three:

- One action per line reduces working memory load
- English commands reduce semantic distance
- Explicit structure eliminates ambiguity

1.7 Key Takeaways

1. **Programming is communication, not mathematical notation**
2. **Explicit intent reduces cognitive load for humans and AI**
3. **Visual reasoning should precede execution**
4. **Clarity is a prerequisite for correctness**

This philosophy underpins every feature in the chapters ahead.

1.8 Exercises

Beginner Level

Rewrite the following traditional expressions in Astra intent form:

```python
total = total * 2
count = count - 1
name = name + " Jr."
```

Advanced Level

Draw a cognitive load diagram comparing:

```python
result = (x + y) * z - 3
```

vs. the Astra equivalent with separate commands.

Chapter 2: The Astra Runtime Environment

From Intent to Execution

2.1 Understanding Program Execution

Every programming language makes promises about how code runs. In many languages, these guarantees are:

- Scattered across documentation
- Implied by examples
- Learned only after encountering bugs

Astra makes execution explicit and visual. Programs are not strings of symbols—they are sequences of validated intent instructions interpreted by the **Astra Runtime Environment (ARE).**

2.2 The Four Execution Laws

Law 1: Top-to-Bottom Sequential Execution

Astra programs execute from first line to last, unless control flow explicitly redirects execution.

Implication: No hidden execution order. No hoisting. No implicit parallel execution.

Law 2: Intent Parsing Before Execution

Every line is parsed into a validated intent structure before any state changes occur.

Implication: Syntax errors are caught before runtime. Partial execution is prevented.

Law 3: No Hidden Side Effects

State changes occur only through explicit commands visible in source code.

Implication: Reading the code tells you everything about what changes. No global mutation. No action-at-a-distance.

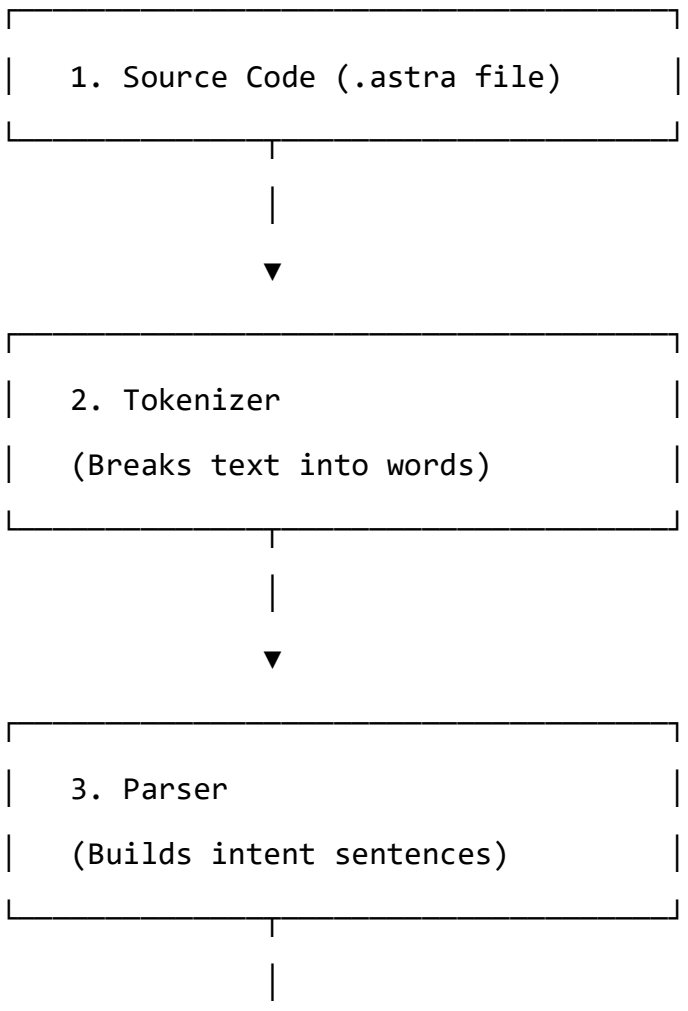
Law 4: Deterministic Semantics

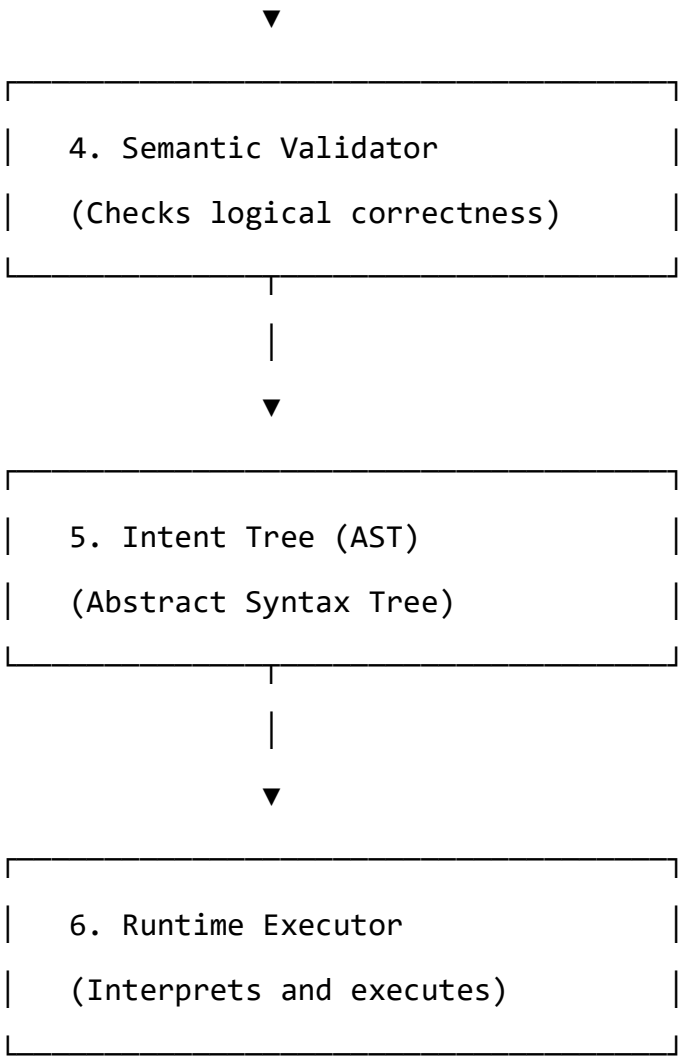
Given identical inputs and initial state, an Astra program produces identical outputs.

Implication: Programs are reproducible, testable, and debuggable.

2.3 The Runtime Pipeline

When you execute an Astra program, it flows through six distinct stages:





Each stage has a **single responsibility** and **well-defined outputs**.

2.4 Stage-by-Stage Walkthrough

Let's trace this program through the pipeline:

```astra
Set temperature to 105
If temperature is greater than 100 then
    Write "Overheating"
End check
```


Stage 1: Source Code

Raw text stored in file.

Stage 2: Tokenization

```
[Set] [temperature] [to] [105]
[If] [temperature] [is greater than] [100] [then]
[Write] ["Overheating"]
[End check]
```

Stage 3: Parsing

```
Command: SetVariable
  - Name: temperature
  - Value: 105

Command: IfStatement
  - Condition: Comparison(temperature > 100)
  - Body:
      Command: Write
        - Text: "Overheating"
```

Stage 4: Semantic Validation

- ✓ Variable `temperature` is defined before use
- ✓ Comparison operator is valid for numeric types
- ✓ Block structure is properly terminated

Stage 5: Intent Tree Construction

```
Program
├── SetVariable(temperature, 105)
└── IfStatement
    ├── Condition: Greater(temperature, 100)
    └── Body
        └── Write("Overheating")
```

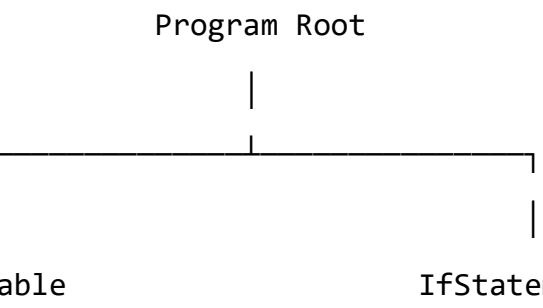
Stage 6: Execution

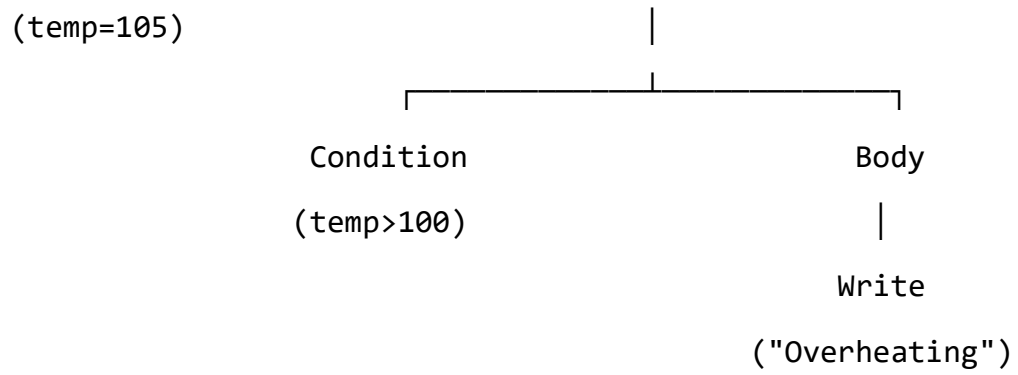
```
Execute: temperature = 105
Evaluate: 105 > 100 → True
Execute: Write("Overheating")
Output: "Overheating"
```

2.5 The Intent Tree (Abstract Syntax Tree)

Rather than executing raw text, Astra builds an **Intent Tree**—a hierarchical representation of program logic.

Visual Representation:





Why This Matters:

- Trees are traversed predictably (depth-first, left-to-right)
- Each node represents one logical operation
- Structure matches visual indentation
- AI can generate and verify tree structure before execution

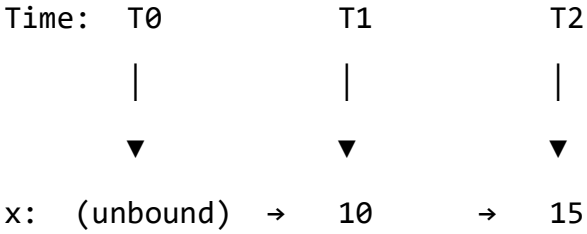
2.6 State, Memory, and Bindings

Astra maintains a clear separation between **names** (identifiers) and **values** (data).

Example Program:

```astra
Set x to 10
Add 5 to x
```

State Evolution Diagram:



Key Insight: The name `x` remains stable. The value bound to `x` changes.

2.7 Validation-Before-Execution Principle

Astra validates **all** operations before changing **any** state.

Invalid Program:

```astra
Add "Hello" to 5
```

Execution Flow:

```
┌──────────────┐
│   Parse      │ ✓ Syntax valid
└──────┬───────┘
       │
       ▼
┌──────────────┐
│  Validate    │ ✗ Type mismatch
└──────┬───────┘   (String + Integer)
       │
       ▼
┌──────────────┐
│    STOP      │ (State unchanged)
└──────────────┘
```


Contrast with languages that fail during execution:

```python
x = 10
y = x + "hello"  # Runtime error AFTER x is set
```

Astra catches this at validation, preventing partial state corruption.

2.8 Error Handling at Runtime

When validation fails, Astra provides:

1. **Line number** where error occurred
2. **Nature of error** (type mismatch, undefined variable, etc.)
3. **Suggested fix** when applicable

Example Error Message:

```
Error on line 3: Cannot add String to Integer
  Add "Hello" to 5
      ^^^^^^^
Suggestion: Convert value to Integer first, or use string concatenation
```

2.9 Professional Insights

Astra's runtime model aligns with established systems:

1. Interpreters

Like Python and Ruby, Astra executes through interpretation, enabling dynamic features while maintaining safety.

2. Workflow Engines

Business process tools (BPMN, Camunda) execute flowcharts. Astra's Intent Trees map directly to workflow diagrams.

3. Rule-Based Systems

Expert systems and business rules engines use condition-action pairs—exactly what Astra If-Then blocks represent.

4. AI Planning Systems

Automated planners generate action sequences. Astra's explicit action syntax makes it an ideal target for AI code generation.

2.10 Key Takeaways

1. **Astra executes validated intent structures, not raw text**
2. **The runtime pipeline ensures syntax and semantic correctness before execution**
3. **Intent Trees provide visual, debuggable program representation**
4. **Validation-before-execution prevents partial state corruption**
5. **Deterministic execution makes programs reproducible and testable**

Understanding this execution model makes every subsequent chapter more intuitive.

2.11 Exercises

Beginner Level

Draw the runtime pipeline for this program and identify which stage would catch the error:

```astra
Set x to 10
Write y
```

Intermediate Level

Convert this program into an Intent Tree diagram:

```astra
Set score to 85
If score is greater than 90 then
    Write "Excellent"
Else
    Write "Good"
End check
```

Advanced Level

Explain why validation-before-execution is critical in:

- Financial transaction systems
- Medical device software
- Distributed database operations

Provide a concrete example where partial execution would cause data corruption.

Chapter 3: Variables, Identifiers & Scope

Naming and Lifetime

3.1 The Mental Model

Variables are one of the most misunderstood concepts in programming. Common misconceptions include:

- "Variables are containers that store values"
- "Assignment copies data into variables"
- "Variables exist forever once created"

The truth is simpler and more precise:

> A variable is a **name** that is **bound** to a **value**. The binding has a **scope** (where it's visible) and a **lifetime** (how long it exists).

This chapter establishes the mental model that prevents subtle bugs throughout your Astra programming career.

3.2 The Four Scope Laws

Law 1: Identifiers Are Contracts

An identifier is a stable label that may be rebound to different values over time, but the label itself doesn't change.

Law 2: Binding Is Explicit

Names are bound to values only through explicit commands (`Set`, `Create`, function parameters). No implicit binding.

Law 3: Scope Is Lexical and Visible

Where a name is valid is determined by block structure, visible in the source code. Not by indentation, not by runtime state.

Law 4: Inner Scopes May Shadow Outer Names

A name defined inside a block temporarily hides an outer name with the same identifier. The outer binding remains intact.

3.3 Valid Identifiers

Astra identifiers must follow these rules:

Requirements:

- Start with a letter (A-Z, a-z)
- Contain only letters, numbers, or underscores
- Be case-sensitive

Valid Examples:

```astra
Set user_name to "Ismail"
Set User_Name to "Different Person"
Set temperature2 to 98.6
Set _internal_counter to 0
```

Invalid Examples:

```astra
Set 1user to "Error"        # Cannot start with number
```



```
Set user-name to "Error"     # Hyphen not allowed
Set user.name to "Error"     # Dot not allowed
Set class to "Error"         # Reserved word
```

Note: `user_name` and `User_Name` are **distinct** identifiers due to case sensitivity.

3.4 Binding and Rebinding

Initial Binding:

```astra
Set score to 10
```

Memory State:

```
score ──────► 10
```

Rebinding:

```astra
Set score to 20
```

New Memory State:

```
score ──────► 20
         (10 is garbage collected)
```

Key Insight: The name `score` remains. The value it points to changes. For immutable types (numbers, strings), the old value is discarded.

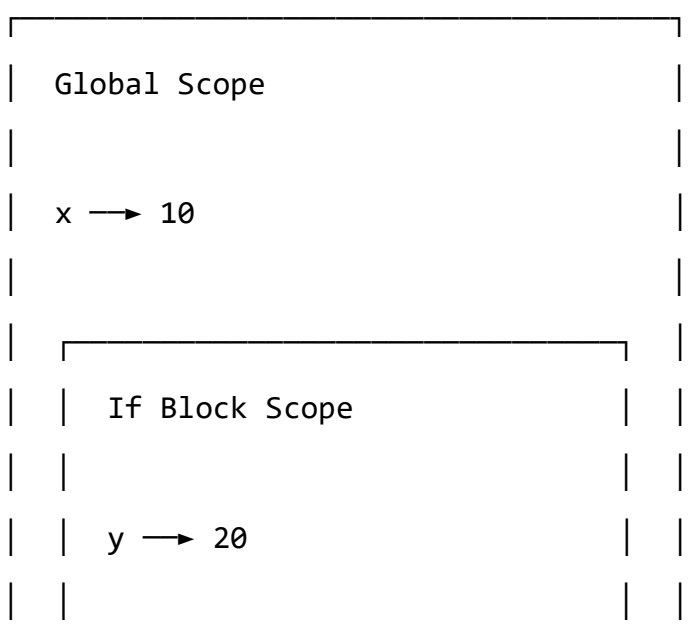
3.5 Scope Visualization

Scope determines **where** a name is visible and valid.

Example Program:

```astra
Set x to 10                    # Global scope

If x is greater than 5 then
    Set y to 20                # Inner scope
    Write x                    # Can see outer x
    Write y                    # Can see inner y
End check

Write x                        # Can see outer x
Write y                        # ERROR: y doesn't exist here
```

Scope Diagram:




```
│  │  (x is visible here too)     │  │
│  └──────────────────────────────┘  │
│                                    │
│  (y is NOT visible here)           │
└────────────────────────────────────┘
```

3.6 Shadowing: Temporary Name Hiding

When an inner scope defines a name that already exists in an outer scope, the inner binding **temporarily shadows** the outer one.

Example:

```astra
Set x to 10                    # Outer x

If True then
    Set x to 20                # Inner x (shadows outer)
    Write x                    # Outputs: 20
End check

Write x                        # Outputs: 10 (outer restored)
```

Lifetime Diagram:

```
Time: ──────────────────────────────────►
Outer x: [=10]═══════════[=10]═══════►
               │         │
Inner x:       [=20]═════┘
               (created) (destroyed)
```


Critical Understanding:

- The outer `x` is **hidden**, not destroyed
- The inner `x` is **destroyed** when block ends
- The outer `x` becomes visible again

3.7 Lifetime of Bindings

Global Scope: Exists for entire program execution

```astra
Set program_name to "Astra"   # Lives until program ends
```

Block Scope: Created when block starts, destroyed when block ends

```astra
If temperature is greater than 100 then
    Set alert to "Warning"     # Created here
    Write alert
End check                      # Destroyed here
# alert no longer exists
```

Timeline Visualization:

```
Program Start
    │
    ├── Create global variables
    │
    ├── Enter block
    │   ├── Create block variables
    │   ├── Execute block body
    │   └── Destroy block variables ◄── Important!
```



```
    │
    └── Continue with global variables
    │
Program End
```

3.8 Common Mistakes Astra Prevents

Problem in Other Languages	**How Astra Prevents It**
Accidental global mutation	Explicit scope blocks with boundaries
Variable leakage from loops	Automatic destruction at block end
Confusion about shadowing	Clear visual block structure
Using undefined variables	Validation catches before execution
Typos creating new variables	Strict identifier rules

3.9 Function Scope (Preview)

Functions create their own scope:

```astra
Set x to 10                    # Global

To do MyFunction
    Set x to 20                # Local to function
    Write x                    # Outputs: 20
End function

Run MyFunction
Write x                        # Outputs: 10
```


Scope Hierarchy:

```
Global Scope
    │
    └── Function Scope (separate)
```

3.10 Professional Insights

Clear naming and scope rules provide:

1. Reduced Side Effects

Explicit scope prevents accidental mutation of outer state, a major source of bugs in large systems.

2. Safe Refactoring

When scope is explicit, extracting code into functions doesn't break variable references.

3. Concurrency Safety

Well-defined scope makes it easier to reason about which state might be shared in concurrent programs.

4. AI Generation Accuracy

Language models generate correct Astra code more reliably because scope rules are explicit and consistent.

3.11 Key Takeaways

1. **Variables are names bound to values, not containers**
2. **Binding is always explicit—no implicit variable creation**
3. **Scope is determined by block structure, visible in source code**

4. **Shadowing is temporary and safe—outer bindings survive**

5. **Lifetime is tied to scope—block-scoped variables are destroyed at block end**

3.12 Exercises

Beginner Level

Draw a diagram showing the difference between a name and a value.

Use the example:

```astra
Set age to 25
Set years_old to age
```

Intermediate Level

Predict the output of this program:

```astra
Set x to 10
If x is greater than 5 then
    Set x to 20
    Set y to 30
    Write x
    Write y
End check
Write x
Write y
```


Chapter 4: Data Types & the Object Model

What Values Really Are

4.1 Everything Is an Object

To write correct programs, you must understand three fundamental truths about values:

1. **What kinds of values exist** (types)
2. **Whether values can change** (mutability)
3. **How values are shared or copied** (reference semantics)

Astra adopts a clear, consistent object model that mirrors real execution behavior while remaining readable.

Core Principle:

> In Astra, every value is an object with a type and identity.

Primitive objects are immutable.

Composite objects are mutable.

Names bind to object references, not copies.

4.2 The Four Object Laws

Law 1: Everything Is an Object

Numbers, strings, booleans, lists, dictionaries—all values are objects with type and identity.

Law 2: Primitive Objects Are Immutable

Numbers, strings, and booleans cannot be modified after creation. Operations create new objects.

Law 3: Composite Objects Are Mutable

Collections (lists, dictionaries, sets) can be modified through explicit commands.

Law 4: Names Bind to Objects, Not Copies

Assignment creates a new reference to the same object, not a duplicate.

4.3 Primitive Types

Primitive objects represent single, indivisible values.

Integer

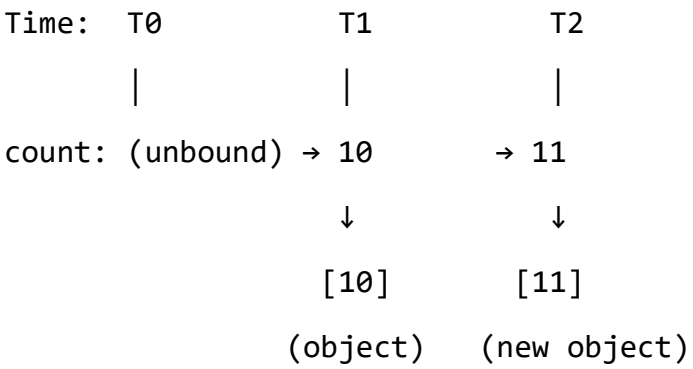
- Arbitrary precision (no overflow)
- Safe for counting, finance, and scientific computing

```astra
Set count to 10
Add 1 to count
Write count
```

Output:

`11`

Memory Diagram:



Key Insight: Adding 1 creates a **new** integer object. The value 10 is unchanged (immutable).

Float

- Decimal numbers with IEEE 754 precision
- Used for measurements, scientific calculations

```astra
Set temperature to 36.6
Add 1.4 to temperature
Write temperature
```

Output: `38.0`

Professional Note: Be aware of floating-point precision limits. For financial calculations, use Decimal type (covered in advanced sections).

String

- Immutable text sequences
- Concatenation creates new strings


```astra
Set title to "Astra"
Add " Programming" to title
Write title
```

Output: `Astra Programming`

Memory Diagram:

```
title: "Astra" ──X── (original unchanged)
         │
         └──────► "Astra Programming" (new object)
```

Boolean

- Logical truth values: `True` or `False`
- Used in conditionals and logic

```astra
Set is_ready to True
If is_ready is True then
    Write "System Ready"
End check
```

Output: `System Ready`

Nothing

- Represents intentional absence of value
- Neither zero, False, nor empty string


```astra
Set result to Nothing
If result is Nothing then
    Write "No result available"
End check
```

Output: `No result available`

Use Cases:
- Uninitialized optional values
- Missing data in datasets
- Function returns when no value is meaningful

4.4 Composite Types

Composite objects group multiple values and **can change over time.**

Lists: Ordered, Mutable Collections

```astra
Create list named items
Add "Apple" to items
Add "Banana" to items
Write items
```

Output: `["Apple", "Banana"]`

Memory Diagram:

```
items ──────► [Box 0: "Apple"]
```



```
              [Box 1: "Banana"]
              (mutable object)
```

4.5 Reference Sharing (Critical Concept)

This is where many bugs originate. Understanding reference semantics prevents countless errors.

Example:

```astra
Create list named items
Add "A" to items
Add "B" to items

Set copy to items          # Both names reference SAME object
Add "C" to copy

Write items
Write copy
```

Output:

```
["A", "B", "C"]
["A", "B", "C"]
```

Memory Diagram:

```
items ──────┐
            ├──────→ ["A", "B", "C"]
```



```
copy ──────┘        (single object)
```

Key Insight: Assignment does **not** copy composite objects. Both names point to the same object in memory.

4.6 Mutability vs. Rebinding

Primitive Rebinding (Creates New Objects):

```astra
Set x to 10
Set y to x
Add 5 to y
Write x
Write y
```

Output:

```
10
15
```

Memory Diagram:

```
x ──────► [10]
y ──────► [15]  (new object created)
```

Explanation: Since integers are immutable, `Add 5 to y` creates a new integer (15) and binds `y` to it. `x` remains bound to the original object (10).

Composite Mutation (Modifies Existing Object):

```astra
Create list named x
Add 10 to x

Set y to x
Add 20 to y

Write x
Write y
```

Output:

```
[10, 20]
[10, 20]
```

Memory Diagram:

```
x ─────┐
       ├─────► [10, 20]  (single object modified)
y ─────┘
```

Explanation: Both `x` and `y` reference the same list object. Adding to `y` mutates the shared object, so both names reflect the change.

4.7 Type System Summary

Type	Category	Mutable?	Use Cases
Integer	Primitive	No	Counting, indexing, IDs
Float	Primitive	No	Measurements, calculations
String	Primitive	No	Text, messages, labels
Boolean	Primitive	No	Flags, conditionals
Nothing	Primitive	N/A	Missing data, optional values
List	Composite	Yes	Ordered collection, sequences
Dictionary	Composite	Yes	Key-value mappings, records
Set	Composite	Yes	Unique collection, membership
Tuple.	Composite	No	Immutable record, coordinates

4.8 Common Misconceptions

Misconception	Reality
"Variables store values"	Names bind to object references
"Assignment copies data"	Shares references for composites
"All values are mutable"	Primitives immutable,composites mutable
"Strings can be changed"	Strings immutable;operations create new
"*Nothing* equals zero"	*Nothing* is distinct from other values

---

4.9 Professional Applications

Understanding Astra's object model prevents bugs in:

1. Data Science Pipelines

When filtering datasets, knowing whether operations mutate or create new objects prevents unexpected data loss.

2. Concurrent Systems

Immutable primitives are thread-safe by nature. Mutable composites require synchronization.

3. AI Model Training

Parameter updates must mutate shared weight tensors. Understanding reference semantics ensures gradients update the correct objects.

4. Memory Management

Knowing when objects are copied vs. referenced helps predict memory usage in large-scale applications.

4.10 Key Takeaways

1. **Every value in Astra is an object with type and identity**
2. **Primitive objects (numbers, strings, booleans) are immutable**
3. **Composite objects (lists, dictionaries, sets) are mutable**
4. **Assignment creates references, not copies (for composites)**
5. **Operations on primitives create new objects; operations on composites modify existing objects**

This object model matches Python's execution reality while making behavior explicit and readable.

4.11 Exercises

Beginner Level

Draw memory diagrams for each step:

```astra
```



```
Set a to 5
Set b to a
Add 3 to b
```

Intermediate Level

Explain why this program produces its output:

```astra
Create list named x
Add 1 to x
Set y to x
Add 2 to y
Write x
```

Advanced Level

Design a scenario where misunderstanding mutability vs. rebinding causes a bug in a data processing pipeline.

Include:
- The buggy code
- Expected vs. actual output
- Memory diagrams showing what went wrong
- The corrected version

PART II: COMPUTATION & LOGIC

Chapter 5: Operations & State Mutation

Explicit Change Over Hidden Computation

5.1 Why Astra Forbids Inline Expressions

Most programming languages treat computation as expressions:

```python
x = x + 5 * y
```

This single line compresses multiple operations:

1. Read current value of `x`
2. Read current value of `y`
3. Multiply `y` by 5
4. Add result to `x`
5. Store result back into `x`

For humans—especially beginners—this creates cognitive overload.

Astra replaces expressions with **explicit state mutation commands**:

```astra
Multiply y by 5
Add y to x
```

Each line answers exactly one question: **What changed?**

5.2 The Mutation Philosophy

In Astra, operations are **commands**, not **symbols**.

Traditional operators:

```
+  -  *  /  ==  !=  <  >  &&  ||
```

Astra equivalents:

```
Add, Subtract, Multiply, Divide
is, is not, is greater than, is less than
and, or
```

Why this matters:

- Words have consistent meaning across contexts
- Symbols require memorization and create ambiguity
- Commands make program flow traceable in debuggers
- Natural language is easier for AI to generate and verify

5.3 Arithmetic Operations

Addition: Add

The `Add` command behaves differently based on target type:

Adding to Numbers:


```astra
Set score to 10
Add 5 to score
Write score
```

Output: `15`

Adding to Strings (Concatenation):

```astra
Set greeting to "Hello"
Add " World" to greeting
Write greeting
```

Output: `Hello World`

Adding to Lists (Append):

```astra
Create list named numbers
Add 10 to numbers
Add 20 to numbers
Write numbers
```

Output: `[10, 20]`

Type-Aware Behavior:

Astra chooses the appropriate operation based on the target's type. This is **intentional polymorphism**—the same command word has different but related meanings.

Subtraction: Subtract

```astra
Set balance to 1000
Subtract 250 from balance
Write balance
```

Output: `750`

Execution Trace:

```
Start:  balance = 1000
After:  balance = 750
```

Note: Subtraction only applies to numbers. Attempting to subtract from strings or lists produces a validation error.

Multiplication: Multiply

```astra
Set quantity to 5
Multiply quantity by 3
Write quantity
```

Output: `15`

String Repetition:

```astra
Set pattern to "="
```



```
Multiply pattern by 10
Write pattern
```

Output: `==========`

Division: Divide

```astra
Set total to 100
Divide total by 4
Write total
```

Output: `25.0`

Critical Rule: Division **always** produces a Float, even when dividing integers evenly. This prevents silent data loss.

```astra
Set x to 10
Divide x by 2
Write x
```

Output: `5.0` (not `5`)

Professional Note: For integer division, use `Divide evenly` (covered in advanced operations).

5.4 Complete Arithmetic Example

```astra
Set balance to 1000
Subtract 250 from balance    # 1000 - 250 = 750
Multiply balance by 2        # 750 * 2 = 1500
Divide balance by 4          # 1500 / 4 = 375.0
Write balance
```

Execution Trace:

```
Start:            balance = 1000
After Subtract:   balance = 750
After Multiply:   balance = 1500
After Divide:     balance = 375.0
```

Output: `375.0`

5.5 Comparison Operations

Astra comparisons read like English sentences.

Equality: is / is not

```astra
Set age to 25
If age is 25 then
    Write "Correct age"
End check
```


Inequality:

```astra
If age is not 30 then
    Write "Not 30"
End check
```

Magnitude Comparisons

```astra
If temperature is greater than 100 then
    Write "Overheating"
End check

If temperature is less than 0 then
    Write "Freezing"
End check

If age is greater than or equal to 18 then
    Write "Adult"
End check

If score is less than or equal to 50 then
    Write "Failed"
End check
```

Complete Set:

- `is` → equality
- `is not` → inequality

- `is greater than` → >
- `is less than` → <
- `is greater than or equal to` → ≥
- `is less than or equal to` → ≤

Why no symbols? Symbols like `>`, `<`, `==`, `!=` are:
- Easy to misread (`=` vs `==`)
- Ambiguous in some contexts
- Harder to verbalize in code reviews
- More difficult for non-programmers to understand

5.6 Logical Connectors

AND: Combining Conditions

```astra
If age is greater than 18 and status is "Active" then
    Write "Access Granted"
End check
```

Truth Table:

```
A     B     A and B
True  True  True
True  False False
False True  False
False False False
```


Short-Circuit Evaluation:

If the first condition is `False`, the second is never evaluated.

```astra
If x is not Nothing and x is greater than 10 then
    Write "Valid and large"
End check
```

This prevents errors if `x` is `Nothing`.

OR: Alternative Conditions

```astra
If role is "Admin" or role is "Moderator" then
    Write "Has elevated privileges"
End check
```

Truth Table:

```
A     B     A or B
True  True  True
True  False True
False True  True
False False False
```

Short-Circuit Evaluation:

If the first condition is `True`, the second is never evaluated.

NOT: Negation

```astra
Set is_valid to True
If not is_valid then
    Write "Invalid"
Else
    Write "Valid"
End check
```

Output: `Valid`

5.7 Complex Logical Expressions

```astra
If age is greater than 18 and (has_license is True or has_permit is True) then
    Write "Can drive"
End check
```

Evaluation Order:

1. Parentheses first
2. `and` before `or` (if no parentheses)
3. Left to right

Best Practice: Use parentheses for clarity even when not strictly required.

5.8 Common Errors Astra Prevents

Problem in Other Languages	**How Astra Prevents**
`=` vs `==` confusion	No `=` in conditions; use `is`
Order of operations ambiguity	One operation per line
Type coercion surprises	Explicit type-aware operations
Precedence errors	No complex expressions
Division by zero at runtime	Validation before execution

5.9 Type-Safe Operations

Astra validates operations against types:

Valid:

```astra
Set x to 10
Add 5 to x              # Integer + Integer ✓
```

Invalid:

```astra
Set x to 10
Add "Hello" to x        # Integer + String ✗
```

Error Message:

```
Error: Cannot add String to Integer
Line 2: Add "Hello" to x
Suggestion: Convert x to String first, or change operation
```


5.10 Professional Insights

Astra's explicit mutation model aligns with:

1. State Machines

Industrial control systems model state transitions explicitly. Astra's one-change-per-line maps directly to state diagrams.

2. Audit Trails

Financial systems require traceable operations. Each Astra line is one auditable action.

3. Distributed Systems

When operations span network boundaries, explicit mutations make transaction boundaries clear.

4. AI Code Generation

Language models generate Astra more reliably because operations have consistent, unambiguous syntax.

5.11 Key Takeaways

1. **Operations are commands, not symbols**
2. **One operation per line reduces cognitive load**
3. **Type-aware operations prevent entire classes of errors**
4. **Comparisons read like natural language**
5. **Logical connectors use short-circuit evaluation**
6. **Explicit operations make programs easier to debug, explain, and generate**

5.12 Exercises

Beginner Level

Rewrite these traditional expressions as Astra commands:

```python
total = total + price * quantity
discount = total * 0.1
final = total - discount
```

Intermediate Level

Predict the output and explain the evaluation order:

```astra
Set x to 10
Set y to 5
If x is greater than 8 and y is less than 10 or x is 5 then
    Write "Condition met"
Else
    Write "Condition not met"
End check
```

Advanced Level

Design a mutation sequence for a shopping cart system that:

- Adds items with quantities
- Applies discounts
- Calculates tax
- Shows running total after each operation

Include proper type handling and validation logic.

Chapter 6: Control Structures

How Programs Make Decisions

6.1 Programs as Decision Machines

A running program is fundamentally a **decision machine**: at each step, it decides what to do next based on:

- Current state
- Input data
- Conditional logic

Most languages hide decision structure behind symbols and indentation rules. **Astra makes decisions explicit and readable.**

```astra
If temperature is greater than 100 then
    Write "Overheating"
Else
    Write "Normal"
End check
```

This reads exactly like the logic it represents—no hidden meanings, no symbol memorization.

6.2 The Sandwich Architecture

Every Astra control structure follows the same pattern:

```
[START KEYWORD]
    Body
```



```
[END KEYWORD]
```

Why explicit boundaries matter:

1. **No invisible scope** (unlike Python's indentation-based blocks)
2. **No ambiguous endings** (unlike missing braces in C-family languages)
3. **Parsers, humans, and AI all agree** where logic begins and ends

Visual Block Model:

```
If condition then
    │
    │  Execute statements
    │  (this is the body)
    │
End check
```

The block is closed **intentionally**, not accidentally.

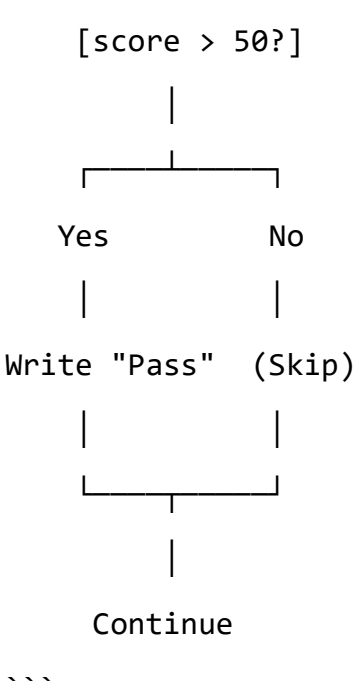
6.3 Conditional Logic: If Statements

Single-Path Decision

Execute code only when condition is true:

```astra
If score is greater than 50 then
    Write "Pass"
End check
```


Decision Flow:

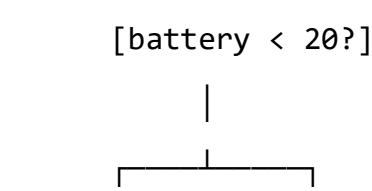


Two-Path Decision (Else)

Execute one path or the other:

```astra
If battery is less than 20 then
    Write "Low Battery"
Else
    Write "Battery OK"
End check
```

Decision Flow:




```
     Yes         No
      |          |
  "Low Battery" "Battery OK"
      |          |
      └─────┬────┘
            |
       Continue
```

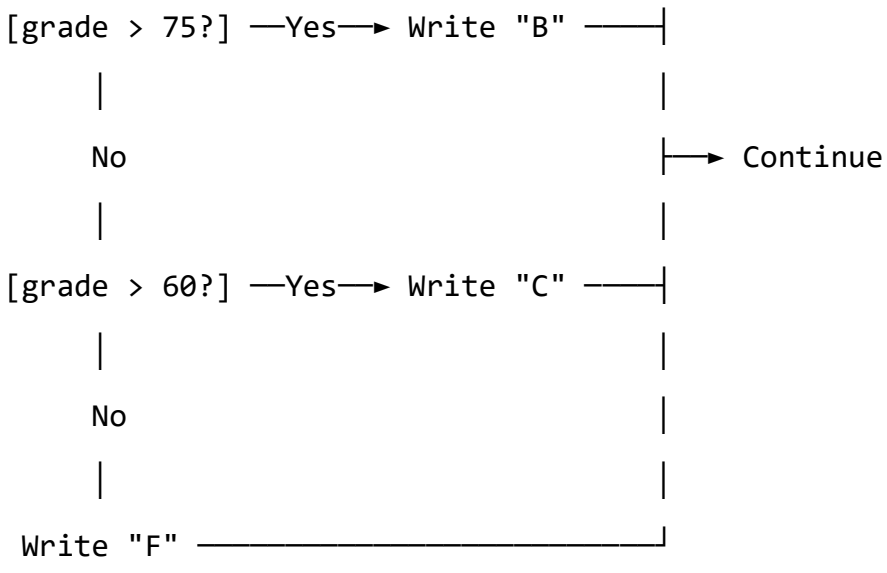
Multi-Path Decision (Else if)

Choose from multiple exclusive options:

```astra
If grade is greater than 90 then
    Write "A"
Else if grade is greater than 75 then
    Write "B"
Else if grade is greater than 60 then
    Write "C"
Else
    Write "F"
End check
```

Decision Chain:

```
[grade > 90?] ──Yes──► Write "A" ────┐
     |                               |
     No                              |
     |                               |
```

Key Properties:

- Conditions evaluated **top to bottom**
- First `True` condition executes, then exits
- `Else` catches all remaining cases
- Exactly **one** path executes

6.4 Iteration: Repeating Actions

Astra provides three distinct loop models, each optimized for different mental models:

1. **Fixed Repetition** → "Do this N times"
2. **Conditional Repetition** → "Keep doing this while X is true"
3. **Collection Traversal** → "Do this for every item"

6.4.1 Fixed Repetition (Repeat N Times)

Use when the count is known in advance:


```astra
Repeat 3 times
    Write "Hello"
End repeat
```

Execution Timeline:

```
Iteration 1 → Hello
Iteration 2 → Hello
Iteration 3 → Hello
```

No counters. No off-by-one errors. Just simple repetition.

With Counter Access:

```astra
Repeat 5 times using counter as i
    Write i
End repeat
```

Output:

```
0
1
2
3
4
```

Note: Counter starts at 0 (standard in computing for zero-based indexing).

6.4.2 Conditional Repetition (Loop while)

Use when repetition depends on changing state:

```astra
Set attempts to 0
Loop while attempts is less than 3
    Write "Trying..."
    Add 1 to attempts
End loop
```

State Evolution:

```
Start: attempts = 0

[attempts < 3?] Yes → Execute body → attempts = 1
[attempts < 3?] Yes → Execute body → attempts = 2
[attempts < 3?] Yes → Execute body → attempts = 3
[attempts < 3?] No  → Exit loop
```

Output:

```
Trying...
Trying...
Trying...
```

Critical Responsibility: The programmer **must** modify the condition variable inside the loop, or it becomes infinite.

Infinite Loop (Bug):

```astra
Set x to 0
Loop while x is less than 10
    Write "Forever"
    # BUG: x never changes!
End loop
```

6.4.3 Collection Traversal (Iterator Loop)

Use to process every item in a collection safely:

```astra
Create list named scores
Add 85 to scores
Add 92 to scores
Add 78 to scores

Repeat for every score in scores
    Write score
End repeat
```

Output:

```
85
92
78
```


Iterator Model:

```
scores ──► [85] → [92] → [78]
            ↓      ↓      ↓
          score  score  score
          (current item in each iteration)
```

Benefits:

- No manual indexing
- No index-out-of-bounds errors
- No counter management
- Works with any collection type

With Index Access:

```astra
Repeat for every score in scores using index as i
    Write "Score at position"
    Write i
    Write "is"
    Write score
End repeat
```

Output:

```
Score at position 0 is 85
Score at position 1 is 92
Score at position 2 is 78
```


6.5 Flow Interruption

Sometimes you need to exit or skip iterations early.

Stop loop (Break)

Exit the loop immediately:

```astra
Repeat 10 times using counter as count
    If count is 5 then
        Stop loop
    End check
    Write count
End repeat
Write "Done"
```

Output:

```
0
1
2
3
4
Done
```

Flow Diagram:

```
Loop starts
```



```
  │
  ├─ Iteration 0 ✓
  ├─ Iteration 1 ✓
  ├─ Iteration 2 ✓
  ├─ Iteration 3 ✓
  ├─ Iteration 4 ✓
  ├─ Iteration 5 → STOP → Exit loop
  │
Continue after loop
```

Skip iteration (Continue)

Skip to next iteration:

```astra
Create list named numbers
Add 1 to numbers
Add 0 to numbers
Add 5 to numbers
Add 0 to numbers
Add 3 to numbers

Repeat for every n in numbers
    If n is 0 then
        Skip iteration
    End check
    Write n
End repeat
```


Output:

```
1
5
3
```

Flow Diagram:

```
Item: 1 → Process ✓ → Output: 1
Item: 0 → Skip → (no output)
Item: 5 → Process ✓ → Output: 5
Item: 0 → Skip → (no output)
Item: 3 → Process ✓ → Output: 3
```

6.6 Nested Control Structures

Control structures can contain other control structures:

```astra
Create list named matrix
Create list named row1
Add 1 to row1
Add 2 to row1
Add row1 to matrix

Create list named row2
Add 3 to row2
```



```
Add 4 to row2
Add row2 to matrix

Repeat for every row in matrix
    Repeat for every value in row
        Write value
    End repeat
End repeat
```

Output:

```
1
2
3
4
```

Nesting Diagram:

```
Outer Loop (rows)
│
├─ Inner Loop (values in row 1)
│  ├─ Process value 1
│  └─ Process value 2
│
└─ Inner Loop (values in row 2)
   ├─ Process value 3
   └─ Process value 4
```


6.7 Common Errors Astra Eliminates

Problem in Other Languages	Astra Solution
Missing braces `{}`	Explicit `End check`, `End loop`
Indentation bugs	Indentation is optional
Infinite loops by accident	Clear condition visibility
Index out of bounds	Iterator loops no manual indexing
Off-by-one errors	`Repeat N times` no manual counter

6.8 Professional Insights

Astra control structures are designed to:

1. Map cleanly to flowcharts

Every Astra control structure has an obvious visual representation.

2. Be generated safely by AI

Explicit boundaries and keywords reduce generation errors.

3. Be readable aloud

Code reviews can be conducted verbally without ambiguity.

4. Scale from classroom to production

The same patterns work in simple scripts and complex systems.

6.9 Key Takeaways

1. **Control structures use explicit start/end keywords (Sandwich Architecture).**

2. **If/Else if/Else provides multi-path decisions.**

3. **Three loop types match three mental models: count, condition, collection.**

4. **Stop loop and Skip iteration provide flow control.**

5. **Nested structures are explicit and unambiguous.**

Control flow is not about syntax cleverness — it's about clear decision-making.

6.10 Exercises

Beginner Level

Rewrite this Python code in Astra:

```python
for i in range(5):
    if i % 2 == 0:
        print(i)
```

Intermediate Level

Write an Astra program that:

- Creates a list of 10 numbers
- Uses a conditional loop to find the first number greater than 50
- Stops searching once found
- Reports the result

Advanced Level

Design a nested loop structure that:

- Processes a 2D grid (list of lists)
- Skips any cell containing 0
- Stops processing entirely if any cell contains -1
- Counts how many valid cells were processed

Chapter 7: Error Handling & Safety

Designing for Failure

7.1 Failure Is Normal, Crashes Are Not

Professional systems operate under a fundamental assumption: **things will go wrong.**

- Files will be missing
- Network connections will timeout
- Users will input invalid data
- Divisions by zero will be attempted
- Memory will be exhausted

The difference between amateur and professional software is not whether errors occur—it's **how they're handled.**

Amateur approach: Hope nothing breaks, crash if it does.

Professional approach: Anticipate failure, recover gracefully, maintain system integrity.

Astra treats failure as **first-class control flow**, not an afterthought.

7.2 The Three Safety Laws

Law 1: Explicit Failure Handling

Risky operations must be wrapped in safety blocks. The programmer explicitly acknowledges risk.

Law 2: Guaranteed Cleanup

Resources are always released, even when errors occur.

Law 3: No Silent Failures

Errors must be either handled or explicitly propagated. Nothing fails silently.

7.3 The Attempt-Recover-Always Model

Astra replaces symbolic `try/catch/finally` with readable safety corridors:

```astra
Attempt
    Divide 10 by 0
Recover
    Write "Division by zero occurred"
Always
    Write "Cleanup complete"
End attempt
```

Output:

```
Division by zero occurred
Cleanup complete
```

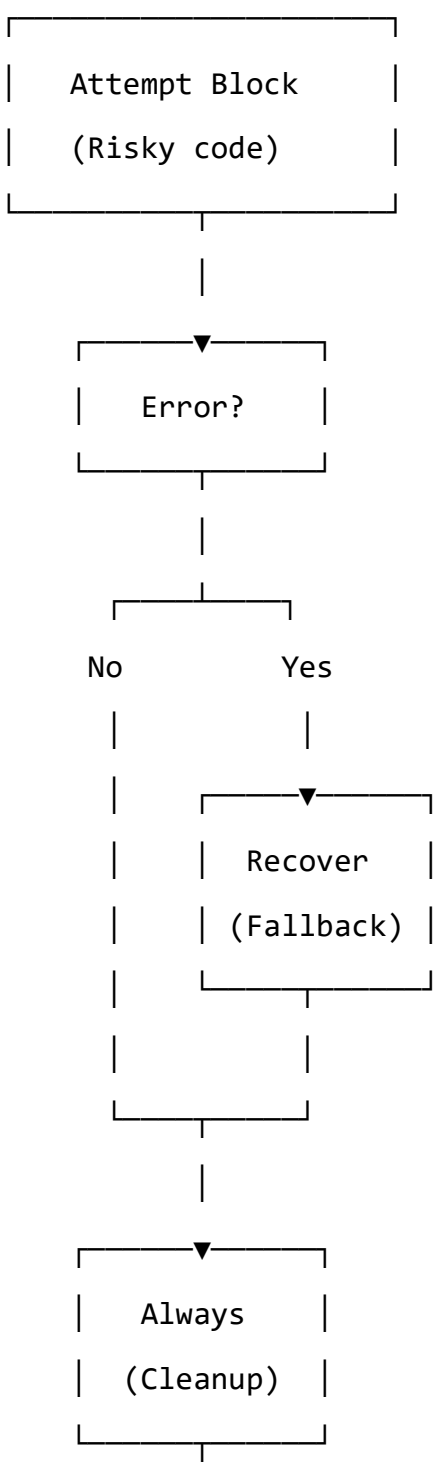
Three Sections:

1. **Attempt** → Code that might fail
2. **Recover** → What to do if failure occurs

3. **Always** → Guaranteed execution (cleanup)

7.4 Failure Flow Visualization

Core Concept Diagram:




```
           |
      Continue ──►
```

Key Insight: The `Always` block executes regardless of success or failure.

7.5 Timeline: Successful Execution

When no error occurs:

```
Time ──────────────────────────────────────►
Attempt ──────────✔──────────────────►
  |
  └─ Code executes successfully

Recover (SKIPPED)
Always ──────────✔──────────────────►
  |
  └─ Cleanup runs
Continue ──────────────────────────────────►
```

Example:

```astra
Attempt
    Set x to 10
    Divide x by 2
    Write "Result is 5"
Recover
    Write "Error occurred"
```



```
Always
    Write "Done"
End attempt
```

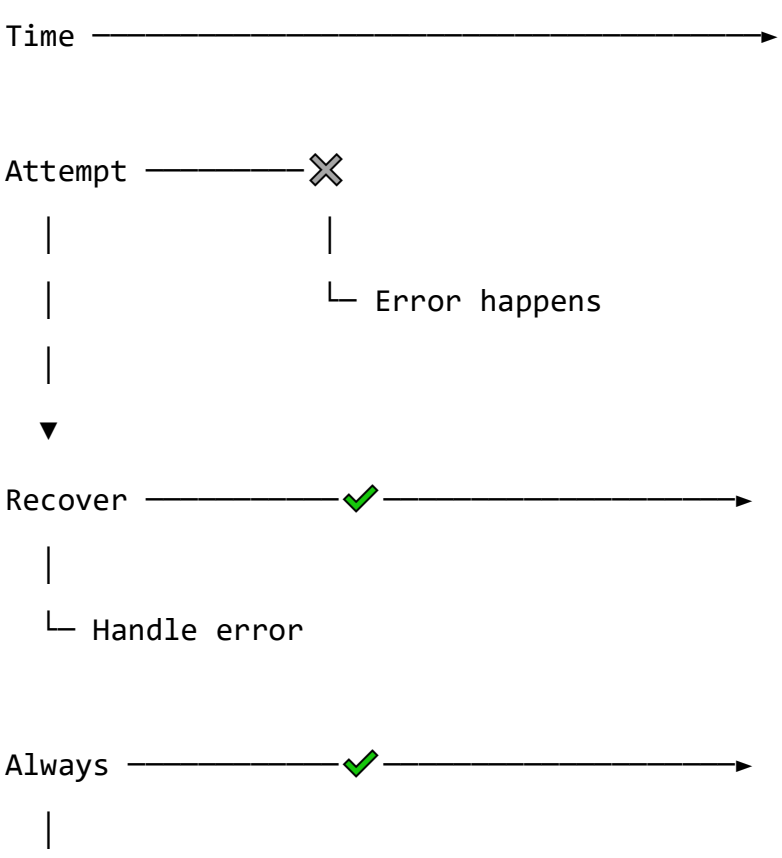
Output:

```
Result is 5
Done
```

The `Recover` block was skipped entirely.

7.6 Timeline: Failure Execution

When error occurs:




```
  └─ Cleanup runs


Continue ──────────────────────────────────►
```

Example:

```astra
Attempt
    Divide 10 by 0
    Write "This never runs"
Recover
    Write "Cannot divide by zero"
Always
    Write "Done"
End attempt
```

Output:

```
Cannot divide by zero
Done
```

The `Attempt` block stopped immediately at the error. `Always` still executed.

7.7 The Error Message Channel

Inside the `Recover` block, Astra provides a special variable: `error_message`.

```astra
```



```
Attempt
    Read file "missing.txt" save to data
Recover
    Write "Error: "
    Write error_message
Always
    Write "Attempted file operation"
End attempt
```

Output:

```
Error: File not found: missing.txt
Attempted file operation
```

Data Flow:

```
Runtime Error
     │
     ▼
error_message ──────► "File not found: missing.txt"
     │
     ▼
Available in Recover block only
```

Professional Note: The `error_message` variable exists **only** within the `Recover` block. It is not visible outside.

7.8 Optional Sections

Attempt-Recover (No Always)

If no cleanup is needed:

```astra
Attempt
    Divide 10 by 0
Recover
    Write "Error handled"
End attempt
```

Attempt-Always (No Recover)

If you want guaranteed cleanup but don't need error handling:

```astra
Attempt
    Write "Processing"
Always
    Write "Cleanup"
End attempt
```

Use case: Logging, timing, or resource tracking regardless of success.

7.9 Nested Safety Blocks

Safety blocks can be nested for layered error handling:

```astra
Attempt
    Write "Outer attempt"

    Attempt
        Divide 5 by 0
    Recover
        Write "Inner error caught"
    End attempt

    Write "Outer continues"
Recover
    Write "Outer error caught"
Always
    Write "Outer cleanup"
End attempt
```

Output:

```
Outer attempt
Inner error caught
Outer continues
Outer cleanup
```


Safety Stack Visualization:

```
┌──────────────────────────────────┐
│  Outer Attempt                   │
│  ┌──────────────────────────┐    │
│  │  Inner Attempt           │    │
│  │  Error! ✗                │    │
│  │  ┌──────────────────┐    │    │
│  │  │  Inner Recover ✓ │    │    │
│  │  └──────────────────┘    │    │
│  │  (Error contained)       │    │
│  └──────────────────────────┘    │
│  Continue normally               │
│  ┌──────────────────────────┐    │
│  │  Outer Always ✓          │    │
│  └──────────────────────────┘    │
└──────────────────────────────────┘
```

Key Principle: Inner errors can be caught without affecting outer blocks.

7.10 Practical Error Scenarios

File Operations

```astra
Attempt
    Read file "config.txt" save to config
    Write "Configuration loaded"
Recover
```



```
    Write "Using default configuration"
    Set config to "default"
Always
    Write "Configuration stage complete"
End attempt
```

Network Operations

```astra
Attempt
    Call python requests.get with inputs "https://api.example.com" save to response
    Write "Data retrieved"
Recover
    Write "Network error: "
    Write error_message
    Set response to Nothing
Always
    Write "Request attempt complete"
End attempt
```

User Input Validation

```astra
Attempt
    Get input "Enter age: " save to age_text
    Convert age_text to integer save to age

    If age is less than 0 or age is greater than 150 then
```



```
        Raise error "Invalid age range"
    End check
Recover
    Write "Invalid input: "
    Write error_message
    Set age to Nothing
End attempt
```

7.11 Raising Custom Errors

You can explicitly raise errors when detecting invalid state:

```astra
To do ValidatePassword with inputs password
    If length of password is less than 8 then
        Raise error "Password must be at least 8 characters"
    End check

    Return True
End function

Attempt
    Run ValidatePassword with inputs "abc"

Recover
    Write error_message
End attempt
```


Output:

```
Password must be at least 8 characters
```

7.12 Why Astra Avoids Silent Failures

Anti-Pattern (Other Languages):

```python
def get_user(id):
    try:
        return database.query(id)
    except:
        return None  # Silent failure!
```

Problem: Caller can't distinguish between "user not found" and "database error".

Astra Pattern:

```astra
To do GetUser with inputs user_id
    Attempt
        Call python database.query with inputs user_id save to user
        Return user
    Recover
        Write "Database error: "
        Write error_message
```



```
        Raise error "Database unavailable"
    End attempt
End function
```

Benefit: Errors propagate with context, forcing explicit handling at appropriate levels.

7.13 Error Propagation

If you don't handle an error, it propagates to outer scopes:

```astra
To do InnerFunction
    Divide 10 by 0
End function

To do OuterFunction
    Attempt
        Run InnerFunction
    Recover
        Write "Caught error from inner function"
    End attempt
End function

Run OuterFunction
```

Output:

```
Caught error from inner function
```



```
```

Propagation Flow:

```
InnerFunction
    │
    ├─ Error occurs ✗
    │
    └─ No Recover block
         │
         └─ Propagate upward
              │
              ▼
OuterFunction Recover ✓
```

7.14 Best Practices

1. Handle Specific Errors

```astra
Attempt
    Read file "data.txt" save to data
Recover
    If error_message contains "not found" then
        Write "Creating new file"
    Else if error_message contains "permission" then
        Write "Access denied"
    Else
```



```
        Write "Unknown file error"
    End check
End attempt
```

2. Always Provide Context

```astra
Attempt
    Divide total by count
Recover
    Write "Division failed with total="
    Write total
    Write " and count="
    Write count
End attempt
```

3. Use Always for Resources

```astra
Set file_handle to Nothing
Attempt
    Open file "data.txt" save to file_handle
    # ... process file
Always
    If file_handle is not Nothing then
        Close file_handle
    End check
End attempt
```


7.15 Common Error Patterns

Scenario	Error Type	Handling Strategy
File not found	IOError	Recover with default or create
Division by zero	MathError	Recover with sentinel value
Invalid input	ValueError	Recover with retry prompt
Network timeout	NetworkError	Recover with retry logic
Memory exhausted	MemoryError	Propagate(usually unrecoverable)

7.16 Professional Insights

Astra's safety model mirrors production-grade systems:

1. Transaction Management

Database transactions use similar patterns: attempt, rollback on error, always close connection.

2. Resource Acquisition Is Initialization (RAII)

The `Always` block guarantees cleanup, preventing resource leaks.

3. Circuit Breakers

Microservices use attempt-recover patterns to prevent cascading failures.

4. Defensive Programming

Explicit error handling makes code auditable and verifiable—critical in regulated industries.

7.17 Key Takeaways

1. Failure is first-class control flow, not an exception.
2. Attempt-Recover-Always provides structured safety.
3. The Always block guarantees cleanup regardless success/failure.
4. error_message provides context in Recover blocks.
5. Nested safety blocks enable layered error handling.
6. Errors propagate upward until handled.
7. Explicit error handling prevents silent failures.

7.18 Exercises

Beginner Level

Write an Astra program that:
- Attempts to divide two numbers
- Recovers by printing an error message
- Always prints "Calculation complete"

Advanced Level

Create a function that reads a configuration file with:
- Primary file path
- Fallback file path
- Default values if both fail

Use nested Attempt blocks.

PART III – STRUCTURING PROGRAMS

Chapter 8: Functions as Reusable Intent

Building Blocks of Modularity

8.1 Functions Are Recipes, Not Mathematics

In mathematics, a function is a mapping from inputs to outputs:

f(x) = x²

In programming, **functions are named procedures**—recipes that can be:

- **Defined once** and used many times
- **Called** with different inputs
- **Return** results to the caller
- **Composed** into larger programs

Astra calls them "To do" blocks to emphasize their procedural nature.

8.2 The Four Function Laws

Law 1: Define Before Use

Functions must be defined before they can be called. No forward references.

Law 2: Explicit Parameters

Input requirements are declared in the function signature.

Law 3: Explicit Returns

Functions explicitly return values using the `Return` command.

Law 4: Local Scope

Variables created inside functions are destroyed when the function ends.

8.3 Defining Functions

Basic Syntax:

```astra
To do FunctionName
    # Function body
End function
```

Simple Example:

```astra
To do Greet
    Write "Hello, World!"
End function
```

Nothing happens yet. The function is just **defined**, not executed.

8.4 Calling Functions

Syntax:

```astra
Run FunctionName
```



```

**Example**:

```astra
To do Greet
    Write "Hello, World!"
End function

Run Greet
Run Greet
```

Output:

```
Hello, World!
Hello, World!
```

Lifecycle Diagram:

```
Define ──────► Store Recipe ──────► Run ──────► Execute ──────► Return
  (once)        (in memory)        (many)       (body)        (to caller)
```

8.5 Functions with Parameters

Syntax:

```astra
To do FunctionName with inputs param1, param2
    # Use param1 and param2
```


```

```
End function
```

Example:

```astra
To do GreetPerson with inputs name
    Write "Hello, "
    Write name
    Write "!"
End function

Run GreetPerson with inputs "Alice"
Run GreetPerson with inputs "Bob"
```

Output:

```
Hello, Alice!
Hello, Bob!
```

Parameter Flow Diagram:

```
Caller              Function
  │                    │
  ├──"Alice"──────────►│ name = "Alice"
  │                    │ Execute body
  │◄───────────────────┤ (implicit return)
  │                    │
  ├──"Bob"────────────►│ name = "Bob"
  │                    │ Execute body
  │◄───────────────────┤ (implicit return)
```



```

---

## 8.6 Functions That Return Values

**Syntax:**

```astra
To do FunctionName with inputs param
    # ... computation
    Return result
End function
```

Example:

```astra
To do Square with inputs number
    Multiply number by number
    Return number
End function

Run Square with inputs 5 save to result
Write result
```

Output: `25`

Data Flow:

```
Caller                Function Square
  │                          │
  ├──── 5 ──────────────────►│ number = 5
  │                          │ number = number * number
```


```

```
  │   │ number = 25
  │◄──────── 25 ─────────────┤ Return 25
  │   │
result = 25
```

8.7 Multiple Parameters

```astra
To do Add with inputs a, b
    Add b to a
    Return a
End function

Run Add with inputs 10, 5 save to sum
Write sum
```

Output: `15`

Parameter Binding:

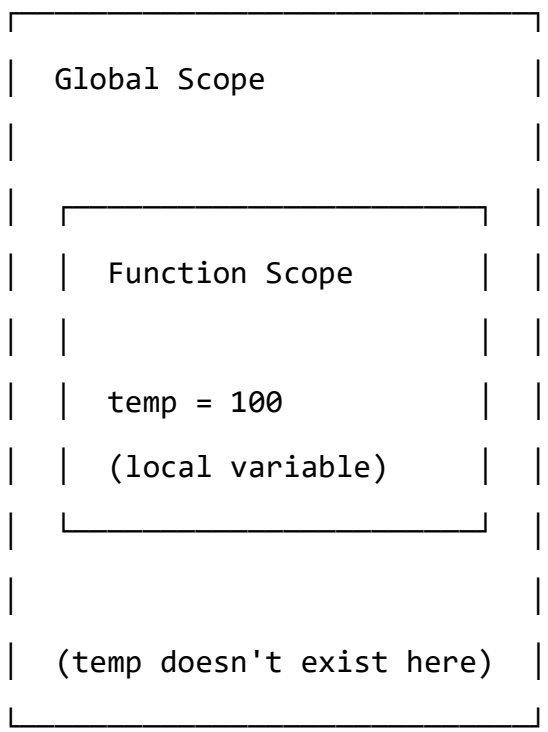
```
Call: Run Add with inputs 10, 5

Binds to:
    a = 10
    b = 5

Returns: 15
```

8.8 Local Scope in Functions

Variables created inside functions are **local**—they don't exist outside:

```astra
To do Calculate
    Set temp to 100
    Write temp
End function

Run Calculate
Write temp              # ERROR: temp doesn't exist here
```

Scope Diagram:



8.9 Global vs. Local Variables

Global variable (accessible everywhere):

```astra
Set global_count to 0

To do Increment
    Add 1 to global_count
End function

Run Increment
Write global_count
```

Output: `1`

Local variable (accessible only in function):

```astra
To do Increment
    Set local_count to 0
    Add 1 to local_count
    Write local_count
End function

Run Increment
Run Increment
```

Output:

```
```



```
1
1
```

Each call creates a **new** local variable.

8.10 Call Stack Visualization

When functions call other functions, Astra maintains a **call stack**:

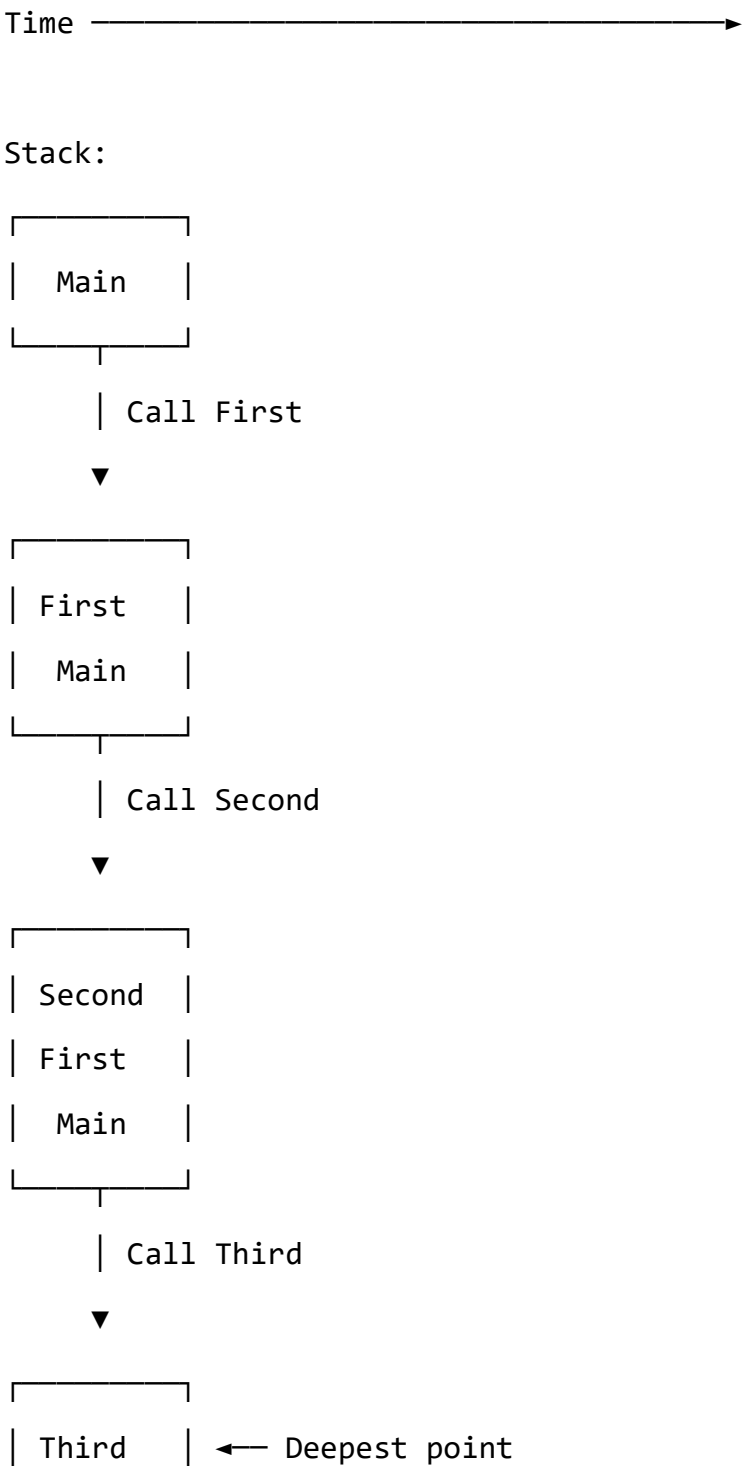
```astra
To do Third
    Write "In Third"
End function

To do Second
    Write "In Second"
    Run Third
End function

To do First
    Write "In First"
    Run Second
End function

Run First
```

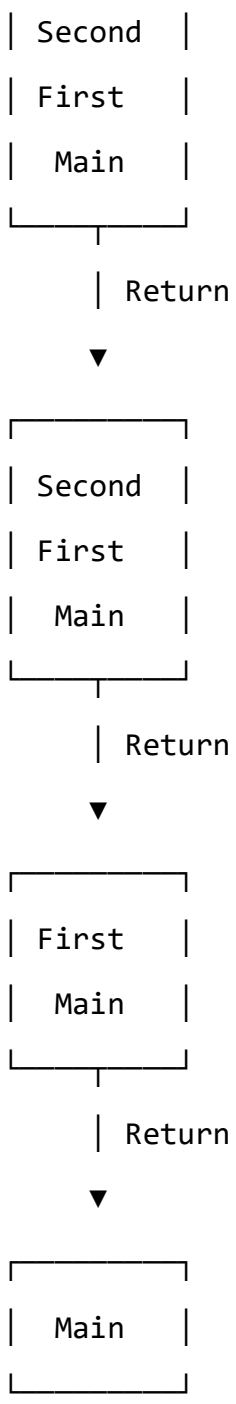
Output:

```

```

```
In First
In Second
In Third
```

Call Stack Timeline:





8.11 Early Return

Functions can return early:

```astra
To do CheckAge with inputs age
```



```
    If age is less than 18 then
        Return "Too young"
    End check
    Return "Valid age"
End function

Run CheckAge with inputs 15 save to result1
Run CheckAge with inputs 25 save to result2

Write result1
Write result2
```

Output:

```
Too young
Valid age
```

Control Flow:

```
CheckAge(15):
    ├─ age < 18? Yes
    └─ Return "Too young" ──► Exit immediately

CheckAge(25):
    ├─ age < 18? No
    └─ Return "Valid age" ──► Exit
```

8.12 Functions Without Return Values

If no `Return` is specified, function returns `Nothing`:

```astra
To do PrintBanner
    Write "============"
    Write "  Welcome"
    Write "============"
End function

Run PrintBanner save to result
Write result
```

Output:

```
============
  Welcome
============
Nothing
```

8.13 Default Parameter Values

Pattern (simulated):

```astra
```



```
To do Greet with inputs name
    If name is Nothing then
        Set name to "Guest"
    End check

    Write "Hello, "
    Write name
End function

Run Greet with inputs "Alice"
Run Greet with inputs Nothing
```

Output:

```
Hello, Alice
Hello, Guest
```

8.14 Practical Function Examples

Validation Function

```astra
To do IsValidEmail with inputs email
    If length of email is less than 5 then
        Return False
    End check

    If email contains "@" then
        Return True
    End check
```



```
    Return False
End function

Run IsValidEmail with inputs "test@example.com" save to valid
Write valid
```

Output: `True`

Data Processing Function

```astra
To do CalculateGrade with inputs score
    If score is greater than 90 then
        Return "A"
    Else if score is greater than 80 then
        Return "B"
    Else if score is greater than 70 then
        Return "C"
    Else
        Return "F"
    End check
End function
Run CalculateGrade with inputs 85 save to grade
Write grade
```

Output: `B`

List Processing Function

```astra
```



```
To do Sum with inputs numbers
    Set total to 0

    Repeat for every num in numbers
        Add num to total
    End repeat

    Return total
End function

Create list named scores
Add 10 to scores
Add 20 to scores
Add 30 to scores

Run Sum with inputs scores save to result
Write result
```

Output: `60`

8.15 Recursive Functions

Functions can call themselves:

```astra
To do Factorial with inputs n
    If n is less than or equal to 1 then
        Return 1
    End check
```



```
    Run Factorial with inputs n - 1 save to result
    Multiply result by n
    Return result
End function

Run Factorial with inputs 5 save to answer
Write answer
```

Output: `120`

Recursion Stack:

```
Factorial(5)
  │
  ├─ 5 * Factorial(4)
  │      │
  │      ├─ 4 * Factorial(3)
  │      │      │
  │      │      ├─ 3 * Factorial(2)
  │      │      │      │
  │      │      │      ├─ 2 * Factorial(1)
  │      │      │      │      │
  │      │      │      │      └─ Returns 1
  │      │      │      └─ Returns 2
  │      │      └─ Returns 6
  │      └─ Returns 24
  └─ Returns 120
```

8.16 Function Documentation Pattern

Best Practice:

```astra
To do CalculateArea with inputs width, height
    # Purpose: Calculate rectangle area
    # Inputs: width (number), height (number)
    # Returns: area (number)

    Multiply width by height
    Return width
End function
```

8.17 Professional Insights

Functions are the foundation of:

1. Code Reuse

Write once, use many times. Reduces duplication and bugs.

2. Abstraction

Hide complex implementation behind simple interfaces.

3. Testing

Isolated functions can be tested independently.

4. Modularity

Break large programs into manageable pieces.

5. Team Collaboration

Different developers can work on different functions.

8.18 Key Takeaways

1. **Functions are named, reusable procedures**
2. **Define before use—no forward references**
3. **Parameters provide inputs; Return provides outputs**
4. **Local scope isolates function variables**
5. **Call stack tracks nested function calls**
6. **Early returns simplify conditional logic**
7. **Functions enable modularity, reuse, and testing**

8.19 Exercises

Beginner Level

Write a function that:
- Takes a temperature in Celsius
- Converts it to Fahrenheit
- Returns the result

Formula: F = C × 9/5 + 32

Intermediate Level

Write a function that:
- Takes a list of numbers
- Returns the largest number
- Handles empty lists appropriately

Chapter 9: Data Structures

Organizing Memory for Efficient Access

9.1 Why Data Structures Matter

Programs are not just logic—they are **organized memory**. The way you structure data determines:

- **Performance**: How fast can you find what you need?
- **Correctness**: Can your structure represent relationships accurately?
- **Clarity**: Does the structure match the problem domain?

Astra provides built-in data structures that cover the vast majority of programming needs, from simple lists to specialized collections.

9.2 The Data Structure Hierarchy

```
Data Structures
│
├─ Sequential (Ordered)
│  ├─ List (mutable)
│  ├─ Tuple (immutable)
│  ├─ Stack (LIFO)
│  └─ Queue (FIFO)
│
├─ Associative (Key-Value)
│  └─ Dictionary
```



```
│
└─ Set-Based (Unique)
   └─ Set
```

9.3 Lists: Ordered, Mutable Sequences

A **List** is a row of boxes, each holding a value. Lists are:

- **Ordered**: Items have positions (indices)
- **Mutable**: Can add, remove, modify items
- **Dynamic**: Grow and shrink as needed

Creating Lists

```astra
Create list named scores
```

Memory State:

```
scores ──────► [ ] (empty list)
```

Adding Elements

```astra
Add 85 to scores
Add 92 to scores
Add 78 to scores
```


Memory Evolution:

```
Step 1: scores ──► [85]
Step 2: scores ──► [85, 92]
Step 3: scores ──► [85, 92, 78]
```

9.4 List Indexing

Lists use **zero-based indexing** (industry standard):

```astra
Create list named fruits
Add "Apple" to fruits
Add "Banana" to fruits
Add "Cherry" to fruits

Get item 0 from fruits save to first
Get item 2 from fruits save to third

Write first
Write third
```

Output:

```
Apple
Cherry
```


Index Diagram:

```
Index:  0        1        2
        |        |        |
        ▼        ▼        ▼
fruits: [Apple] [Banana] [Cherry]
```

9.5 List Operations

Setting Values

```astra
Set item 1 in fruits to "Blueberry"
Write fruits
```

Output: `["Apple", "Blueberry", "Cherry"]`

Getting Length

```astra
Get length of fruits save to count
Write count
```

Output: `3`

Removing Items

```astra
Remove item 0 from fruits
Write fruits
```


Output: `["Blueberry", "Cherry"]`

Checking Membership

```astra
If "Cherry" in fruits then
    Write "Found Cherry"
End check
```

Output: `Found Cherry`

9.6 List Slicing

Extract sublists:

```astra
Create list named numbers
Add 10 to numbers
Add 20 to numbers
Add 30 to numbers
Add 40 to numbers
Add 50 to numbers

Get items 1 to 3 from numbers save to subset
Write subset
```

Output: `[20, 30, 40]`

Slice Diagram:

```
Index:    0   1   2   3   4
numbers: [10][20][30][40][50]
              ▲─────────▲
              |         |
           Start     End (inclusive)

subset:      [20][30][40]
```

9.7 Nested Lists (2D Arrays)

Lists can contain other lists:

```astra
Create list named matrix

Create list named row1
Add 1 to row1
Add 2 to row1
Add 3 to row1
Add row1 to matrix

Create list named row2
Add 4 to row2
Add 5 to row2
Add 6 to row2
Add row2 to matrix
```



```
Get item 0 from matrix save to first_row
Get item 1 from first_row save to value

Write value
```

Output: `2`

Memory Structure:

```
matrix ──────┐
             │
             ├──► row1 ──► [1][2][3]
             │
             └──► row2 ──► [4][5][6]
```

9.8 Dictionaries: Key-Value Storage

A **Dictionary** is a cabinet of labeled drawers. Each **key** maps to a **value.**

Creating Dictionaries

```astra
Create dictionary named user
Set "name" in user to "Alice"
Set "age" in user to 30
Set "city" in user to "New York"
```


Memory Diagram:

```
user ──────► {
    "name"  → "Alice"
    "age"   → 30
    "city"  → "New York"
}
```

9.9 Dictionary Operations

Getting Values

```astra
Get "name" from user save to username
Write username
```

Output: `Alice`

Checking Keys

```astra
If "email" in user then
    Write "Email found"
Else
    Write "Email not found"
End check
```

Output: `Email not found`

Safe Access (Returns Nothing)

```astra
Get "email" from user save to email_value
If email_value is Nothing then
    Write "No email on file"
End check
```

Output: `No email on file`

9.10 Dictionary Iteration

```astra
Create dictionary named grades
Set "Math" in grades to 90
Set "Science" in grades to 85
Set "History" in grades to 92

Repeat for every subject in keys of grades
    Get subject from grades save to score
    Write subject
    Write ": "
    Write score
End repeat
```

Output:

```
Math: 90
Science: 85
History: 92
```

9.11 Sets: Unique Collections

A **Set** is a collection that automatically removes duplicates and provides fast membership testing.

Creating Sets

```astra
Create unique group named tags
Add "python" to group tags
Add "programming" to group tags
Add "python" to group tags
Add "data" to group tags

Write tags
```

Output: `{"python", "programming", "data"}`

Key Property: The duplicate "python" was automatically removed.

9.12 Set Operations

Membership Testing

```astra
If "python" in tags then
    Write "Python tag exists"
End check
```


Set Size

```astra
Get length of tags save to count
Write count
```

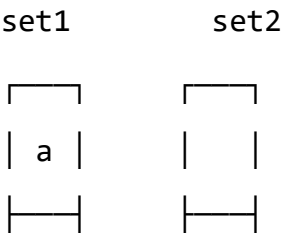
Output: `3`

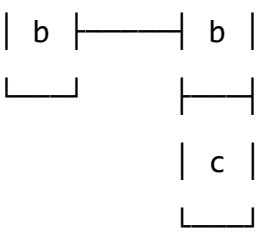
Union (Combine Sets)

```astra
Create unique group named set1
Add "a" to group set1
Add "b" to group set1

Create unique group named set2
Add "b" to group set2
Add "c" to group set2

Combine set1 and set2 save to union_set
Write union_set
```

Output: `{"a", "b", "c"}`

Venn Diagram:






```
union: {a, b, c}
```

9.13 Tuples: Fixed Lists

Tuples are immutable lists—once created, they cannot be modified.

```astra
Create fixed list named point containing 10, 20
Get item 0 from point save to x
Get item 1 from point save to y

Write "X: "
Write x
Write " Y: "
Write y
```

Output: `X: 10 Y: 20`

Immutability:

```astra
Set item 0 in point to 15    # ERROR: Cannot modify tuple
```


Use Cases:

- Coordinates (x, y, z)
- RGB colors (r, g, b)
- Database records (immutable after fetch)
- Function return multiple values

9.14 Stacks: Last In, First Out (LIFO)

A **Stack** is like a stack of plates—you can only add/remove from the top.

```astra
Create stack named history
Push "Page1" to history
Push "Page2" to history
Push "Page3" to history

Pop from history save to current
Write current

Pop from history save to previous
Write previous
```

Output:

```
Page3
Page2
```


Stack Diagram:

```
Initial:          After Pop:       After Pop Again:


Top → [Page3]
      [Page2]     Top → [Page2]
      [Page1]           [Page1]    Top → [Page1]
```

Use Cases:

- Browser back button
- Undo/redo functionality
- Function call stack
- Expression evaluation

9.15 Queues: First In, First Out (FIFO)

A **Queue** is like a line at a store—first person in is first served.

```astra
Create queue named tasks
Enqueue "Task A" to tasks
Enqueue "Task B" to tasks
Enqueue "Task C" to tasks

Dequeue from tasks save to first
Write first

Dequeue from tasks save to second
```



```
Write second
```

Output:

```
Task A
Task B
```

Queue Diagram:

```
Initial:
Front → [Task A] → [Task B] → [Task C] → Back

After First Dequeue:
Front → [Task B] → [Task C] → Back

After Second Dequeue:
Front → [Task C] → Back
```

Use Cases:

- Print job scheduling
- Message processing
- Breadth-first search
- Task scheduling

9.16 Choosing the Right Structure

Need	Use	Reason
Ordered items with indexing	List	Random access, mutable
Unique items only	Set	Automatic deduplication
Key-value pairs	Dict	Fast lookup by key

Immutable sequence	Tuple	Prevents modification
Last-in-first-out	Stack	Recent items processed first
First-in-first-out	Queue	Fair ordering

9.17 Memory and Reference Semantics

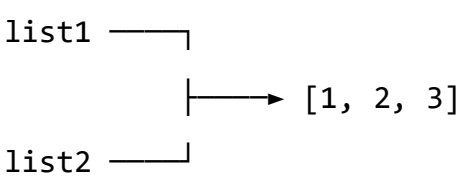
Critical Understanding:

```astra
Create list named list1
Add 1 to list1
Add 2 to list1

Set list2 to list1        # Reference, not copy
Add 3 to list2

Write list1               # Both show [1, 2, 3]
Write list2
```

Output:

```
[1, 2, 3]
[1, 2, 3]
```

Memory Diagram:



To Create an Independent Copy:

```astra
Copy list1 to list2
Add 3 to list2

Write list1               # [1, 2]
Write list2               # [1, 2, 3]
```

9.18 Complex Data Structures

List of Dictionaries (Database Rows)

```astra
Create list named users

Create dictionary named user1
Set "name" in user1 to "Alice"
Set "age" in user1 to 30
Add user1 to users

Create dictionary named user2
Set "name" in user2 to "Bob"
Set "age" in user2 to 25
Add user2 to users

Repeat for every user in users
    Get "name" from user save to name
    Write name
```



```
End repeat
```

Output:

```
Alice
Bob
```

Dictionary of Lists (Categories)

```astra
Create dictionary named courses

Create list named math_courses
Add "Algebra" to math_courses
Add "Calculus" to math_courses
Set "Math" in courses to math_courses

Create list named cs_courses
Add "Python" to cs_courses
Add "Data Structures" to cs_courses
Set "CS" in courses to cs_courses
Get "Math" from courses save to math_list
Get item 0 from math_list save to first_math
Write first_math
```

Output: `Algebra`

9.19 Performance Characteristics

Operation	List	Dict	Set	Stack	Queue
Access by index	O(1)	N/A	N/A	O(n)	O(n)
Access by key	O(n)	O(1)	N/A	N/A	N/A
Search	O(n)	O(1)	O(1)	O(n)	O(n)
Insert/Add	O(1)*	O(1)	O(1)	O(1)	O(1)
Remove	O(n)	O(1)	O(1)	O(1)	O(1)

9.20 Professional Insights

Data structures are the foundation of:

1. Algorithm Efficiency

Choosing the right structure can change $O(n^2)$ algorithms to $O(n)$.

2. Database Design

Tables are lists of dictionaries; indexes are dictionaries.

3. API Design

JSON responses map directly to dictionaries and lists.

4. System Architecture

Message queues, caches (dictionaries), and event logs (lists) power modern systems.

9.21 Key Takeaways

1. **Lists provide ordered, mutable sequences with indexing**
2. **Dictionaries map keys to values for fast lookup**

3. **Sets ensure uniqueness and fast membership testing**
4. **Tuples provide immutable sequences**
5. **Stacks (LIFO) and Queues (FIFO) model specific access patterns**
6. **Reference semantics: assignment shares, not copies**
7. **Choose structures based on access patterns and requirements**

9.22 Exercises

Beginner Level

Create a shopping list program that:
- Stores items in a list
- Allows checking if an item exists
- Displays all items
- Removes purchased items

Intermediate Level

Build a contact manager using a dictionary where:
- Keys are names
- Values are dictionaries with email and phone
- Implement add, search, update, delete operations

Advanced Level

Implement a task scheduler using a queue that:
- Accepts tasks with priorities
- Processes high-priority tasks first
- Handles task dependencies (some tasks must complete before others)
- Provides status reporting

Chapter 10: Blueprints (Object-Oriented Programming)

Bundling Data with Behavior

10.1 Why Objects Exist

As programs grow, data and behavior must stay together. Consider a game character:

Without Objects (Scattered):

```astra
Set player_name to "Hero"
Set player_health to 100
Set player_level to 1

To do PlayerTakeDamage with inputs amount
    Subtract amount from player_health
End function
```

Problems:

- Variables scattered globally
- Easy to forget which functions work with which data
- Difficult to create multiple players

Objects solve this by bundling properties (data) and actions (behavior).

10.2 The Four Blueprint Laws

Law 1: Blueprints Are Templates

A blueprint defines structure; objects are instances created from that template.

Law 2: Properties Hold State

Each object has its own copy of properties.

Law 3: Actions Define Behavior

Actions (methods) operate on the object's properties.

Law 4: Encapsulation

Related data and behavior are packaged together.

10.3 Defining a Blueprint

Astra calls classes **Blueprints** to emphasize their role as templates.

```astra
Define Blueprint named Dog
    Has name set to "Unknown"
    Has age set to 0
    Action Bark
        Write "Woof! My name is "
        Write name
    End Action
End Blueprint
```


Blueprint Diagram (Template Only):

```
┌────────── Blueprint: Dog ──────────┐
│  Properties (State):               │
│    • name → "Unknown"              │
│    • age → 0                       │
│                                    │
│  Actions (Behavior):               │
│    • Bark()                        │
└────────────────────────────────────┘
```

Key Point: The blueprint exists in memory but is **not** an object yet.

10.4 Creating Objects (Instantiation)

```astra
Create object my_dog from Dog
```

Instantiation Diagram:

```
Blueprint: Dog (Template)
     │
     │ Create object
     ▼
┌────────── Object: my_dog ──────────┐
│  Properties:                       │
│    name → "Unknown"                │
│    age → 0                         │
```



```
│                                    │
│  Actions:                          │
│    Bark()                          │
└────────────────────────────────────┘
```

Now `my_dog` is an actual object with memory allocated.

10.5 Setting Object Properties

```astra
Set name in my_dog to "Max"
Set age in my_dog to 3

Write name in my_dog
```

Output: `Max`

State Diagram:

```
my_dog:
  ├─ name: "Unknown" → "Max"
  └─ age: 0 → 3
```

10.6 Calling Object Actions

```astra
Run Bark in my_dog
```



```
```

Output: `Woof! My name is Max`

Method Call Flow:

```
Caller
  │
  ├─ Run Bark in my_dog
  │
  ▼
my_dog object
  │
  ├─ Access name property
  │
  └─ Execute Bark action
       │
       └─ Output: "Woof! My name is Max"
```

10.7 Multiple Objects, Independent State

```astra
Create object dog1 from Dog
Set name in dog1 to "Max"
Set age in dog1 to 3

Create object dog2 from Dog
Set name in dog2 to "Bella"
Set age in dog2 to 5
```



```
Run Bark in dog1
Run Bark in dog2
```

Output:

```
Woof! My name is Max
Woof! My name is Bella
```

Memory Diagram:

```
dog1 ──────► { name: "Max", age: 3 }
             { Bark() }

dog2 ──────► { name: "Bella", age: 5 }
             { Bark() }
```

Critical Insight: Each object has its **own** property values. Changing one doesn't affect the other.

10.8 Actions That Modify State

```astra
Define Blueprint named Counter
    Has count set to 0

    Action Increment
        Add 1 to count
    End Action
```



```
    Action GetValue
        Return count
    End Action
End Blueprint

Create object counter1 from Counter
Run Increment in counter1
Run Increment in counter1
Run GetValue in counter1 save to value

Write value
```

Output: `2`

State Evolution:

```
counter1:
  count: 0 → 1 → 2
```

10.9 Actions with Parameters

```astra
Define Blueprint named BankAccount
    Has balance set to 0

    Action Deposit with inputs amount
        Add amount to balance
    End Action
```



```
    Action Withdraw with inputs amount
        If amount is greater than balance then
            Write "Insufficient funds"
            Return False
        End check

        Subtract amount from balance
        Return True
    End Action

    Action GetBalance
        Return balance
    End Action
End Blueprint

Create object account from BankAccount
Run Deposit in account with inputs 100
Run Withdraw in account with inputs 30
Run GetBalance in account save to current_balance

Write current_balance
```

Output: `70`

10.10 Constructor-Like Initialization

```astra
Define Blueprint named Person
```



```
    Has name set to Nothing
    Has age set to Nothing

    Action Initialize with inputs person_name, person_age
        Set name to person_name
        Set age to person_age
    End Action

    Action Introduce
        Write "Hello, I'm "
        Write name
        Write " and I'm "
        Write age
        Write " years old"
    End Action
End Blueprint

Create object person from Person
Run Initialize in person with inputs "Alice", 30
Run Introduce in person
```

Output: `Hello, I'm Alice and I'm 30 years old`

10.11 Blueprints Referencing Blueprints

```astra
Define Blueprint named Engine
    Has horsepower set to 0
```



```
    Action Start
        Write "Engine started"
    End Action
End Blueprint

Define Blueprint named Car
    Has brand set to "Unknown"
    Has engine set to Nothing

    Action InstallEngine with inputs new_engine
        Set engine to new_engine
    End Action

    Action StartCar
        If engine is Nothing then
            Write "No engine installed"
            Return
        End check

        Run Start in engine
        Write brand
        Write " is ready to drive"
    End Action
End Blueprint

Create object my_engine from Engine
Set horsepower in my_engine to 200

Create object my_car from Car
```



```
Set brand in my_car to "Tesla"
Run InstallEngine in my_car with inputs my_engine
Run StartCar in my_car
```

Output:

```
Engine started
Tesla is ready to drive
```

Object Relationship:

```
my_car ──────┐
             │
             ├─ brand: "Tesla"
             │
             └─ engine ──────► my_engine
                                  │
                                  └─ horsepower: 200
```

10.12 Comparison with Python

Concept	**Python**	**Astra**
Class def	`class Dog:`	`Define Blueprint named Dog`
Constructor	`def (self, name):`	`Action Initialize with name`
Method	`def bark(self):`	`Action Bark`
Property	`self.name = name`	`Has name set to value`
Instantiate	`dog = Dog("Max")`	`Create object dog from Dog`
Method call	`dog.bark()`	`Run Bark in dog`

Key Difference: Astra removes the confusing `self` parameter — the object context is implicit.

10.13 Encapsulation Benefits

1. Data Protection

```astra
Define Blueprint named SecureAccount
    Has balance set to 0

    Action Deposit with inputs amount
        If amount is less than 0 then
            Write "Cannot deposit negative amount"
            Return
        End check
        Add amount to balance
    End Action
End Blueprint
```

Negative deposits are prevented by the action.

2. Interface Stability

```astra
# Users interact through actions, not direct property access
Run Deposit in account with inputs 100  # Good
Set balance in account to 100           # Direct access (can bypass validation)
```

Best Practice: Prefer actions over direct property modification.

10.14 Real-World Blueprint Examples

Game Character

```astra
Define Blueprint named Character
    Has name set to "Hero"
    Has health set to 100
    Has level set to 1

    Action TakeDamage with inputs amount
        Subtract amount from health
        If health is less than 0 then
            Set health to 0
        End check
    End Action

    Action Heal with inputs amount
        Add amount to health
        If health is greater than 100 then
            Set health to 100
        End check
    End Action

    Action IsAlive
        Return health is greater than 0
    End Action
End Blueprint
```


Shopping Cart

```astra
Define Blueprint named ShoppingCart
    Has items set to Nothing
    Has total set to 0

    Action Initialize
        Create list named new_items
        Set items to new_items
    End Action

    Action AddItem with inputs name, price
        Create dictionary named item
        Set "name" in item to name
        Set "price" in item to price
        Add item to items
        Add price to total
    End Action

    Action GetTotal
        Return total
    End Action
End Blueprint
```


10.15 Professional Insights

Blueprints enable:

1. Domain Modeling

Objects represent real-world entities: User, Order, Product, Transaction.

2. Maintainability

Changes to an object's behavior are localized to its blueprint.

3. Reusability

Blueprints can be used across projects.

4. Team Collaboration

Different developers work on different blueprints.

5. Testing

Objects can be tested in isolation with mock dependencies.

10.16 Key Takeaways

1. **Blueprints bundle data (properties) and behavior (actions)**
2. **Objects are instances created from blueprints**
3. **Each object has independent state**
4. **Actions operate on the object's properties**
5. **Encapsulation protects data integrity**
6. **Astra removes boilerplate while maintaining power**

10.17 Exercises

Beginner Level

Create a `Rectangle` blueprint with:

- Properties: width, height
- Actions: CalculateArea(), CalculatePerimeter()

Intermediate Level

Create a `Student` blueprint with:

- Properties: name, grades (list)
- Actions: AddGrade(), GetAverage(), PassingStatus()

Test with multiple students.

Advanced Level

Design a library system with:

- `Book` blueprint (title, author, isCheckedOut)
- `Library` blueprint (books list)
- Actions: AddBook(), CheckOutBook(), ReturnBook(), SearchByAuthor()

Handle edge cases (book not found, already checked out).

PART IV: SYSTEMS & EXTERNAL INTEGRATION

Chapter 11: File Handling & Resources

Safe Interaction with External Storage

11.1 Files as External Memory

Files live **outside** your program. They:

- Persist after program ends
- Can be accessed by multiple programs
- May be locked, missing, or corrupted
- Represent system resources that must be managed

Because of this, file handling must be:

- **Explicit** (operations are visible)
- **Safe** (errors are handled)
- **Clean** (resources are released)

11.2 The Three Resource Laws

Law 1: Files Must Be Opened

Before reading or writing, establish a connection to the file.

Law 2: Files Must Be Closed

After use, release the file handle to prevent resource leaks.

Law 3: Cleanup Is Guaranteed

Even if errors occur, files must be closed.

11.3 File Lifecycle

Every file operation follows this pattern:

```
Open → Use → Close
```

If any stage is skipped, systems leak resources.

Lifecycle Diagram:

```
Program            File System
  │                     │
  ├─── Open ──────────►│ (Acquire handle)
  │                     │
  ├─── Read/Write ────►│ (Transfer data)
  │                     │
  └─── Close ─────────►│ (Release handle)
```

11.4 Simple File Reading

Read Entire File:

```astra
Read file "notes.txt" save to content
Write content
```


Data Flow:

```
File System: notes.txt
     │
     │ [Read operation]
     │
     ▼
Variable: content (String)
     │
     ▼
Output
```

Example notes.txt:

```
Meeting at 3 PM
Don't forget to call Sarah
```

Output:

```
Meeting at 3 PM
Don't forget to call Sarah
```

11.5 Simple File Writing

Overwrite File:

```astra
```



```
Write "Meeting at 5 PM" into file "reminder.txt"
```

Effect:

```
Before:          After:
reminder.txt → (empty or old content)
reminder.txt → "Meeting at 5 PM"
```

Warning: This **replaces** existing content completely.

11.6 Appending to Files

Add Without Deleting:

```astra
Append "
Next task: Review code" to file "reminder.txt"
```

Effect:

```
Before:          After:
Meeting at 5 PM  Meeting at 5 PM
                 Next task: Review code
```

11.7 The Resource Safety Problem

Dangerous Pattern (Missing Close):

```python
```



```
# Python example showing the problem
file = open("data.txt")
data = file.read()
# Program crashes here - file never closed!
# Result: File handle leaked
```

Problems:

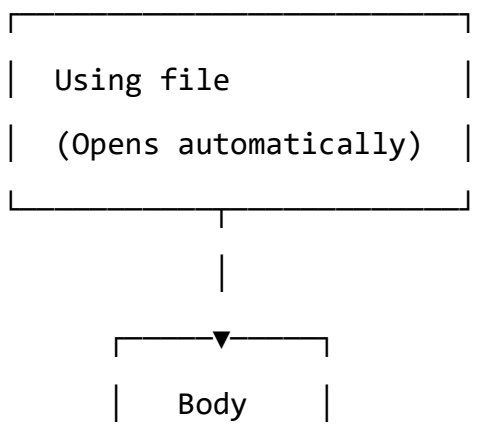
- File remains locked
- Other programs can't access it
- System runs out of file handles
- Data may not be flushed to disk

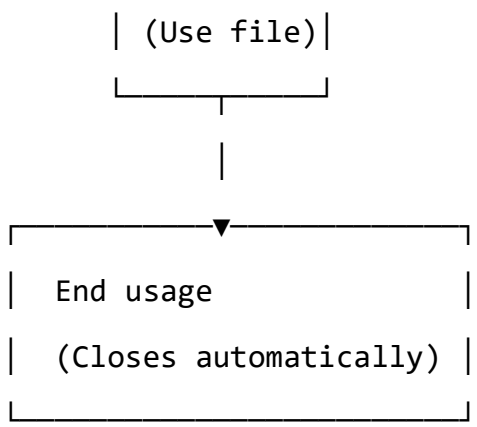
11.8 The Using Block (Context Manager)

Astra solves this with **guaranteed cleanup**:

```astra
Using file "log.txt" save to log_file
    Write "System started" into log_file
End usage
```

Lifecycle Guarantee:



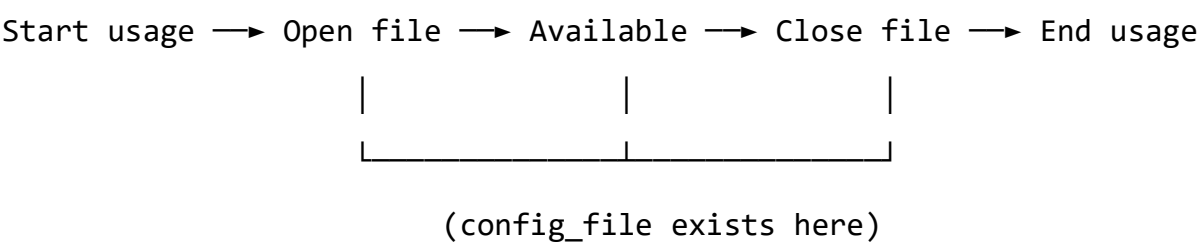


The file is ALWAYS closed, even if errors occur.

11.9 Reading Files Safely

```astra
Using file "config.txt" save to config_file
    Read from config_file save to settings
    Write "Configuration: "
    Write settings
End usage
```

File Handle Lifetime:



11.10 Writing Files Safely


```astra
Using file "output.txt" save to out_file
    Write "Line 1" into out_file
    Write "Line 2" into out_file
End usage
```

Result in output.txt:

```
Line 1
Line 2
```

11.11 Combining Using + Error Handling

The Ultimate Safety Pattern:

```astra
Attempt
    Using file "data.txt" save to data_file
        Read from data_file save to content
        Write "File content:"
        Write content
    End usage
Recover
    Write "Error reading file: "
    Write error_message
Always
    Write "File operation complete"
End attempt
```



```
┌──────────────────────────────┐
│  Attempt                     │
│  ┌────────────────────────┐  │
│  │  Using file            │  │
│  │  ┌──────────────────┐  │  │
│  │  │  Read/Write      │  │  │
│  │  └──────────────────┘  │  │
│  │  (File closed)         │  │
│  └────────────────────────┘  │
│  ┌────────────────────────┐  │
│  │  Always (runs)         │  │
│  └────────────────────────┘  │
└──────────────────────────────┘
```

Guarantee: File is closed before `Recover` or `Always` execute.

11.12 Line-by-Line File Processing

Reading Large Files:

```astra
Using file "large_log.txt" save to log_file
    Repeat for every line in log_file
        If line contains "ERROR" then
            Write line
        End check
```



```
    End repeat
End usage
```

Memory Efficiency:

```
File: [Line1, Line2, ..., LineN]
       │
       ├─► Read Line1 → Process → Discard
       ├─► Read Line2 → Process → Discard
       └─► Read LineN → Process → Discard

(Only one line in memory at a time)
```

11.13 File Existence Checking

```astra
Attempt
    Read file "optional_config.txt" save to config
    Write "Config loaded"
Recover
    If error_message contains "not found" then
        Write "Using defaults"
        Set config to "default settings"
    Else
        Write "Unexpected error: "
        Write error_message
    End check
End attempt
```



```

---

## 11.14 Working with File Paths

```astra
Set project_dir to "/home/user/projects/myapp"
Set config_file to project_dir
Add "/config.txt" to config_file

Using file config_file save to conf
    Read from conf save to data
End usage
```

11.15 CSV File Processing

```astra
Using file "users.csv" save to csv_file
    Read from csv_file save to content

    Split content by newline save to lines
    Repeat for every line in lines
        Split line by "," save to fields
        Get item 0 from fields save to name
        Get item 1 from fields save to age
        Write name
        Write " is "
        Write age
    End repeat
```


```

```
End usage
```

11.16 Professional File Patterns

Backup Before Overwrite

```astra
Attempt
    Read file "important.txt" save to backup_data
    Write backup_data into file "important.txt.backup"

    Write "New content" into file "important.txt"
    Write "File updated successfully"
Recover
    Write "Update failed, backup preserved"
End attempt
```

Atomic Write Pattern

```astra
Write data into file "data.txt.tmp"
Rename file "data.txt.tmp" to "data.txt"
```

This prevents corruption if the write fails midway.

11.17 Binary Files

```astra
Using binary file "image.png" save to img_file
    Read from img_file save to image_data
    # Process binary data
End usage
```

11.18 Key Takeaways

1. **Files are external resources that require explicit management**
2. **The Using block guarantees cleanup**
3. **Combine Using with Attempt for complete safety**
4. **Line-by-line reading conserves memory**
5. **Always handle file-not-found scenarios**
6. **Use atomic write patterns for critical data**

11.19 Exercises

Beginner Level

Write a program that:

- Reads a text file
- Counts the number of lines
- Writes the count to another file

Intermediate Level

Create a log analyzer that:

- Reads a log file line by line
- Counts ERROR, WARNING, and INFO messages
- Writes a summary report

Advanced Level

Build a configuration manager that:

- Reads JSON or key=value config files
- Provides safe defaults for missing keys
- Validates required fields
- Writes updated config back atomically

Chapter 12: Python Integration

The Bridge to Unlimited Power

12.1 Why a Bridge Exists

No language exists in isolation. The programming ecosystem contains:

- **Millions of existing libraries**
- **Decades of tested code**
- **Specialized tools** for every domain

Astra's Philosophy: Don't reinvent—integrate.

Astra achieves power not by rewriting Python's ecosystem, but by **bridging to it safely and readably.**

12.2 The Three Bridge Laws

Law 1: Explicit Import

Python libraries are imported explicitly, making dependencies visible.

Law 2: Explicit Calls

Python functions are called with clear syntax, not hidden behind operators.

Law 3: Type Safety

Data crosses the bridge with validation.

12.3 Importing Python Libraries

Syntax:

```astra
Use python library library_name
```

Example:

```astra
Use python library math
```

Bridge Diagram:

```
Astra Program
     │
     │ Use python library math
     ▼
┌──────────────────┐
│  Python Runtime  │
│  math module     │
└──────────────────┘
```

The entire `math` library becomes available.

12.4 Calling Python Functions

Syntax:

```astra
Call python function_name from library with inputs args save to result
```



```

**Example**:

```astra
Use python library math

Call python sqrt from math with inputs 25 save to root
Write root
```

Output: `5.0`

Bridge Call Flow:

```
Astra Command
     │
     ├─ Call python sqrt from math with inputs 25
     │
     ▼
Python Function: math.sqrt(25)
     │
     ├─ Compute: 5.0
     │
     ▼
Result saved to: root
     │
     ▼
Astra Variable: root = 5.0
```


```

### 12.5 Multiple Arguments

```astra
Use python library math

Call python pow from math with inputs 2, 10 save to result
Write result
```

Output: `1024.0`

Translation:
```
Astra: Call python pow from math with inputs 2, 10
  ↓
Python: math.pow(2, 10)
```

12.6 Data Science: Pandas Integration

Loading CSV Data:

```astra
Use python library pandas

Call python read_csv from pandas with inputs "sales.csv" save to data
Write "Data loaded:"
Write data
```


Data Pipeline:

```
CSV File: sales.csv
     │
     ├─ pandas.read_csv()
     │
     ▼
DataFrame Object
     │
     ▼
Astra Variable: data
```

12.7 Working with DataFrames

```astra
Use python library pandas

Call python read_csv from pandas with inputs "employees.csv" save to df

# Get column
Get property "Name" from df save to names
Write names

# Get shape
Get property "shape" from df save to dimensions
Write "Rows and columns:"
Write dimensions
```



```
```

Property Access:

```
DataFrame: df
     │
     ├─ df.Name → names
     ├─ df.shape → dimensions
     └─ df.columns → column_list
```

12.8 NumPy for Numerical Computing

```astra
Use python library numpy

Call python array from numpy with inputs [1, 2, 3, 4, 5] save to arr
Call python mean from numpy with inputs arr save to average

Write "Average:"
Write average
```

Output: `Average: 3.0`

12.9 HTTP Requests

```astra
Use python library requests
```



```
Attempt
    Call python get from requests with inputs
"https://api.github.com" save to response
    Get property "status_code" from response save to status
    Write "Status:"
    Write status
Recover
    Write "Network error:"
    Write error_message
End attempt
```

12.10 JSON Processing

```astra
Use python library json

Set data_string to '{"name": "Alice", "age": 30}'

Call python loads from json with inputs data_string save to
data_dict

Get "name" from data_dict save to name
Write name
```

Output: `Alice`

12.11 Date and Time

```astra
Use python library datetime

Call python now from datetime save to current_time
Write "Current time:"
Write current_time

Get property "year" from current_time save to year
Write "Year:"
Write year
```

12.12 File System Operations

```astra
Use python library os

Call python listdir from os with inputs "." save to files

Repeat for every file in files
    Write file
End repeat
```

12.13 Regular Expressions

```astra
Use python library re

Set text to "Contact: john@example.com"
Set pattern to "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"

Call python findall from re with inputs pattern, text save to emails

Write "Found emails:"
Repeat for every email in emails
    Write email
End repeat
```

Output:

```
Found emails:
john@example.com
```

12.14 Installing Packages (Pip Integration)

```astra
Use system command "pip install requests"
```

Best Practice: Document required packages in a requirements.txt file.

12.15 Complex Example: Web Scraping

```astra
Use python library requests
Use python library json

Attempt
    # Fetch data
    Call python get from requests with inputs "https://api.coindesk.com/v1/bpi/currentprice.json" save to response

    # Parse JSON
    Get property "text" from response save to json_text
    Call python loads from json with inputs json_text save to data

    # Extract price
    Get "bpi" from data save to bpi
    Get "USD" from bpi save to usd
    Get "rate" from usd save to price

    Write "Bitcoin price:"
    Write price
Recover
    Write "Failed to fetch data:"
    Write error_message
End attempt
```


12.16 Machine Learning: Scikit-Learn

```astra
Use python library sklearn.linear_model as linear_model
Use python library numpy as np

# Create training data
Call python array from np with inputs [[1], [2], [3], [4]] save to X
Call python array from np with inputs [2, 4, 6, 8] save to y

# Create and train model
Call python LinearRegression from linear_model save to model
Call python fit from model with inputs X, y

# Make prediction
Call python array from np with inputs [[5]] save to new_X
Call python predict from model with inputs new_X save to prediction

Write "Prediction for 5:"
Write prediction
```

Output: `Prediction for 5: [10.]`

12.17 Why Astra Doesn't Hide Python

Transparency Principle:

Astra is not a replacement for Python—it's a **human interface layer.**

Benefits:

1. **Full Power**: Access entire Python ecosystem
2. **No Waiting**: New Python libraries work immediately
3. **Community**: Leverage Python's massive community
4. **Professional Tools**: Use production-grade libraries
5. **Future-Proof**: As Python grows, Astra grows

Philosophy:

```
Astra → (Readability Layer)
   │
   └─► Python → (Power Layer)
          │
          └─► C/C++ → (Performance Layer)
```

12.18 Best Practices

1. Import at Top

```astra
Use python library pandas
Use python library numpy
Use python library matplotlib.pyplot as plt
# ... rest of program
```

2. Handle Errors

```astra
Attempt
    Call python read_csv from pandas with inputs "missing.csv" save to df
Recover
```



```
    Write "File not found, using empty dataset"
    Call python DataFrame from pandas save to df
End attempt
```

3. Document Python Dependencies

```astra
# Required Python packages:
# - pandas==2.0.0
# - numpy==1.24.0
# - requests==2.31.0
```

12.19 Professional Insights

The bridge architecture enables:

1. Rapid Prototyping

Write readable Astra code, use powerful Python libraries.

2. Team Collaboration

Non-programmers write Astra, programmers write Python extensions.

3. Gradual Migration

Start with Astra, drop to Python for performance-critical sections.

4. Educational Path

Learn programming concepts in Astra, transition to Python naturally.

12.20 Key Takeaways

1. **Astra bridges to Python for unlimited library access**
2. **Imports and calls are explicit and readable**
3. **Data crosses the bridge safely with validation**
4. **The entire Python ecosystem is available**
5. **Astra is a readability layer, not a replacement**
6. **Professional libraries work without modification**

12.21 Exercises

Beginner Level

Use the `random` library to:

- Generate 10 random numbers
- Store them in a list
- Calculate their average using Python's `statistics` module

Intermediate Level

Create a weather data analyzer:

- Fetch weather data from an API using `requests`
- Parse JSON response
- Extract temperature and conditions
- Store in a dictionary
- Handle network errors

Advanced Level

Build a data pipeline that:

- Reads CSV with `pandas`
- Cleans missing values
- Performs statistical analysis with `numpy`
- Trains a simple ML model with `sklearn`

Chapter 13: Asynchronous Programming

Time, Waiting, and Concurrency

13.1 Why Time Is Hard

Traditional programs run **synchronously**—one operation completes before the next begins:

```astra
Fetch data from server    # Wait 2 seconds
Process data              # Wait 1 second
Update display            # Wait 0.1 seconds
```

Total time: 3.1 seconds

The Problem: While waiting for the server, the program is frozen. Users see:

- Unresponsive interfaces
- Loading spinners
- Frustration

Real systems must:

- Wait for network requests
- Respond to user clicks
- Run multiple tasks simultaneously
- Keep interfaces responsive

Asynchronous programming solves this by allowing tasks to wait without blocking.

13.2 The Three Async Laws

Law 1: Async Functions May Pause

Functions marked `async` can suspend execution while waiting.

Law 2: Await Marks Wait Points

The `Await` keyword explicitly marks where pauses occur.

Law 3: The Event Loop Manages Time

A runtime scheduler decides which tasks run when.

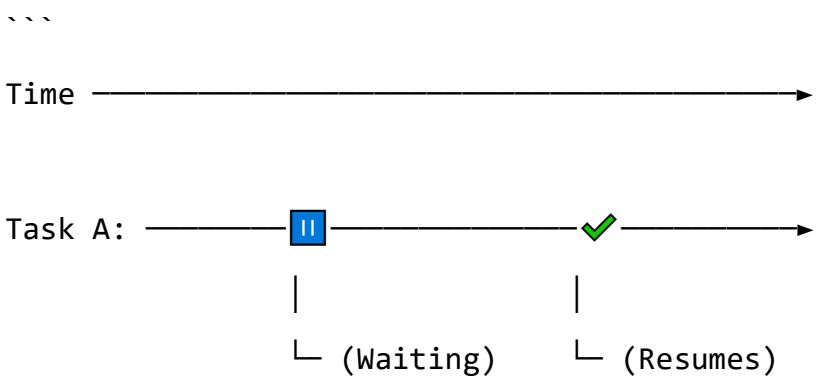
13.3 Synchronous vs. Asynchronous

Synchronous Timeline:

```
Time ──────────────────────────────────────────────►

Main Thread:
├─ Task A ──────────────────✓──────────────►
│  (UI Frozen)
│
└─ Task B ──────────►
   (Starts after A completes)
```

User Experience: Application freezes during Task A.

Asynchronous Timeline:



User Experience: Application remains interactive.

13.4 Defining Async Functions

Syntax:

```astra
To do async FunctionName
    # Function body
End function
```

Example:

```astra
```



```
To do async FetchData
    Write "Starting fetch..."
    Await Sleep for 2 seconds
    Write "Data fetched!"
End function
```

Key Marker: The `async` keyword signals this function may pause.

13.5 The Await Keyword

`Await` marks explicit pause points:

```astra
To do async ProcessOrder
    Write "Validating order..."

    Await ValidatePayment
    Write "Payment validated"

    Await ReserveInventory
    Write "Inventory reserved"

    Write "Order complete"
End function
```

Execution Flow:

```
Start ProcessOrder
    |
```



```
    ├─ Write "Validating order..."
    │
    ├─ Await ValidatePayment
    │   ├─ Suspend ⏸️
    │   │  (Other tasks can run)
    │   └─ Resume ▶️
    │
    ├─ Write "Payment validated"
    │
    ├─ Await ReserveInventory
    │   ├─ Suspend ⏸️
    │   └─ Resume ▶️
    │
    └─ Write "Order complete"
```

13.6 Calling Async Functions

From async context:

```astra
To do async Main
    Await FetchData
    Write "Done"
End function

Run async Main
```

Key Rule: You must `Await` async functions to wait for completion.

13.7 Practical Example: Web Request

```astra
Use python library aiohttp
Use python library asyncio

To do async FetchWebsite with inputs url
    Write "Fetching: "
    Write url

    # Simulate async HTTP request
    Call python get from aiohttp with inputs url save to response
    Await response

    Get property "text" from response save to content
    Return content
End function

To do async Main
    Await FetchWebsite with inputs "https://example.com" save to
page
    Write "Page length:"
    Get length of page save to length
    Write length
End function

Run async Main
```


13.8 Running Multiple Tasks Concurrently

Sequential (Slow):

```astra
To do async Sequential
    Await FetchWebsite with inputs "https://site1.com"
    Await FetchWebsite with inputs "https://site2.com"
    Await FetchWebsite with inputs "https://site3.com"
    # Total: 6 seconds (2 seconds each)
End function
```

Concurrent (Fast):

```astra
To do async Concurrent
    Create task list named tasks

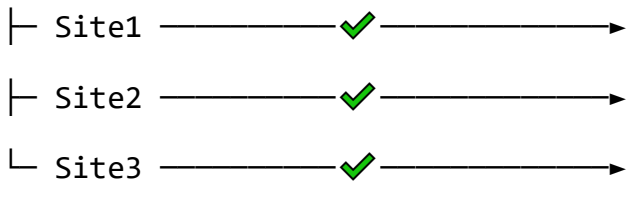
    Add FetchWebsite with inputs "https://site1.com" to tasks
    Add FetchWebsite with inputs "https://site2.com" to tasks
    Add FetchWebsite with inputs "https://site3.com" to tasks

    Await all tasks
    # Total: 2 seconds (all run simultaneously)
End function
```

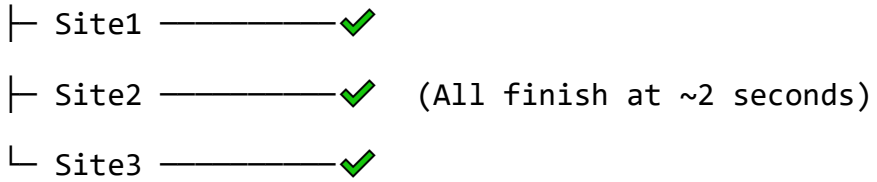
Timeline Comparison:

```
Sequential:

 
```



Concurrent:




```

---

**13.9 Event-Driven Programming**

Events allow programs to **react** rather than **poll**:

```astra
On event "ButtonClick" do HandleClick

To do HandleClick
    Write "Button was clicked!"
End function
```

Event Flow:

```
Program Running
    │
    ├─ (Waiting for events)
    │
User Action: Click
```


```

```
    │
    └─► Trigger "ButtonClick" event
         │
         └─► Execute HandleClick
              │
              └─► Write "Button was clicked!"
```

13.10 Event Types

User Interface Events

```astra
On event "KeyPress" do HandleKeyPress
On event "MouseMove" do HandleMouseMove
On event "WindowResize" do HandleResize
```

Network Events

```astra
On event "DataReceived" do ProcessData
On event "ConnectionLost" do Reconnect
```

Timer Events

```astra
On event "TimerTick" do UpdateClock
```

13.11 Async + Events Combined

```astra
On event "FileSelected" do ProcessFile

To do async ProcessFile with inputs filename
    Write "Processing: "
    Write filename

    Await ReadFile with inputs filename save to data
    Await AnalyzeData with inputs data save to results

    Write "Results:"
    Write results
End function
```

Flow:

```
User selects file
    │
    └─► Event "FileSelected" triggered
         │
         └─► ProcessFile starts
              │
              ├─ Await ReadFile ⏸
              │  (UI remains responsive)
              │
              ├─ Await AnalyzeData ⏸
              │  (Still responsive)
```

```
              │
              └─ Write results
```

13.12 Timeouts

Prevent waiting forever:

```astra
To do async FetchWithTimeout
    Attempt
        Await FetchWebsite with inputs "https://slow-site.com"
timeout 5 seconds save to data
        Write "Success"
    Recover
        If error_message contains "timeout" then
            Write "Request timed out"
        End check
    End attempt
End function
```

13.13 Cancellation

Stop tasks that are no longer needed:

```astra
Create async task fetch_task from FetchWebsite with inputs
"https://example.com"
Start task fetch_task
# User changes mind
Cancel task fetch_task
Write "Task cancelled"
```

13.14 Rate Limiting

Control how often operations occur:

```astra
To do async RateLimitedFetch with inputs urls
    Repeat for every url in urls
        Await FetchWebsite with inputs url
        Await Sleep for 1 second    # Wait between requests
    End repeat
End function
```

13.15 Progress Tracking

```astra
To do async DownloadWithProgress with inputs url
    Create progress tracker named progress

    Call python download from downloader with inputs url, progress
save to result

    On event "ProgressUpdate" from progress do ShowProgress

    Await result
End function

To do ShowProgress with inputs percent
    Write "Downloaded: "
```



```
    Write percent
    Write "%"
End function
```

13.16 Async Context Managers

Combine async with resource safety:

```astra
To do async ProcessDatabase
    Using async database "users.db" save to db
        Await ExecuteQuery in db with inputs "SELECT * FROM users"
save to users

        Repeat for every user in users
            Write user
        End repeat
    End usage
End function
```

Lifecycle:

```
Start usage
    │
    ├─ Await Open database
    │
    ├─ Execute queries
    │
```



```
    └─ Await Close database
         (Guaranteed even with errors)
```

13.17 Common Async Patterns

Retry with Backoff

```astra
To do async FetchWithRetry with inputs url
    Set attempts to 0
    Set max_attempts to 3

    Loop while attempts is less than max_attempts
        Attempt
            Await FetchWebsite with inputs url save to data
            Return data
        Recover
            Add 1 to attempts
            Write "Retry attempt: "
            Write attempts

            Multiply attempts by 2 save to wait_seconds
            Await Sleep for wait_seconds seconds
        End attempt
    End loop

    Raise error "Max retries exceeded"
End function
```

Producer-Consumer

```astra
To do async Producer with inputs queue
    Repeat 10 times using counter as i
        Await PutInQueue in queue with inputs i
        Await Sleep for 0.5 seconds
    End repeat
End function

To do async Consumer with inputs queue
    Loop while True
        Await GetFromQueue in queue save to item

        If item is Nothing then
            Stop loop
        End check

        Write "Processing: "
        Write item
    End loop
End function
```

13.18 Deadlock Prevention

Dangerous Pattern (Deadlock):

```astra
To do async Task1
```



```
    Await AcquireLock with inputs "A"
    Await AcquireLock with inputs "B"
    # Do work
    Release lock "B"
    Release lock "A"
End function

To do async Task2
    Await AcquireLock with inputs "B"  # Opposite order!
    Await AcquireLock with inputs "A"  # Deadlock risk
    # Do work
End function
```

Safe Pattern (Ordered Locking):

```astra
To do async SafeTask1
    Await AcquireLock with inputs "A"  # Always same order
    Await AcquireLock with inputs "B"
    # Do work
    Release all locks
End function

To do async SafeTask2
    Await AcquireLock with inputs "A"  # Same order
    Await AcquireLock with inputs "B"
    # Do work
    Release all locks
End function
```

13.19 Async Performance

Benefits:

- **Better resource utilization** (CPU works while waiting for I/O)
- **Improved responsiveness** (UI never freezes)
- **Higher throughput** (handle more requests with same resources)

Costs:

- **Complexity** (harder to reason about timing)
- **Debugging challenges** (non-linear execution)
- **Memory overhead** (task tracking)

When to Use Async:

- Network I/O (API calls, databases)
- File I/O (large files)
- User interfaces (keep responsive)
- Concurrent operations (web servers)

When NOT to Use:

- CPU-bound tasks (use parallelism instead)
- Simple scripts (overhead not worth it)
- When sequential logic is clearer

13.20 Professional Insights

Async programming powers:

1. Web Servers

Handle thousands of requests concurrently without threads.

2. Real-Time Applications

Chat apps, games, live dashboards remain responsive.

3. Data Pipelines

Fetch from multiple sources simultaneously.

4. IoT Systems

React to sensor events without blocking.

13.21 Key Takeaways

1. **Async functions can pause without blocking the program**
2. **Await marks explicit wait points**
3. **Events enable reactive programming**
4. **Concurrent execution improves performance for I/O-bound tasks**
5. **Combine async with error handling and resource management**
6. **Proper patterns prevent deadlocks and race conditions**

13.22 Exercises

Beginner Level

Create an async function that:

- Simulates fetching user data (await 2 seconds)
- Simulates fetching user posts (await 1 second)
- Returns combined result

Intermediate Level

Build a concurrent downloader:

- Takes list of URLs
- Downloads all concurrently
- Tracks progress
- Handles timeouts and errors
- Returns results with status codes

Advanced Level

Design a task scheduler that:

- Accepts tasks with priorities and dependencies
- Runs high-priority tasks first
- Waits for dependencies before starting tasks
- Limits concurrent tasks to N
- Provides real-time progress updates
- Handles task failures gracefully

PART V: DATA SCIENCE & MACHINE LEARNING

Chapter 14: Data Tables & Analysis

Working with Tabular Data

14.1 Data as Tables

Most real-world data is **tabular**—organized in rows and columns like a spreadsheet:

```
┌─────────┬─────────┬────────┬────────┐
│  Name   │   Age   │  City  │ Salary │
├─────────┼─────────┼────────┼────────┤
│ Alice   │   30    │   NY   │ 75000  │
│ Bob     │   25    │   LA   │ 65000  │
│ Charlie │   35    │   SF   │ 85000  │
└─────────┴─────────┴────────┴────────┘
```

Terminology:

- **Row** = Record/Observation (one person)
- **Column** = Field/Variable (one attribute)
- **Cell** = Single value

14.2 The Four Table Laws

Law 1: Rows Are Records

Each row represents one complete observation.

Law 2: Columns Are Variables

Each column represents one measured or recorded attribute.

Law 3: Cells Contain Atomic Values

Each cell holds a single, indivisible value.

Law 4: Column Types Are Consistent

All values in a column should be the same type.

14.3 Loading Datasets

Astra integrates with Pandas for powerful data manipulation:

```astra
Use python library pandas

Load dataset from "sales.csv" save to sales
```

Supported Formats:

- CSV (Comma-Separated Values)
- Excel (.xlsx, .xls)
- JSON
- SQL databases
- Parquet

14.4 Inspecting Data

View First Rows

```astra
Peek at sales
```

Output:

```
   Date        Product    Quantity    Price
0  2024-01-01  Widget A   5           10.99
1  2024-01-02  Widget B   3           15.49
2  2024-01-03  Widget A   8           10.99
...
```

Get Dimensions

```astra
Get shape of sales save to dimensions
Write dimensions
```

Output: `(1000, 4)` → 1000 rows, 4 columns

Statistical Summary

```astra
Describe sales
```



```
```

Output:

```
```

```
       Quantity         Price
count  1000.00          1000.00
mean   5.43             12.75
std    2.18             3.42
min    1.00             5.99
25%    4.00             10.99
50%    5.00             12.49
75%    7.00             15.49
max    15.00            24.99
```

```

---

## 14.5 Selecting Columns

```astra
Get column "Product" from sales save to products
Write products
```

Output:

```
0    Widget A
1    Widget B
2    Widget A
...
```

```

### 14.6 Filtering Rows

```astra
Filter sales where "Price" is greater than 15 save to expensive
Write expensive
```

Output:

```
   Date        Product    Quantity    Price
1  2024-01-02  Widget B   3           15.49
5  2024-01-06  Widget C   2           19.99
...
```

Filter Diagram:

```
All Rows
    │
    ├─ Price > 15? Yes ──► Include
    ├─ Price > 15? No  ──► Exclude
    ├─ Price > 15? Yes ──► Include
    └─ ...
```

14.7 Multiple Conditions

```astra
Filter sales where "Price" is greater than 15 and "Quantity" is greater than 5 save to filtered
```


Logic:

```
Row included IF:
  (Price > 15) AND (Quantity > 5)
```

14.8 Sorting Data

```astra
Sort sales by "Price" descending save to sorted_sales
Peek at sorted_sales
```

Output:

```
   Date        Product    Quantity    Price
999 2024-12-31 Widget D   10          24.99
487 2024-06-15 Widget C   7           22.49
...
```

14.9 Creating New Columns

```astra
Calculate "Price" times "Quantity" in sales save as "Revenue"
```

Result:

```

 
```

```
   Date        Product    Quantity    Price    Revenue
0  2024-01-01  Widget A   5           10.99    54.95
1  2024-01-02  Widget B   3           15.49    46.47
...
```

Column Operation Diagram:

```
Price Column    Quantity Column    Revenue Column
     |               |                  |
     10.99   ×       5          =       54.95
     15.49   ×       3          =       46.47
     ...
```

14.10 Aggregation

Sum

```astra
Sum column "Revenue" in sales save to total_revenue
Write "Total Revenue: "
Write total_revenue
```

Average

```astra
Average column "Price" in sales save to avg_price
Write "Average Price: "
Write avg_price
```


Count

```astra
Count rows in sales save to num_sales
Write "Number of sales: "
Write num_sales
```

14.11 Grouping Data

```astra
Group sales by "Product" save to grouped

Aggregate grouped
    Sum "Revenue"
    Average "Quantity"
End aggregate save to summary

Write summary
```

Output:

```
Product     Revenue_sum    Quantity_avg
Widget A    15420.50       5.2
Widget B    12380.75       4.8
Widget C    18750.25       6.1
```

Grouping Concept:

```
All Rows
    |
```



```
    ├─ Product = "Widget A" ─┐
    ├─ Product = "Widget A" ─┤
    ├─ Product = "Widget B" ─┼─► Groups
    ├─ Product = "Widget B" ─┤
    └─ Product = "Widget C" ─┘
         │
         └─► Aggregate each group
```

14.12 Handling Missing Data

Detect Missing Values

```astra
Count missing in sales save to missing_count
Write "Missing values per column:"
Write missing_count
```

Output:

```
Date        0
Product     0
Quantity    5
Price       12
```

Remove Rows with Missing Data

```astra
```


Drop rows with missing data from sales save to clean_sales

```

---

**Fill Missing Values**

```astra
Fill missing in column "Quantity" of sales with 0 save to
filled_sales
```

Before:

Quantity

5

NaN

3

NaN

After:

Quantity

5

0

3

0

14.13 Data Cleaning Pipeline

```astra
Use python library pandas

# Load data
```


```

```
Load dataset from "raw_data.csv" save to data

Inspect
Write "Original shape:"
Get shape of data save to original_shape
Write original_shape

Remove duplicates
Remove duplicates from data save to data

Handle missing values
Fill missing in data with 0 save to data

Filter invalid records
Filter data where "Age" is greater than 0 save to data
Filter data where "Age" is less than 150 save to data

Final check
Write "Clean shape:"
Get shape of data save to clean_shape
Write clean_shape

Save cleaned data
Save data to "clean_data.csv"
```

14.14 Merging Datasets

```astra

 
```

```
Load dataset from "customers.csv" save to customers
Load dataset from "orders.csv" save to orders

# Join on customer ID
Merge customers and orders on "CustomerID" save to combined

Peek at combined
```

Join Types:

Inner Join (only matching records)

```
Customers:          Orders:             Result:
ID  Name            ID  Product         ID  Name    Product
1   Alice           1   Book            1   Alice   Book
2   Bob             3   Pen             3   Carol   Pen
3   Carol
```

Left Join (all from left, matching from right)

```
Result:
ID  Name    Product
1   Alice   Book
2   Bob     NaN
3   Carol   Pen
```

14.15 Pivoting Data

Transform from long to wide format:

```astra
```



```
Load dataset from "sales_long.csv" save to data

Pivot data with rows "Date" columns "Product" values "Revenue" save
to pivoted
```

Before (Long Format):

```
Date        Product    Revenue
2024-01-01  Widget A   100
2024-01-01  Widget B   150
2024-01-02  Widget A   120
2024-01-02  Widget B   160
```

After (Wide Format):

```
Date        Widget A    Widget B
2024-01-01  100         150
2024-01-02  120         160
```

14.16 Time Series Operations

```astra
Load dataset from "stock_prices.csv" save to stocks

# Convert to datetime
Convert column "Date" in stocks to datetime save to stocks

# Set date as index
```



```
Set index of stocks to "Date" save to stocks

# Resample to monthly average
Resample stocks to monthly using average save to monthly_avg

Write monthly_avg
```

14.17 String Operations

```astra
# Convert to lowercase
Transform column "Product" in sales using lowercase save to sales

# Extract patterns
Extract pattern "[A-Z]" from column "Product" in sales save to letters

# Replace values
Replace "Widget" with "Item" in column "Product" of sales save to sales
```

14.18 Categorical Data

```astra
# Convert to category
Convert column "Product" in sales to category save to sales
```



```
# Get unique values
Get unique values from column "Product" in sales save to products
Write "Unique products:"
Write products

# Value counts
Count values in column "Product" of sales save to counts
Write counts
```

Output:

```
Unique products:
['Widget A', 'Widget B', 'Widget C']

Widget A    350
Widget B    280
Widget C    370
```

14.19 Professional Data Workflow

```astra
Use python library pandas

To do AnalyzeSales with inputs filename

    1. Load
    Write "Loading data..."
    Load dataset from filename save to data
```



```
2. Inspect
Write "Data shape:"
Get shape of data save to shape
Write shape

3. Clean
Write "Cleaning..."
Remove duplicates from data save to data
Drop rows with missing data from data save to data

4. Transform
Write "Transforming..."
Calculate "Price" times "Quantity" in data save as "Revenue"

5. Aggregate
Write "Aggregating..."
Group data by "Product" save to grouped
Aggregate grouped sum "Revenue" save to summary

6. Sort
Sort summary by "Revenue" descending save to summary

7. Report
Write "Top Products by Revenue:"
Peek at summary

8. Save
Save summary to "sales_summary.csv"
Write "Analysis complete"
```



```
End function

Run AnalyzeSales with inputs "sales_data.csv"
```

14.20 Performance Optimization

Chunked Reading

For large files:

```astra
Load dataset from "huge_file.csv" in chunks of 10000 save to chunks

Repeat for every chunk in chunks
    Process chunk
End repeat
```

Memory Usage

```astra
Get memory usage of data save to memory
Write "Memory used: "
Write memory
Write " MB"
```

14.21 Professional Insights

Tabular data operations form the foundation of:

1. Business Intelligence

Sales reports, customer analytics, inventory management.

2. Scientific Research

Experimental data, survey results, measurements.

3. Machine Learning

Feature engineering, data preprocessing, model input.

4. Financial Analysis

Stock prices, transaction records, risk assessment.

14.22 Key Takeaways

1. **Tables organize data in rows (records) and columns (variables)**
2. **Pandas integration provides powerful data manipulation**
3. **Filtering, sorting, and aggregation are fundamental operations**
4. **Data cleaning is essential before analysis**
5. **Grouping enables summarization by categories**
6. **Merging combines data from multiple sources**
7. **Professional workflows follow load-inspect-clean-transform-analyze patterns**

14.23 Exercises

Beginner Level

Load a CSV of student grades and:

- Display first 10 rows
- Calculate average grade
- Find highest and lowest grades
- Count how many students passed (grade ≥ 60)

Intermediate Level

Analyze sales data:

- Load monthly sales CSV
- Remove duplicates and missing values
- Calculate revenue for each sale
- Group by product category
- Identify top 5 products by revenue
- Create a summary report

Advanced Level

Build a complete data pipeline:

- Load customer and transaction data from separate files
- Clean both datasets
- Merge on customer ID
- Calculate customer lifetime value
- Segment customers (high/medium/low value)
- Identify trends over time
- Generate executive summary
- Export results in multiple formats

Chapter 15: Data Visualization

Turning Data Into Insight

15.1 Why Visualization Matters

Humans are visual creatures. We can:

- Detect patterns in images instantly
- Compare magnitudes visually
- Spot outliers at a glance

A table of numbers:

```
Month   Sales
Jan     1200
Feb     1350
Mar     1280
Apr     1450
May     1620
```

Is harder to grasp than a line chart showing the upward trend.

Visualization transforms data into insight.

15.2 The Four Visualization Laws

Law 1: Choose the Right Chart Type

Different relationships require different visualizations.

Law 2: Axes Must Be Clear

Viewers should understand what each axis represents without guessing.

Law 3: Simplicity Over Decoration

Remove chart junk; keep only what conveys information.

Law 4: Accessibility Matters

Consider color blindness, labels, and clarity.

15.3 The Visualization Pipeline

```
Dataset
    │
    ├─ Select Columns
    │
    ├─ Choose Chart Type
    │
    ├─ Configure Axes/Labels
    │
    └─ Render
```


15.4 Basic Visualization Syntax

```astra
Visualize dataset_name as ChartType
    Set X to "column_name"
    Set Y to "column_name"
    Set Title to "Chart Title"
    Set X label to "X Axis Label"
    Set Y label to "Y Axis Label"
End visualization
```

15.5 Line Charts: Trends Over Time

Use Case: Show how a value changes over time.

```astra
Use python library pandas

Load dataset from "revenue.csv" save to revenue

Visualize revenue as LineChart
    Set X to "Month"
    Set Y to "Revenue"
    Set Title to "Monthly Revenue Trend"
    Set X label to "Month"
    Set Y label to "Revenue ($)"
End visualization
```


Visual Representation:

```
Revenue ($)
▲
│        ●———●
│     ●—┘
│  ●—┘
│●—┘
└──────────────────► Month
Jan Feb Mar Apr May
```

When to Use:

- Stock prices over time
- Temperature changes
- Website traffic trends
- Sales over months/years

15.6 Bar Charts: Comparing Categories

Use Case: Compare discrete categories.

```astra
Load dataset from "sales_by_product.csv" save to sales

Visualize sales as BarChart
    Set X to "Product"
    Set Y to "Sales"
    Set Title to "Sales by Product"
```


End visualization

```

**Visual Representation**:

```

Sales

▲

│ ██

│ ██ ████

│ ██ ████ ███

└──────────────────► Product

A B C

```

**When to Use**:

- Comparing products
- Survey responses
- Regional sales
- Market share

**Use Case**: Compare discrete categories.

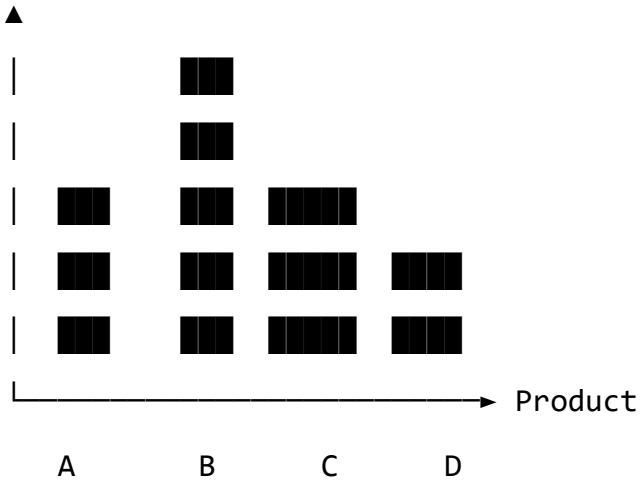
```astra
Load dataset from "sales_by_product.csv" save to sales

Visualize sales as BarChart
    Set X to "Product"
    Set Y to "Sales"
    Set Title to "Sales by Product"
    Set X label to "Product Name"
    Set Y label to "Sales ($1000s)"
```


```

```
    Set Color to "steelblue"
End visualization
```

Visual Representation:



When to Use:

Comparing products, services, or categories

Survey responses across groups

Regional comparisons

Market share analysis

Performance metrics by team/department

Horizontal vs. Vertical:

Horizontal bars (better for long labels)

```
Visualize sales as HorizontalBarChart
    Set Y to "Product"
    Set X to "Sales"
    Set Title to "Sales by Product"
End visualization
```


Result:

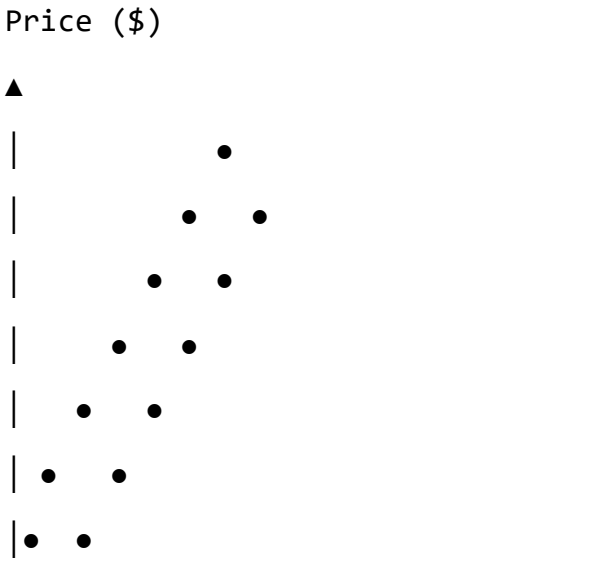
15.7 Scatter Plots: Relationships Between Variables

Use Case: Explore relationships and correlations between two continuous variables.

```
Load dataset from "housing.csv" save to houses

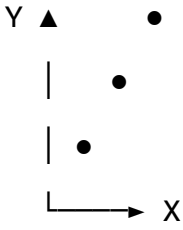
Visualize houses as ScatterPlot
    Set X to "SquareFeet"
    Set Y to "Price"
    Set Title to "House Price vs. Square Footage"
    Set X label to "Square Feet"
    Set Y label to "Price ($)"
    Set Color to "darkblue"
    Set Alpha to 0.6
End visualization
```

Visual Representation:



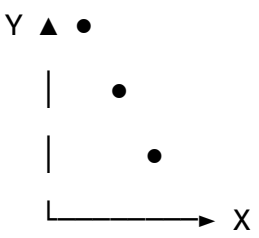
Patterns to Identify

Positive Correlation:



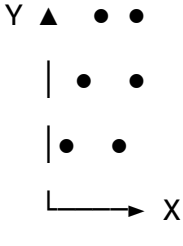
(As X increases, Y increases)

Negative Correlation:



(As X increases, Y decreases)

No Correlation:



(Random scatter)

With Color by Category:

```
Visualize houses as ScatterPlot
    Set X to "SquareFeet"
    Set Y to "Price"
    Set Color to "Neighborhood"
    Set Title to "House Prices by Neighborhood"
    Set Legend to True
End visualization
```

With Size by Third Variable:

```
Visualize houses as ScatterPlot
    Set X to "SquareFeet"
    Set Y to "Price"
    Set Size to "Bedrooms"
    Set Title to "House Prices (size = bedrooms)"
End visualization
```

Adding Trendline:

```
Visualize houses as ScatterPlot
    Set X to "SquareFeet"
    Set Y to "Price"
    Set Trendline to True
    Set Trendline type to "linear"
End visualization
```

15.8 Histograms: Understanding Distributions

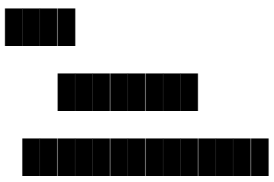
Use Case: Visualize the distribution of a single continuous variable.


```
Load dataset from "customer_ages.csv" save to customers
Visualize customers as Histogram
    Set X to "Age"
    Set Bins to 20
    Set Title to "Customer Age Distribution"
    Set X label to "Age (years)"
    Set Y label to "Frequency"
    Set Color to "skyblue"
    Set Edge color to "navy"
End visualization
```

Visual Representation:

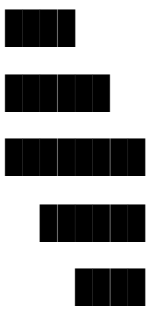
Distribution Shapes:

Normal Distribution (Bell Curve):



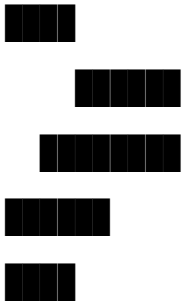
Most values cluster around the mean.

Skewed Right:



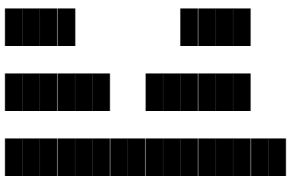
Long tail to the right (positive skew).

Skewed Left:



Long tail to the left (negative skew).

Bimodal:



Two distinct peaks (two subgroups).

Uniform:

All values roughly equally likely.

Customizing Bins:

```
# Too few bins - lose detail
Visualize data as Histogram
    Set X to "Value"
    Set Bins to 5
End visualization

# Too many bins - too noisy
Visualize data as Histogram
    Set X to "Value"
    Set Bins to 100
End visualization

# Just right - clear pattern
Visualize data as Histogram
    Set X to "Value"
    Set Bins to 30
End visualization
```

Rule of Thumb: Number of bins ≈ √(number of data points)

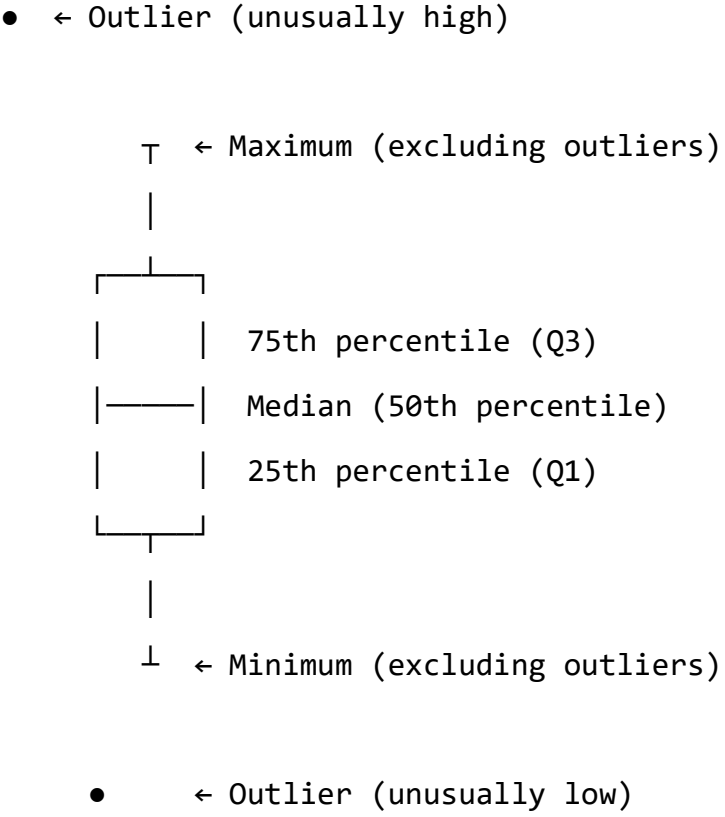
15.9 Box Plots: Statistical Summaries

Use Case: Show distribution summary with quartiles and outliers.

```
Load dataset from "employee_salaries.csv" save to salaries
Visualize salaries as BoxPlot
    Set Y to "Salary"
    Set Groups to "Department"
    Set Title to "Salary Distribution by Department"
    Set Y label to "Annual Salary ($)"
```


End visualization

Box Plot Anatomy:



Key Statistics:

IQR (Interquartile Range): Q3 - Q1 (the box height)

Outliers: Points > 1.5 × IQR beyond Q3 or Q1

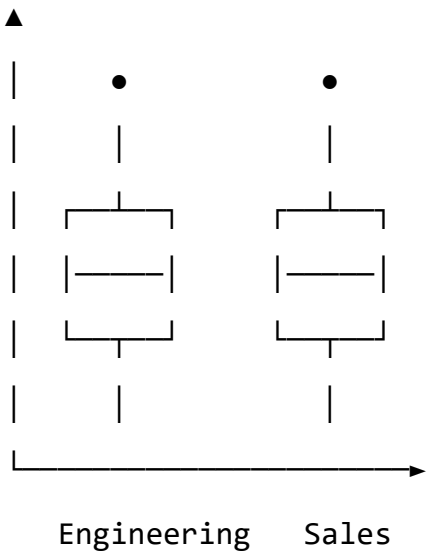
Median line: Shows central tendency

Box: Contains middle 50% of data

Comparing Multiple Groups:

```
Visualize salaries as BoxPlot
    Set Y to "Salary"
    Set X to "Department"
    Set Title to "Salary Distribution by Department"
    Set Color by "Department"
End visualization
```


Visual:



Insights:

Engineering has higher median salary

Sales has more outliers

Engineering has wider IQR (more variance)

15.10 Heatmaps: Matrix Visualization

Use Case: Visualize magnitude in a 2D matrix, especially correlations.

```
Load dataset from "feature_data.csv" save to data

# Calculate correlation matrix

Calculate correlation of data save to corr_matrix

Visualize corr_matrix as Heatmap

    Set Title to "Feature Correlation Matrix"

    Set Color map to "coolwarm"

    Set Annotations to True

    Set Format to ".2f"

End visualization
```


Visual Representation:

```
Feature1  Feature2  Feature3
Feature1    [ 1.00    0.85     -0.32  ]
Feature2    [ 0.85    1.00      0.12  ]
Feature3    [-0.32    0.12      1.00  ]
```

Colors: Blue (negative) ← White (zero) → Red (positive)

Color Schemes:

Sequential: Single color, light to dark (for magnitude)

Diverging: Two colors meeting at neutral (for positive/negative)

Qualitative: Distinct colors (for categories)

Clustering Heatmap:

```
Visualize data as ClusteredHeatmap
    Set Row cluster to True
    Set Column cluster to True
    Set Title to "Hierarchical Clustering"
End visualization
```

Groups similar rows and columns together automatically.

15.11 Pie Charts: Parts of a Whole

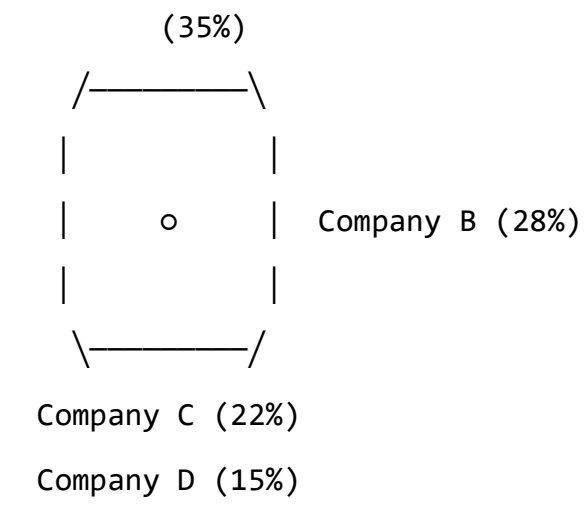
Use Case: Show proportions (use sparingly - only for 3-7 categories).

```
Load dataset from "market_share.csv" save to market
Visualize market as PieChart
    Set Values to "Share"
    Set Labels to "Company"
    Set Title to "Market Share Q4 2024"
    Set Show percentages to True
```



```
    Set Explode slice 0 by 0.1
End visualization
```

Visual:



When NOT to Use Pie Charts:

More than 7 categories (impossible to distinguish)

Small differences between values (hard to compare angles)

Time series data (use line chart instead)

Precise comparisons needed (use bar chart instead)

Better Alternative - Donut Chart:

```
Visualize market as DonutChart
    Set Values to "Share"
    Set Labels to "Company"
    Set Title to "Market Share"
    Set Hole size to 0.4
End visualization
```

Center hole makes labels clearer.

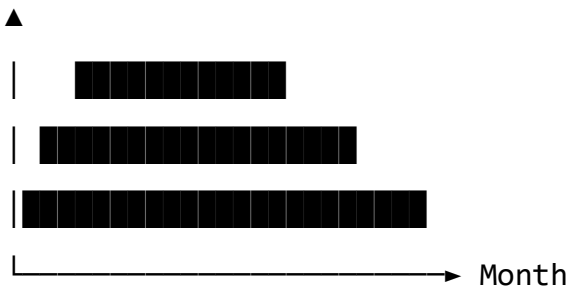
15.12 Area Charts: Cumulative Trends

Use Case: Show trends and cumulative totals over time.

Simple Area Chart:

```
Load dataset from "revenue.csv" save to revenue
Visualize revenue as AreaChart
    Set X to "Month"
    Set Y to "Revenue"
    Set Title to "Monthly Revenue"
    Set Fill alpha to 0.3
    Set Line color to "darkblue"
End visualization
```

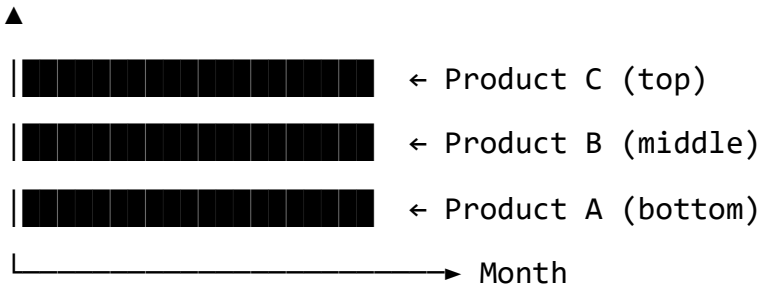
Visual:



Stacked Area Chart:

```
Visualize sales as StackedAreaChart
    Set X to "Month"
    Add series "ProductA" as Y
    Add series "ProductB" as Y
    Add series "ProductC" as Y
    Set Title to "Product Sales Over Time"
    Set Legend to True
End visualization
```


Visual:



Shows:

Individual contribution of each product

Total cumulative sales

Growth trends

100% Stacked Area:

```
Visualize sales as StackedAreaChart
    Set X to "Month"
    Add series "ProductA" as Y
    Add series "ProductB" as Y
    Add series "ProductC" as Y
    Set Normalize to 100
    Set Title to "Product Market Share Over Time"
End visualization
```

Shows proportions that always sum to 100%.

15.13 Multiple Series: Comparing Trends

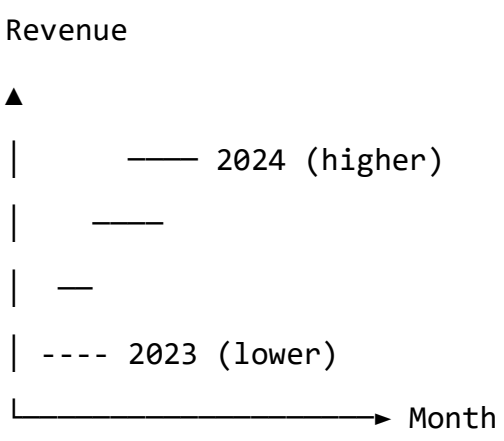
Multiple Lines:

```
Load dataset from "sales_comparison.csv" save to sales
Visualize sales as LineChart
    Set X to "Month"
    Add series "Revenue_2023" as Y with label "2023"
```



```
    Add series "Revenue_2024" as Y with label "2024"
    Set Title to "Year-over-Year Revenue Comparison"
    Set Legend position to "upper left"
    Set Y label to "Revenue ($)"
End visualization
```

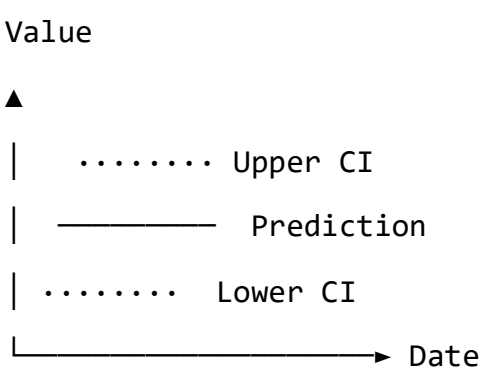
Visual:



With Confidence Intervals:

```
Visualize predictions as LineChartWithBands
    Set X to "Date"
    Set Y to "Prediction"
    Set Upper bound to "Upper_CI"
    Set Lower bound to "Lower_CI"
    Set Band alpha to 0.2
    Set Title to "Forecast with 95% Confidence Interval"
End visualization
```

Visual:



15.14 Subplots: Multiple Charts in One Figure

Grid Layout:

```
Use python library pandas
Load dataset from "complete_data.csv" save to data
Create figure with 2 rows and 3 columns save to fig
Set figure size to 15, 10

# Subplot 1: Line chart
In subplot 1 of fig
    Visualize data as LineChart
        Set X to "Date"
        Set Y to "Revenue"
        Set Title to "Revenue Trend"
    End visualization
End subplot

# Subplot 2: Bar chart
In subplot 2 of fig
    Visualize data as BarChart
        Set X to "Product"
```



```
        Set Y to "Units"
        Set Title to "Units Sold"
    End visualization
End subplot

# Subplot 3: Scatter plot
In subplot 3 of fig
    Visualize data as ScatterPlot
        Set X to "Price"
        Set Y to "Demand"
        Set Title to "Price vs Demand"
    End visualization
End subplot

# Subplot 4: Histogram
In subplot 4 of fig
    Visualize data as Histogram
        Set X to "OrderValue"
        Set Bins to 30
        Set Title to "Order Value Distribution"
    End visualization
End subplot

# Subplot 5: Box plot
In subplot 5 of fig
    Visualize data as BoxPlot
        Set Y to "Profit"
        Set Groups to "Region"
        Set Title to "Profit by Region"
```



```
    End visualization
End subplot

# Subplot 6: Pie chart
In subplot 6 of fig
    Visualize data as PieChart
        Set Values to "CategoryShare"
        Set Labels to "Category"
        Set Title to "Category Distribution"
    End visualization
End subplot

Set overall title to "Comprehensive Sales Analysis Dashboard"
Save figure fig to "dashboard.png"
Result: 2×3 grid of charts, each telling part of the story.
```

15.15 Customization and Styling

```
Complete Styling Example:
Visualize sales as LineChart
    # Data
    Set X to "Month"
    Set Y to "Revenue"

    # Titles and Labels
    Set Title to "Monthly Revenue Trend"
    Set Title size to 16
    Set X label to "Month"
    Set Y label to "Revenue ($1000s)"
```



```
    Set Label size to 12

    # Line Styling
    Set Line color to "navy"
    Set Line width to 2.5
    Set Line style to "solid"
    Set Marker to "o"
    Set Marker size to 6

    # Grid
    Set Grid to True
    Set Grid alpha to 0.3
    Set Grid linestyle to "dashed"

    # Colors and Theme
    Set Background color to "white"
    Set Face color to "whitesmoke"

    # Axis
    Set X limits to 0, 12
    Set Y limits to 0, 500
    Set X ticks to [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
    Set X tick labels to ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    # Legend
    Set Legend to True
    Set Legend position to "upper left"
    Set Legend frame to True
```



```
    # Figure
    Set Figure size to 12, 6
    Set DPI to 100
    Set Tight layout to True
End visualization
```

15.16 Annotations and Highlighting

Adding Annotations:

```
Visualize sales as LineChart
    Set X to "Date"
    Set Y to "Sales"
    Set Title to "Sales with Key Events"

    # Annotate specific point
    Add annotation at date "2024-03-15" with text "Product Launch"
    Set annotation arrow to True
    Set annotation style to "fancy"

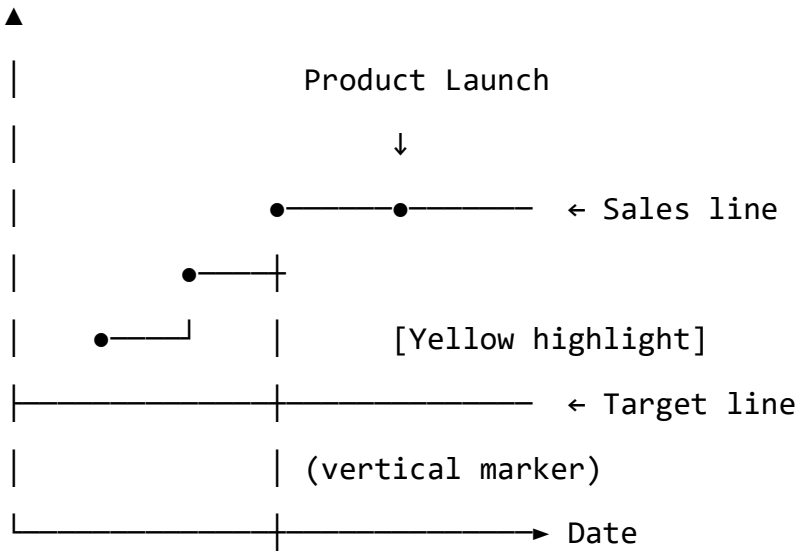
    # Add vertical line
    Add vertical line at "2024-03-15"
    Set line color to "red"
    Set line style to "dashed"
    Set line width to 2

    # Add horizontal threshold
    Add horizontal line at 50000
    Set line color to "green"
    Set line style to "dotted"
    Set line label to "Target"
```



```
    # Highlight region
    Highlight region from "2024-06-01" to "2024-08-31"
    Set highlight color to "yellow"
    Set highlight alpha to 0.2
    Set highlight label to "Summer Sale Period"
End visualization
```

Visual:



15.17 Interactive Visualizations

Using Plotly for Interactivity:

```
Use python library plotly.express as px
Load dataset from "stock_prices.csv" save to stocks
Visualize stocks as InteractiveLine
    Set X to "Date"
    Set Y to "Close"
    Set Color to "Stock"
    Set Title to "Stock Price Comparison"
```



```
    # Interactive features
    Set Hover data to ["Open", "High", "Low", "Volume"]
    Set Zoom enabled to True
    Set Pan enabled to True
    Set Range slider to True
End visualization
Save to "interactive_chart.html"
```

Interactive Features:

Hover: Show detailed data on mouseover

Zoom: Click and drag to zoom into regions

Pan: Shift the view left/right

Toggle: Click legend to show/hide series

Export: Download as PNG from browser

Interactive Scatter with Selection:

```
Visualize data as InteractiveScatter
    Set X to "Feature1"
    Set Y to "Feature2"
    Set Color to "Category"
    Set Size to "Value"

    # Enable selection
    Set Selection mode to "lasso"
    Set Export selection to True
End visualization
```

Users can draw shapes to select points.

15.18 Animated Visualizations

Time-Series Animation:

```
Use python library plotly.express as px
Load dataset from "population_by_year.csv" save to population
Visualize population as AnimatedScatter
    Set X to "GDP_per_capita"
    Set Y to "Life_expectancy"
    Set Size to "Population"
    Set Color to "Continent"
    Set Animation frame to "Year"
    Set Animation group to "Country"

    Set Title to "Global Development 1960-2020"
    Set X label to "GDP per Capita ($)"
    Set Y label to "Life Expectancy (years)"

    Set Range X to 0, 50000
    Set Range Y to 30, 90
End visualization
Save to "animated_development.html"
```

Effect: Chart animates through years, showing bubbles move and grow.

Race Bar Chart:

```
Visualize data as AnimatedBarChart
    Set X to "Value"
    Set Y to "Category"
    Set Animation frame to "Year"
    Set Title to "Top 10 Countries by GDP"
    Set Transition duration to 500
```



```
End visualization
```

Bars race each other as values change over time.

15.19 Geographic Visualizations

Choropleth Map:

```
Use python library plotly.express as px
Load dataset from "sales_by_state.csv" save to sales
Visualize sales as ChoroplethMap
    Set Locations to "State"
    Set Location mode to "USA-states"
    Set Values to "Sales"
    Set Color scale to "Blues"
    Set Title to "Sales by State"
    Set Scope to "usa"
    Set Show state borders to True
End visualization
```

Result: US map with states colored by sales volume (light = low, dark = high).

Scatter on Map:

```
Visualize stores as ScatterMap
    Set Latitude to "Lat"
    Set Longitude to "Lon"
    Set Size to "Revenue"
    Set Color to "Region"
    Set Hover name to "StoreName"
    Set Title to "Store Locations and Revenue"
End visualization
```


Bubble Map:

```
Visualize cities as BubbleMap
    Set Latitude to "Lat"
    Set Longitude to "Lon"
    Set Size to "Population"
    Set Color to "GDP_per_capita"
    Set Title to "Global Cities"
End visualization
```

15.20 3D Visualizations

When to Use 3D:

Surface plots (mathematical functions)

True 3D spatial data

Molecular structures

Terrain elevation

When NOT to Use 3D:

Regular bar charts (distorts perception)

Pie charts (even worse)

Time series (2D is clearer)

3D Scatter Plot:

```
Use python library plotly.express as px
Visualize data as Scatter3D
    Set X to "Feature1"
    Set Y to "Feature2"
    Set Z to "Feature3"
    Set Color to "Cluster"
```



```
    Set Title to "3D Cluster Visualization"
    Set Size to 5
End visualization
```

3D Surface Plot:

```
Use python library matplotlib.pyplot as plt
Create 3D surface from function
    Set X range to -5, 5
    Set Y range to -5, 5
    Set Function to "sin(sqrt(x^2 + y^2))"
    Set Title to "3D Surface"
    Set Color map to "viridis"
End surface
```

15.21 Small Multiples (Facet Grids)

Compare Across Categories:

```
Load dataset from "sales_by_region_product.csv" save to sales
Visualize sales as FacetGrid
    Set Rows to "Region"
    Set Columns to "Product"
    Set Height to 3
    Set Aspect to 1.5

    For each facet create LineChart
        Set X to "Month"
        Set Y to "Sales"
        Set Color to "steelblue"
    End facet
```



```
    Set Overall title to "Sales by Region and Product"
End visualization
```

Result: Grid of small charts, one for each Region×Product combination.

```
Product A    Product B    Product C
North      [chart]      [chart]      [chart]
South      [chart]      [chart]      [chart]
East       [chart]      [chart]      [chart]
West       [chart]      [chart]      [chart]
```

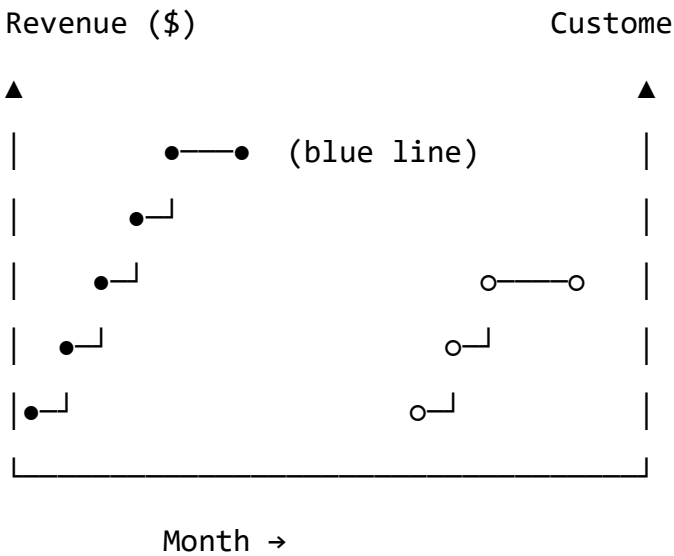
15.22 Dual-Axis Charts

Two Different Scales:

```
Visualize metrics as DualAxisChart
    Set X to "Month"
    # Left Y-axis (primary)
    Set Y1 to "Revenue"
    Set Y1 label to "Revenue ($)"
    Set Y1 color to "blue"

    # Right Y-axis (secondary)
    Set Y2 to "CustomerCount"
    Set Y2 label to "Number of Customers"
    Set Y2 color to "red"
    Set Title to "Revenue and Customer Growth"
    Set Legend to True
End visualization
```


Visual:



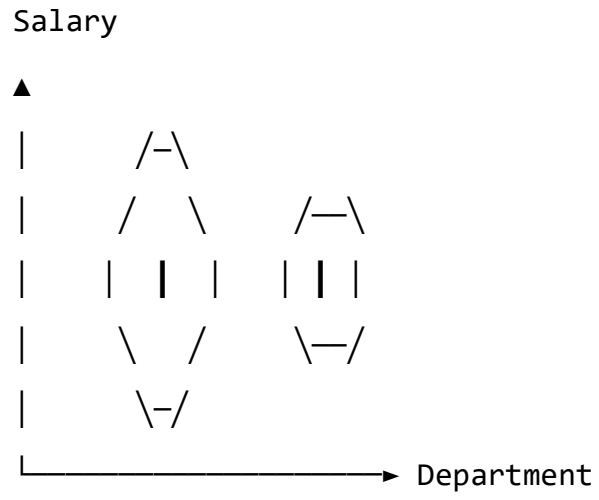
Warning: Use sparingly - can be misleading if scales are manipulated.

15.23 Violin Plots: Distribution + Density

Combines Box Plot with Density:

```
Visualize salaries as ViolinPlot
    Set Y to "Salary"
    Set X to "Department"
    Set Title to "Salary Distribution by Department"
    Set Show quartiles to True
    Set Inner plot to "box"
End visualization
```

Visual:



Width shows density at that salary level.

15.24 Pair Plots: All Relationships

Explore All Feature Pairs:

```
Use python library seaborn as sns
Load dataset from "iris.csv" save to iris
Create pair plot from iris
    Set Hue to "Species"
    Set Diagonal kind to "hist"
    Set Title to "Iris Dataset Pair Plot"
End pair plot
```

Result: Matrix showing scatter plots for all variable pairs, with histograms on diagonal.

	SepalLength	SepalWidth	PetalLength	PetalWidth
SepalLength	[hist]	[scatter]	[scatter]	[scatter]
SepalWidth	[scatter]	[hist]	[scatter]	[scatter]
PetalLength	[scatter]	[scatter]	[hist]	[scatter]
PetalWidth	[scatter]	[scatter]	[scatter]	[hist]

15.25 Saving and Exporting

Different Formats:

```
# High-resolution PNG for presentations
Save to "chart.png" with DPI 300

# Vector PDF for publications
Save to "chart.pdf"

# SVG for web (scalable)
Save to "chart.svg"

# Interactive HTML
Save to "chart.html"

# Multiple formats at once
Save to ["chart.png", "chart.pdf", "chart.svg"]
```

Size Presets:

```
# Presentation (16:9)
Set Figure size to 16, 9

# Publication (standard)
Set Figure size to 6, 4

# Poster (large)
Set Figure size to 24, 18

# Mobile (portrait)
```



```
Set Figure size to 6, 10

# Square (social media)
Set Figure size to 10, 10
```

15.26 Color Palettes and Accessibility

Sequential Palettes (Magnitude):

```
Set Color map to "Blues"       # Light to dark blue
Set Color map to "Greens"      # Light to dark green
Set Color map to "Reds"        # Light to dark red
Set Color map to "viridis"     # Perceptually uniform
```

Diverging Palettes (Positive/Negative):

```
Set Color map to "RdBu"        # Red ← White → Blue
Set Color map to "coolwarm"    # Cool ← Neutral → Warm
Set Color map to "seismic"     # Blue ← White → Red
```

Categorical Palettes:

```
Set Color palette to "Set1"         # High contrast
Set Color palette to "tab10"        # 10 distinct colors
Set Color palette to "colorblind"   # Colorblind-safe
```

Accessibility Best Practices:

```
# Use colorblind-safe palette
Set Color palette to "colorblind10"

# Add patterns/textures, not just colors
Set Bar pattern to ["///", "\\\\\\", "|||", "---"]
```



```
# Ensure sufficient contrast
Check color contrast ratio    # Should be ≥ 4.5:1

# Label directly instead of relying only on color
Add data labels to bars
```

15.27 Themes and Styles

Pre-defined Styles:

```
Use style "seaborn"           # Clean, modern
Use style "ggplot"            # R ggplot2 style
Use style "bmh"               # Bayesian Methods for Hackers
Use style "fivethirtyeight"   # FiveThirtyEight style
Use style "dark_background"   # Dark theme
```

Custom Theme:

```
Create custom style named "corporate"
    Set Font to "Arial"
    Set Font size to 12
    Set Title size to 16
    Set Background color to "white"
    Set Grid color to "lightgray"
    Set Grid alpha to 0.5
    Set Line width to 2
    Set Color palette to ["#003366", "#FF9900", "#669933"]
End style

Apply style "corporate"
```



```
# Now all subsequent charts use this style
Visualize data as LineChart
    # Uses corporate style automatically
End visualization
```

15.28 Statistical Plots with Seaborn

Regression Plot:

```
Use python library seaborn as sns
Load dataset from "data.csv" save to data
Create regression plot
    Set X to "StudyHours"
    Set Y to "TestScore"
    Set Data to data
    Set Confidence interval to 95
    Set Title to "Study Hours vs. Test Score"
End plot
```

Shows scatter plot with regression line and confidence interval.

Distribution Plot:

```
Create distribution plot
    Set X to "Heights"
    Set Bins to 30
    Set KDE to True
    Set Rug to True
    Set Title to "Height Distribution"
End plot
```

Shows histogram + smooth density curve + individual data points.

Joint Plot:

```
Create joint plot
```



```
    Set X to "Age"
    Set Y to "Income"
    Set Kind to "hex"
    Set Title to "Age vs. Income"
End plot
```

Shows 2D density hexbin + marginal distributions.

15.29 Real-Time Visualization

Live Updating Charts:

```astra
Use python library matplotlib.animation as animation
Use python library matplotlib.pyplot as plt

Create live chart named sensor_monitor

# Initialize empty data
Create list named time_data
Create list named value_data

To do Update with inputs frame
    # Fetch new data point
    Get current time save to current_time
    Read sensor value save to current_value

    # Append to data
    Add current_time to time_data
    Add current_value to value_data
```



```
    # Keep only last 100 points (sliding window)
    If length of time_data is greater than 100 then
        Remove item 0 from time_data
        Remove item 0 from value_data
    End check

    # Clear and redraw
    Clear chart sensor_monitor

    Plot time_data, value_data on sensor_monitor
        Set Title to "Live Sensor Data"
        Set X label to "Time"
        Set Y label to "Sensor Value"
        Set Line color to "blue"
        Set Line width to 2
    End plot
End function

# Start animation (updates every 1000ms)
Animate sensor_monitor with Update every 1000 milliseconds

Show live chart sensor_monitor
```

Use Cases:

- Server monitoring dashboards
- Stock price tracking
- IoT sensor displays
- Real-time analytics

- Live sports statistics

WebSocket Real-Time Updates:

```astra
Use python library flask
Use python library flask_socketio

Create web app named dashboard
Create socket connection named socketio

On route "/" method GET
    Render template "dashboard.html"
End route

To do async StreamData
    Loop while True
        # Generate/fetch new data
        Get current metrics save to data

        # Send to all connected clients
        Emit "update" with data to socketio

        # Wait before next update
        Await Sleep for 1 second
    End loop
End function

On socketio connection
```

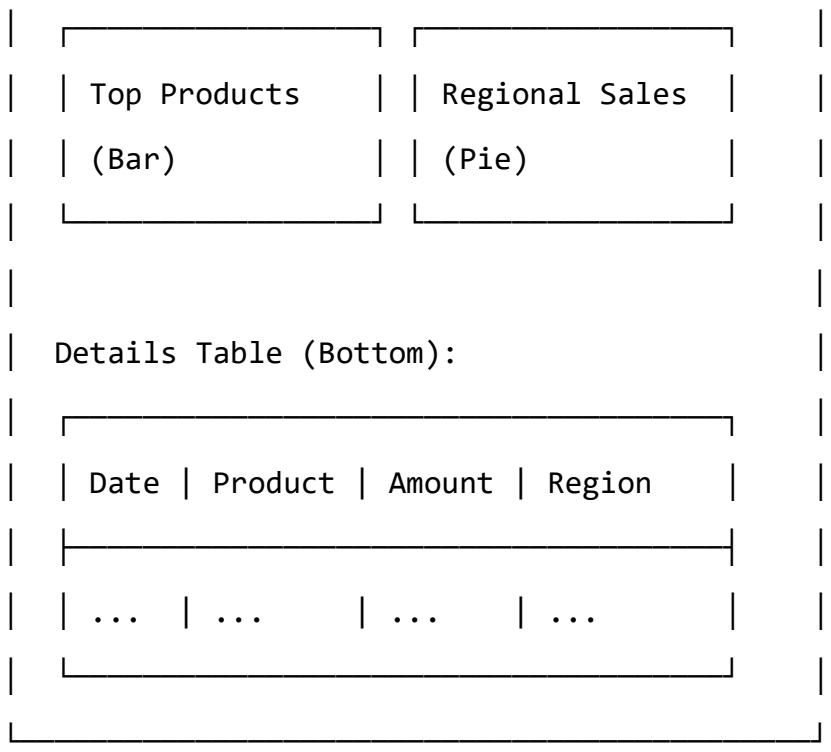


```
    Write "Client connected"
    Start async task StreamData
End connection

Run app on port 5000
```

15.30 Dashboard Design Principles

Effective Dashboard Layout:

```
┌─────────────────────────────────────────────┐
│     Sales Dashboard - Q4 2024               │
├─────────────────────────────────────────────┤
│                                             │
│  KPIs (Top):                                │
│  ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐ │
│  │ Total  │ │ Growth │ │  New   │ │ Churn  │ │
│  │ $2.5M  │ │ +15%   │ │  342   │ │ 2.1%   │ │
│  └────────┘ └────────┘ └────────┘ └────────┘ │
│                                             │
│  Main Chart (60% width):                    │
│  ┌──────────────────────────┐               │
│  │ Revenue Trend (Line)     │               │
│  │        /──\              │               │
│  │   /──/    \──\           │               │
│  │──/            \──        │               │
│  └──────────────────────────┘               │
│                                             │
│  Supporting Charts (40% width):             │
```


```

**Design Principles**:

1. **Most Important Information First**
   - KPIs at top
   - Main visualization prominent
   - Details at bottom

2. **Visual Hierarchy**
   - Size indicates importance
   - Color guides attention
   - White space prevents clutter

3. **Consistent Styling**
   - Same color palette
   - Consistent fonts

 
```

- Aligned elements

4. **Actionable Insights**
   - Highlight anomalies
   - Show trends clearly
   - Enable drill-down

Complete Dashboard Implementation:

```astra
Use python library dash
Use python library plotly.express as px

To do CreateSalesDashboard with inputs data_file
    # Load data
    Load dataset from data_file save to df

    # Calculate KPIs
    Sum column "Revenue" in df save to total_revenue
    Calculate growth rate from df save to growth_rate
    Count unique "CustomerID" in df save to new_customers
    Calculate churn rate from df save to churn_rate

    # Create dashboard app
    Create dash app named app

    # Layout
    Set layout of app to:
        # Header
```



```
    Create header
        Set text to "Sales Dashboard - Q4 2024"
        Set style to {"textAlign": "center", "fontSize": 32}
    End header

    # KPI Cards
    Create row with 4 columns
        Create card
            Set title to "Total Revenue"
            Set value to total_revenue
            Set color to "blue"
        End card

        Create card
            Set title to "Growth"
            Set value to growth_rate
            Set color to "green"
        End card

        Create card
            Set title to "New Customers"
            Set value to new_customers
            Set color to "purple"
        End card

        Create card
            Set title to "Churn Rate"
            Set value to churn_rate
            Set color to "red"
```



```
        End card
    End row

    # Main Chart
    Create row
        Create graph named revenue_trend
            Visualize df as LineChart
                Set X to "Date"
                Set Y to "Revenue"
                Set Title to "Revenue Trend"
            End visualization
        End graph
    End row

    # Supporting Charts
    Create row with 2 columns
        Create graph named top_products
            Group df by "Product" save to grouped
            Sum "Revenue" in grouped save to product_revenue
            Visualize product_revenue as BarChart
                Set Title to "Top Products"
            End visualization
        End graph

        Create graph named regional_sales
            Group df by "Region" save to regional
            Sum "Revenue" in regional save to region_revenue
            Visualize region_revenue as PieChart
                Set Title to "Regional Distribution"
```



```
                End visualization
            End graph
        End row

        # Data Table
        Create row
            Create data table
                Set data to df
                Set columns to ["Date", "Product", "Amount",
"Region"]
                Set page_size to 10
                Set filter_action to "native"
                Set sort_action to "native"
            End table
        End row
    End layout

    Return app
End function

# Create and run dashboard
Run CreateSalesDashboard with inputs "sales_data.csv" save to
dashboard_app
Run dashboard_app on port 8050

Write "Dashboard running at http://localhost:8050"
```


15.31 Professional Insights

Visualization in Industry:

1. **Business Intelligence**
- Executive dashboards require clarity over complexity
- Stakeholders need actionable insights, not raw data
- Color-coded alerts draw attention to problems
- Mobile-friendly design is essential

2. **Scientific Research**
- Publication-quality figures must be reproducible
- Vector formats (PDF, SVG) preserve quality
- Error bars and confidence intervals are mandatory
- Consistent styling across paper figures

3. **Data Journalism**
- Visualizations must tell a story
- Interactivity engages readers
- Accessibility for screen readers
- Performance on slow connections matters

4. **Machine Learning**
- Training curves diagnose model issues
- Confusion matrices reveal class-specific problems
- Feature importance guides model improvement
- Loss landscapes aid hyperparameter tuning

5. **Operations & Monitoring**

- Real-time dashboards detect anomalies early
- Historical trends identify patterns
- Alert thresholds prevent false alarms
- Drill-down capabilities enable root cause analysis

15.32 Key Takeaways

1. **Choose chart types based on the relationship you're showing**
   - Line charts for trends over time
   - Bar charts for comparing categories
   - Scatter plots for correlations
   - Histograms for distributions

2. **Always label axes clearly with units**
   - X-axis: Independent variable
   - Y-axis: Dependent variable
   - Title: What the chart shows
   - Legend: Distinguish multiple series

3. **Simplicity improves comprehension**
   - Remove chart junk
   - Limit colors to 3-5
   - Use white space effectively
   - Focus attention on key insights

4. **Accessibility matters**
   - Colorblind-safe palettes
   - Sufficient contrast
   - Text alternatives for screen readers

   - Keyboard navigation

5. **Interactive visualizations enable exploration**
   - Zoom and pan
   - Tooltips for details
   - Filters and selections
   - Export capabilities

6. **Context is crucial**
   - Show baselines and targets
   - Include comparison periods
   - Annotate significant events
   - Provide data sources

7. **Different audiences need different approaches**
   - Technical: Detailed, precise
   - Executive: High-level, actionable
   - Public: Simple, engaging
   - Academic: Rigorous, reproducible

8. **Testing prevents mistakes**
   - Verify data accuracy
   - Check edge cases
   - Test on different devices
   - Get feedback from users

15.33 Exercises

Beginner Level

Create basic visualizations:

- Load dataset with dates, categories, and values
- Create line chart showing trend over time
- Create bar chart comparing categories
- Create scatter plot showing relationship
- Add proper labels, titles, and legend
- Save all charts to files

Deliverables: 3 properly labeled charts

Intermediate Level

Build a multi-chart analysis:

- Choose dataset (sales, weather, stocks, etc.)
- Create 4 different chart types showing different aspects
- Arrange in 2×2 grid layout
- Use consistent color scheme across all charts
- Add annotations highlighting key insights
- Include statistical information (mean, median, etc.)
- Export as single high-resolution image

Deliverables: Single comprehensive visualization figure

Advanced Level

Create an interactive dashboard:

- Load real-world dataset (Kaggle, UCI, or custom)
- Calculate 4+ KPIs displayed as cards
- Create 6+ different visualizations:
  - Time series line chart
  - Category comparison bar chart
  - Correlation heatmap
  - Distribution histogram
  - Relationship scatter plot
  - Geographic map (if applicable)
- Implement filters (date range, category selection)
- Add interactivity (hover details, click-through)
- Create downloadable reports
- Deploy as web application (Flask/Dash)
- Write user documentation

Deliverables:

- Working dashboard application
- README with setup instructions
- Sample screenshots
- Data dictionary

Chapter 16: Machine Learning Fundamentals

Teaching Machines to Learn from Data

16.1 What Is Machine Learning?

Machine learning represents a fundamental shift in how we approach problem-solving with computers.

Traditional Programming:

```
┌─────────┐    ┌────────────┐    ┌──────────┐
│ Rules │ +  │  Data    │ →  │ Output │
└─────────┘    └────────────┘    └──────────┘
```

A programmer writes explicit rules:

```astra
To do ClassifyEmail with inputs email
    If email contains "free money" then
        Return "spam"
    End check

    If email contains "click here" then
        Return "spam"
    End check

    If email contains "congratulations you won" then
        Return "spam"
```



```
    End check

    Return "not spam"
End function
```

Problems with this approach:

- Endless list of rules to write
- Can't adapt to new patterns
- Doesn't work for complex problems
- Brittle and hard to maintain

Machine Learning Approach:

```
┌──────┐    ┌────────┐    ┌───────┐
│ Data │ +  │ Output │ →  │ Rules │
└──────┘    └────────┘    └───────┘

            (Labels)      (Model)
```

We show the computer examples and it learns the rules:

```astra
# Training data (examples)
Create list named training_emails
Create list named training_labels

Add "Get rich quick! Click now!" to training_emails
Add "spam" to training_labels
```



```
Add "Meeting at 3pm tomorrow" to training_emails
Add "not spam" to training_labels

Add "Your package has shipped" to training_emails
Add "not spam" to training_labels

Add "You've won $1,000,000!!!" to training_emails
Add "spam" to training_labels

# ... thousands more examples ...

# Train model
Create predictor spam_classifier using NaiveBayes
Train spam_classifier using training_emails to predict
training_labels

# Use model
Set new_email to "Free iPhone! Limited time offer!"
Predict using spam_classifier with inputs new_email save to
prediction
Write prediction  # Output: "spam"
```

Key Insight: The model learns patterns from data rather than following hand-coded rules.

When to Use Machine Learning:

✓ Use ML when:

- Patterns exist but are too complex to code

- Rules change over time
- You have lots of examples (data)
- Human experts can do the task
- Task involves pattern recognition

Examples:
- Email spam detection
- Image recognition
- Voice recognition
- Recommendation systems
- Fraud detection
- Medical diagnosis
- Stock price prediction

✗ **Don't use ML when**:
- Simple rule-based solution works
- You have very little data
- Decision must be 100% explainable
- Task requires logical reasoning
- Deterministic outcome required

Examples:
- Calculating tax (use formula)
- Sorting a list (use algorithm)
- Validating email format (use regex)
- Chess rules (use logic)

Real-World Example: House Price Prediction

Traditional Programming (impossible):

```astra
To do PredictHousePrice with inputs square_feet, bedrooms, location
    # How do you write rules for this?
    # Too many factors, complex interactions
    # Rules change by market, time, season
    # Impossible to code manually
End function
```

Machine Learning (learns from data):

```astra
# Collect historical data
Load dataset from "house_sales.csv" save to data

# Features (input)
Get columns ["SquareFeet", "Bedrooms", "Bathrooms", "Age",
"Neighborhood"] from data save to X

# Target (output we want to predict)
Get column "Price" from data save to y

# Train model on historical data
Create predictor price_model using LinearRegression
Train price_model using X to predict y
```



```
# Predict price for new house
Create list named new_house
Add 2000 to new_house    # Square feet
Add 3 to new_house       # Bedrooms
Add 2 to new_house       # Bathrooms
Add 5 to new_house       # Age
Add "Downtown" to new_house  # Neighborhood (encoded)

Predict using price_model with inputs new_house save to
predicted_price
Write "Estimated price: $"
Write predicted_price
# Output: Estimated price: $425,000
```

The model learned from 10,000+ past sales to predict new prices!

The Three Types of Machine Learning:

1. **Supervised Learning**

Learn from labeled examples (input → output pairs)

```
Training Data:
Image of cat → "cat"
Image of dog → "dog"
Image of cat → "cat"
...

Model learns: Image features → Animal type
```


Common Tasks:

- Classification (categories): spam/not spam, cat/dog/bird
- Regression (numbers): house price, temperature, stock price

---

2. **Unsupervised Learning**

Find patterns in unlabeled data

```
Training Data:
Customer purchases: [milk, bread, butter]
Customer purchases: [beer, chips, soda]
Customer purchases: [milk, cheese, yogurt]
...
Model discovers: Customer groups/segments
```

Common Tasks:

- Clustering: Group similar items
- Dimensionality reduction: Compress data
- Anomaly detection: Find outliers

---

3. Reinforcement Learning

Learn by trial and error with rewards

```
Agent: Game player
Environment: Chess board
Actions: Possible moves
Rewards: +1 for winning, -1 for losing
```


Model learns: Which moves lead to winning

```

**Common Tasks**:

- Game playing (Chess, Go, video games)
- Robotics (walking, grasping)
- Autonomous vehicles
- Resource optimization

---

**Key ML Concepts**:

**Features (Inputs)**:

Properties or characteristics used to make predictions

- House: square feet, bedrooms, age, location
- Email: words, sender, length, links
- Image: pixels, colors, edges, textures

**Labels (Outputs)**:

What we want to predict

- House price: $425,000
- Email type: spam or not spam
- Image content: cat, dog, bird

**Model**:

Mathematical function learned from data

```
Model(features) → prediction
```

 
```

```

**Training**:

Process of learning from data
- Show model many examples
- Model adjusts internal parameters
- Improves predictions over time

**Prediction/Inference**:

Using trained model on new data
- Input features
- Model computes output
- Return prediction

---

**Why Machine Learning Works**:

1. **Patterns Exist**
Real-world data contains patterns:
- Spam emails use certain words
- Cat images have certain features
- House prices correlate with size

2. **Sufficient Data**
Enough examples to learn patterns:
- 10,000+ emails to learn spam patterns
- 1,000,000+ images to recognize cats
- Years of house sales to predict prices

 
```

3. **Appropriate Algorithm**

Match algorithm to problem:

- Linear regression for linear relationships
- Neural networks for complex patterns
- Decision trees for hierarchical decisions

4. **Quality Data**

Clean, relevant, representative:

- Accurate labels
- Relevant features
- Representative samples
- Minimal noise/errors

The ML Workflow:

```
1. Define Problem
   ↓
2. Collect Data
   ↓
3. Explore & Clean Data
   ↓
4. Select Features
   ↓
5. Choose Algorithm
   ↓
6. Train Model
   ↓
7. Evaluate Performance
   ↓
```



```
8. Tune & Improve
   ↓
9. Deploy Model
   ↓
10. Monitor & Maintain
```

Machine Learning vs Traditional Programming:

Aspect	**Traditional**	**Machine Learning**
Rules	Explicitly coded	Learned from data
Adaptation	Manual updates	Automatic improvement
Complexity	Limited by programmer	Scales with data
Pattern Recog	Difficult	Natural strength
Explainability	Fully transparent	Often black box
Develop Time	Fast for simple tasks	Requires data collection
Maintenance	Code updates	Data updates

Historical Context:

1950s-1960s: Early AI - rule-based systems

1980s-1990s: Expert systems - hand-coded knowledge

2000s: Modern ML - statistical learning from data

2010s: Deep Learning - neural networks with big data

2020s: Large Language Models - AI generating human-like text

Key Enablers:

- More data (internet, sensors, digitization)
- More compute power (GPUs, cloud computing)
- Better algorithms (deep learning, gradient boosting)
- Open-source tools (TensorFlow, PyTorch, scikit-learn)

Machine Learning in Astra:

Astra makes ML accessible through:

- **Clear syntax**: `Train model using data`
- **Explicit steps**: Each stage visible
- **Python integration**: Access to powerful ML libraries
- **Visual feedback**: Easy to understand what's happening

Example:

```astra
# Crystal clear ML workflow
Load dataset from "data.csv" save to data
Split data into train and test
Train model using training data
Evaluate model on test data
Make predictions on new data
```

No cryptic syntax, no hidden magic, just clear steps.

What Makes ML "Learn"?

ML models don't truly "understand" - they optimize mathematical functions to minimize error:

```
1. Start with random parameters
2. Make predictions (usually wrong at first)
3. Calculate how wrong (loss function)
4. Adjust parameters to reduce error
5. Repeat thousands of times
6. Parameters converge to good values
```

Analogy: Learning to ride a bike

- Start: Fall immediately (high error)
- Practice: Adjust balance, pedaling (adjust parameters)
- Improve: Fall less often (error decreases)
- Master: Ride smoothly (converged to optimal parameters)

The model "learned" through repeated adjustment, not by understanding physics!

16.2 The Four ML Laws

Law 1: ML Is Pattern Recognition

Algorithms find patterns in data, not magic or consciousness.

Law 2: Garbage In, Garbage Out

Quality data matters more than sophisticated algorithms.

Law 3: Training ≠ Deployment

Models must be evaluated on unseen data before production use.

Law 4: All Models Are Wrong, Some Are Useful

No model is perfect; the goal is useful predictions.

16.3 Types of Machine Learning

1.Supervised Learning

Learn from labeled examples.

- **Classification**: Predict categories (spam/not spam, cat/dog)
- **Regression**: Predict numbers (house price, temperature)

2.Unsupervised Learning

Find patterns without labels.

- **Clustering**: Group similar items
- **Dimensionality Reduction**: Compress data

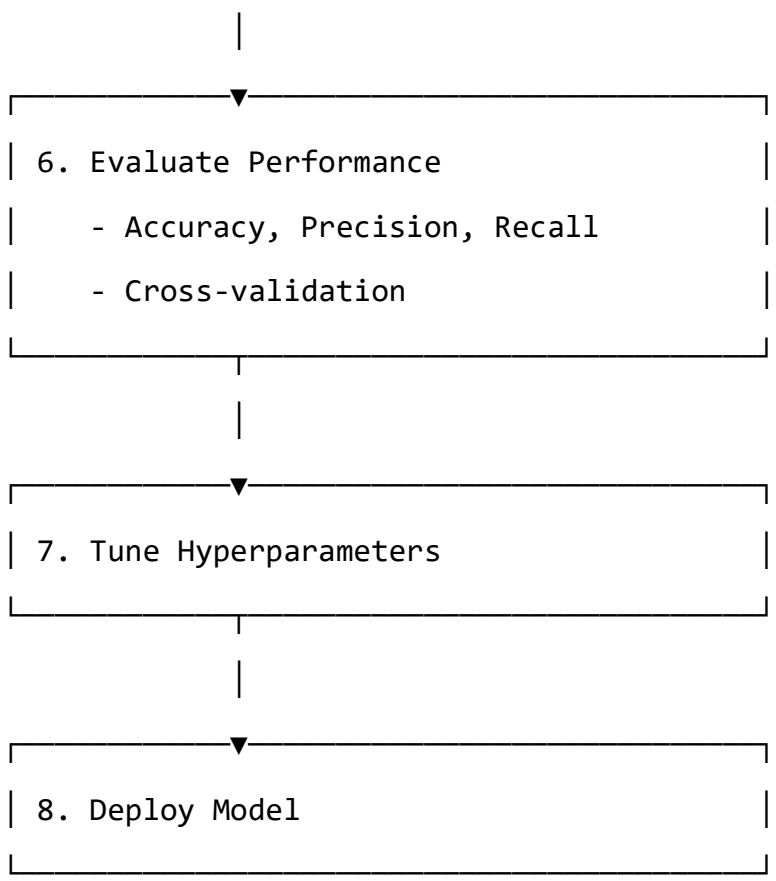
3.Reinforcement Learning

Learn by trial and error with rewards.

- Game playing
- Robotics
- Autonomous systems

16.4 The ML Pipeline

```
┌─────────────────────────────────────────┐
│ 1. Collect Data                         │
└─────────────┬───────────────────────────┘
              │
┌─────────────▼───────────────────────────┐
│ 2. Explore & Clean Data                 │
│    - Handle missing values              │
│    - Remove outliers                    │
│    - Fix errors                         │
└─────────────┬───────────────────────────┘
              │
┌─────────────▼───────────────────────────┐
│ 3. Feature Engineering                  │
│    - Select relevant features           │
│    - Create new features                │
│    - Encode categorical variables       │
└─────────────┬───────────────────────────┘
              │
┌─────────────▼───────────────────────────┐
│ 4. Split Data                           │
│    - Training set (80%)                 │
│    - Test set (20%)                     │
└─────────────┬───────────────────────────┘
              │
┌─────────────▼───────────────────────────┐
│ 5. Choose & Train Model                 │
└─────────────┬───────────────────────────┘
```

16.5 Regression: Predicting Numbers

Use Case: Predict continuous values (prices, temperatures, sales).

Complete Regression Example

```astra
Use python library pandas
Use python library sklearn.model_selection as model_selection
Use python library sklearn.linear_model as linear_model
Use python library sklearn.metrics as metrics

To do TrainHousePriceModel with inputs data_file
```



```
    Write "=== House Price Prediction Model ==="

    # Step 1: Load Data
    Write "Loading data..."
    Load dataset from data_file save to data
    Write "Loaded rows: "
    Get shape of data save to shape
    Get item 0 from shape save to num_rows
    Write num_rows

    # Step 2: Explore Data
    Write "Data summary:"
    Describe data

    # Step 3: Prepare Features
    Write "Preparing features..."
    Get columns ["SquareFeet", "Bedrooms", "Bathrooms", "YearBuilt",
"LotSize"] from data save to X
    Get column "Price" from data save to y

    # Step 4: Split Data
    Write "Splitting data (80% train, 20% test)..."
    Split data X, y into train and test with ratio 0.8 save to
X_train, X_test, y_train, y_test

    Write "Training samples: "
    Get shape of X_train save to train_shape
    Get item 0 from train_shape save to train_count
    Write train_count
```



```
    Write "Test samples: "
    Get shape of X_test save to test_shape
    Get item 0 from test_shape save to test_count
    Write test_count

    # Step 5: Create and Train Model
    Write "Training model..."
    Create predictor model using LinearRegression
    Train model using X_train to predict y_train
    Write "Training complete"

    # Step 6: Make Predictions
    Write "Making predictions on test set..."
    Predict using model with inputs X_test save to predictions

    # Step 7: Evaluate Performance
    Write "Evaluating model performance..."

    Calculate R2 score between y_test and predictions save to r2
    Calculate mean absolute error between y_test and predictions
save to mae
    Calculate root mean squared error between y_test and predictions
save to rmse

    Write "===== Model Performance ====="
    Write "R² Score: "
    Write r2
    Write "Mean Absolute Error: $"
    Write mae
```



```
    Write "Root Mean Squared Error: $"
    Write rmse

    # Step 8: Sample Predictions
    Write "Sample Predictions:"
    Repeat 5 times using counter as i
        Get item i from X_test save to sample_features
        Get item i from y_test save to actual_price
        Get item i from predictions save to predicted_price

        Write "House "
        Write i
        Write ":"
        Write "  Actual: $"
        Write actual_price
        Write "  Predicted: $"
        Write predicted_price

        Calculate actual_price minus predicted_price save to error
        Write "  Error: $"
        Write error
    End repeat

    # Step 9: Save Model
    Write "Saving model..."
    Save model to "house_price_model.pkl"
    Write "Model saved successfully"

    Return model
```



```
End function


# Run the pipeline
Run TrainHousePriceModel with inputs "houses.csv" save to
trained_model


# Use the model for new predictions
Write "===== Making New Prediction ====="
Create list named new_house
Add 2000 to new_house   # Square feet
Add 3 to new_house      # Bedrooms
Add 2 to new_house      # Bathrooms
Add 2010 to new_house   # Year built
Add 5000 to new_house   # Lot size


Predict using trained_model with inputs new_house save to
estimated_price
Write "Estimated price for new house: $"
Write estimated_price
```

Output:

```
=== House Price Prediction Model ===
Loading data...
Loaded rows: 1000
Data summary:
[Statistical summary displayed]
Preparing features...
Splitting data (80% train, 20% test)...
```



```
Training samples: 800
Test samples: 200
Training model...
Training complete
Making predictions on test set...
Evaluating model performance...

===== Model Performance =====
R² Score: 0.847
Mean Absolute Error: $23,450
Root Mean Squared Error: $31,280
Sample Predictions:
House 0:
  Actual: $325,000
  Predicted: $318,500
  Error: $6,500
[... more samples ...]
Saving model...
Model saved successfully
===== Making New Prediction =====
Estimated price for new house: $387,250
```

16.6 Performance Metrics for Regression

R² Score (Coefficient of Determination)

- Range: -∞ to 1.0
- 1.0 = Perfect predictions
- 0.8-0.9 = Good model

- 0.5-0.8 = Moderate model
- < 0.5 = Poor model
- Negative = Worse than predicting the mean

Mean Absolute Error (MAE)

- Average absolute difference between prediction and actual
- Same units as target variable
- Easy to interpret: "On average, predictions are off by $X"

Root Mean Squared Error (RMSE)

- Penalizes large errors more heavily than MAE
- Same units as target variable
- More sensitive to outliers

Mean Absolute Percentage Error (MAPE)

- Error as percentage of actual value
- Useful for comparing across different scales

16.7 Classification: Predicting Categories

Use Case: Predict discrete classes (spam/not spam, disease/healthy, customer churn).

```astra
Use python library pandas
Use python library sklearn.ensemble as ensemble
Use python library sklearn.metrics as metrics

To do TrainCustomerChurnModel with inputs data_file
    Write "=== Customer Churn Prediction ==="
```



```
    # Load data
    Load dataset from data_file save to data

    # Features
    Get columns ["Age", "MonthlyCharges", "TotalCharges",
"ContractLength", "SupportTickets"] from data save to X
    Get column "Churned" from data save to y

    # Split
    Split data X, y into train and test with ratio 0.8 save to
X_train, X_test, y_train, y_test

    # Train Random Forest Classifier
    Write "Training Random Forest Classifier..."
    Create predictor model using RandomForestClassifier with
parameters
        Set n_estimators to 100
        Set max_depth to 10
        Set random_state to 42
    End parameters

    Train model using X_train to predict y_train

    # Predict
    Predict using model with inputs X_test save to predictions

    # Evaluate
    Calculate accuracy between y_test and predictions save to
accuracy
```



```
    Calculate precision between y_test and predictions save to
precision
    Calculate recall between y_test and predictions save to recall
    Calculate f1 score between y_test and predictions save to f1

    Write "===== Model Performance ====="
    Write "Accuracy: "
    Write accuracy
    Write "Precision: "
    Write precision
    Write "Recall: "
    Write recall
    Write "F1 Score: "
    Write f1

    # Confusion Matrix
    Calculate confusion matrix between y_test and predictions save
to cm
    Write "Confusion Matrix:"
    Write cm

    # Feature Importance
    Get feature importance from model save to importance
    Write "Feature Importance:"
    Repeat for every feature in X columns using index
        Write feature
        Write ": "
        Get item index from importance save to imp_value
        Write imp_value
```



```
    End repeat


    Return model
End function


Run TrainCustomerChurnModel with inputs "customer_data.csv" save to
churn_model
```

Output:

```
=== Customer Churn Prediction ===
Training Random Forest Classifier...
===== Model Performance =====
Accuracy: 0.863
Precision: 0.791
Recall: 0.745
F1 Score: 0.767
Confusion Matrix:
[[145  12]
 [ 15  28]]
Feature Importance:
MonthlyCharges: 0.342
TotalCharges: 0.287
SupportTickets: 0.196
ContractLength: 0.112
Age: 0.063
```


16.8 Classification Metrics Explained

Confusion Matrix

```
                    Predicted
                    No      Yes
Actual  No      [  TN      FP  ]
        Yes     [  FN      TP  ]
```

- **True Negative (TN)**: Correctly predicted negative
- **False Positive (FP)**: Incorrectly predicted positive (Type I error)
- **False Negative (FN)**: Incorrectly predicted negative (Type II error)
- **True Positive (TP)**: Correctly predicted positive

Key Metrics

Accuracy:

```
(TP + TN) / (TP + TN + FP + FN)
```

Overall correctness. Can be misleading with imbalanced data.

Precision:

```
TP / (TP + FP)
```

"Of all positive predictions, how many were correct?"

Important when false positives are costly.

Recall (Sensitivity):

```
TP / (TP + FN)
```

"Of all actual positives, how many did we find?"

Important when false negatives are costly.

F1 Score:

```
2 × (Precision × Recall) / (Precision + Recall)
```

Harmonic mean of precision and recall.

16.9 Feature Engineering

Creating Better Features:

```astra
Load dataset from "raw_data.csv" save to data

# Numerical transformations
Calculate "Price" divided by "SquareFeet" in data save as "PricePerSqFt"
Calculate "TotalRooms" minus "Bedrooms" in data save as "OtherRooms"

# Log transformation for skewed data
Apply log transform to "Income" in data save as "LogIncome"
# Binning continuous variables
Bin "Age" in data into categories
```



```
    Set bins to [0, 18, 30, 50, 100]
    Set labels to ["Child", "Young Adult", "Adult", "Senior"]
End binning save as "AgeGroup"

# Interaction features
Calculate "Feature1" multiplied by "Feature2" in data save as "Interaction1_2"

# Date features
Extract year from "PurchaseDate" in data save as "PurchaseYear"
Extract month from "PurchaseDate" in data save as "PurchaseMonth"
Extract day of week from "PurchaseDate" in data save as "PurchaseDay"

# Categorical encoding
One-hot encode "Category" in data save to data
# Creates: Category_A, Category_B, Category_C columns

# Ordinal encoding (when order matters)
Ordinal encode "Education" in data with order ["High School", "Bachelor", "Master", "PhD"] save to data
```

16.10 Handling Categorical Variables

One-Hot Encoding:

```astra
# Before:
```



```
# Color: ["Red", "Blue", "Red", "Green"]

One-hot encode "Color" in data save to data

# After:
# Color_Red:   [1, 0, 1, 0]
# Color_Blue:  [0, 1, 0, 0]
# Color_Green: [0, 0, 0, 1]
```

Label Encoding:

```astra
# For ordinal categories
Label encode "Size" in data with mapping
    Set "Small" to 0
    Set "Medium" to 1
    Set "Large" to 2
End mapping save to data
```

16.11 Dealing with Missing Data

```astra
Load dataset from "data_with_missing.csv" save to data

# Strategy 1: Drop rows with any missing values
Drop rows with missing data from data save to clean_data

# Strategy 2: Drop rows with missing in specific columns
```



```
Drop rows with missing in columns ["Age", "Income"] from data save
to partial_clean


# Strategy 3: Fill with constant

Fill missing in "Age" with 0 in data save to data


# Strategy 4: Fill with mean/median

Calculate mean of "Age" in data save to age_mean

Fill missing in "Age" with age_mean in data save to data


# Strategy 5: Forward fill (time series)

Forward fill "Temperature" in data save to data


# Strategy 6: Interpolate

Interpolate missing in "Value" in data save to data
```

16.12 Cross-Validation

Why Cross-Validation?

Single train/test split might be lucky or unlucky. Cross-validation provides robust estimate.

```astra
Use python library sklearn.model_selection as model_selection


# K-Fold Cross-Validation

Perform cross validation on model with data X, y

    Set folds to 5

    Set scoring to "r2"

End validation save to cv_scores
```



```
Write "Cross-Validation Scores:"
Repeat for every score in cv_scores
    Write score
End repeat

Calculate mean of cv_scores save to mean_score
Calculate std of cv_scores save to std_score

Write "Mean R² Score: "
Write mean_score
Write " (+/- "
Write std_score
Write ")"
```

K-Fold Visualization:

```
Fold 1: [Test][Train][Train][Train][Train]
Fold 2: [Train][Test][Train][Train][Train]
Fold 3: [Train][Train][Test][Train][Train]
Fold 4: [Train][Train][Train][Test][Train]
Fold 5: [Train][Train][Train][Train][Test]

Average the 5 test scores → Robust estimate
```

16.13 Hyperparameter Tuning

Grid Search:

```astra
Use python library sklearn.model_selection as model_selection
```



```
# Define parameter grid
Create parameter grid
    Set "n_estimators" to [50, 100, 200]
    Set "max_depth" to [5, 10, 15, 20]
    Set "min_samples_split" to [2, 5, 10]
End grid save to param_grid

# Search for best parameters
Perform grid search
    Set estimator to RandomForestRegressor
    Set param_grid to param_grid
    Set cv to 5
    Set scoring to "r2"
End search with data X_train, y_train save to grid_search

# Get best model
Get best estimator from grid_search save to best_model
Get best parameters from grid_search save to best_params
Get best score from grid_search save to best_score

Write "Best Parameters:"
Write best_params
Write "Best CV Score: "
Write best_score

# Evaluate on test set
Predict using best_model with inputs X_test save to predictions
Calculate R2 score between y_test and predictions save to test_score
```



```
Write "Test Set Score: "
Write test_score
```

16.14 Model Comparison

```astra
Use python library sklearn.linear_model as linear_model
Use python library sklearn.ensemble as ensemble
Use python library sklearn.svm as svm

To do CompareModels with inputs X_train, X_test, y_train, y_test
    Create list named model_names
    Create list named model_scores

    # Linear Regression
    Write "Training Linear Regression..."
    Create predictor lr using LinearRegression
    Train lr using X_train to predict y_train
    Predict using lr with inputs X_test save to pred_lr
    Calculate R2 score between y_test and pred_lr save to score_lr
    Add "Linear Regression" to model_names
    Add score_lr to model_scores

    # Random Forest
    Write "Training Random Forest..."
    Create predictor rf using RandomForestRegressor
    Train rf using X_train to predict y_train
    Predict using rf with inputs X_test save to pred_rf
    Calculate R2 score between y_test and pred_rf save to score_rf
```



```
    Add "Random Forest" to model_names
    Add score_rf to model_scores

    # Gradient Boosting
    Write "Training Gradient Boosting..."
    Create predictor gb using GradientBoostingRegressor
    Train gb using X_train to predict y_train
    Predict using gb with inputs X_test save to pred_gb
    Calculate R2 score between y_test and pred_gb save to score_gb
    Add "Gradient Boosting" to model_names
    Add score_gb to model_scores

    # Support Vector Regression
    Write "Training SVR..."
    Create predictor svr using SVR
    Train svr using X_train to predict y_train
    Predict using svr with inputs X_test save to pred_svr
    Calculate R2 score between y_test and pred_svr save to score_svr
    Add "SVR" to model_names
    Add score_svr to model_scores

    # Display results
    Write "===== Model Comparison ====="
    Repeat for every model in model_names using index as i
        Write model
        Write ": "
        Get item i from model_scores save to score
        Write score
    End repeat
```



```
    # Find best model
    Get maximum from model_scores save to best_score
    Get index of best_score in model_scores save to best_index
    Get item best_index from model_names save to best_model_name

    Write "Best Model: "
    Write best_model_name
    Write " with R² = "
    Write best_score

    # Visualize comparison
    Visualize model comparison as BarChart
        Set X to model_names
        Set Y to model_scores
        Set Title to "Model Performance Comparison"
        Set Y label to "R² Score"
        Set Color to "steelblue"
    End visualization

    Save to "model_comparison.png"
End function

Run CompareModels with inputs X_train, X_test, y_train, y_test
```

16.15 Overfitting vs. Underfitting

Underfitting:

- Model too simple

- High training error
- High test error
- Solution: More complex model, more features

```
Performance
▲
│  − − − −  (poor fit on both)
│ ●     ●
│   ●      ●
│      ●
└──────────────→ Complexity
  Training  Test
```

Good Fit:

- Model captures patterns
- Low training error
- Low test error

```
Performance
▲
│  ──────────  (good fit on both)
│
└──────────────→ Complexity
  Training  Test
```

Overfitting:

- Model too complex
- Very low training error
- High test error

- Solution: Simpler model, regularization, more data

```
Performance
▲
│  ──────────  (training)
│       \
│        \  (test drops)
│         \
└──────────────► Complexity
```

16.16 Regularization

Prevents Overfitting:

```astra
# Ridge Regression (L2 regularization)
Create predictor ridge using Ridge with parameters
    Set alpha to 1.0
End parameters
Train ridge using X_train to predict y_train

# Lasso Regression (L1 regularization)
Create predictor lasso using Lasso with parameters
    Set alpha to 1.0
End parameters
Train lasso using X_train to predict y_train

# Elastic Net (L1 + L2)
Create predictor elastic using ElasticNet with parameters
    Set alpha to 1.0
```



```
    Set l1_ratio to 0.5
End parameters
Train elastic using X_train to predict y_train
```

Alpha (λ) parameter:

- Low alpha: Less regularization (might overfit)
- High alpha: More regularization (might underfit)
- Find optimal via cross-validation

16.17 Feature Selection

Remove Irrelevant Features:

```astra
# Method 1: Correlation with target
Calculate correlation of data save to corr_matrix
Get column "Target" from corr_matrix save to target_corr
Filter target_corr where absolute value is greater than 0.3 save to important_features

# Method 2: Feature Importance from Tree Models
Create predictor rf using RandomForestRegressor
Train rf using X_train to predict y_train
Get feature importance from rf save to importance

# Select top K features
Get top 10 features by importance save to top_features
Select columns top_features from X save to X_selected
```



```
# Method 3: Recursive Feature Elimination
Perform recursive feature elimination
    Set estimator to RandomForestRegressor
    Set n_features_to_select to 10
End elimination with data X_train, y_train save to selected_features
```

16.18 Ensemble Methods

Combine Multiple Models:

```astra
# Voting Classifier
Create list named estimators
Add LinearRegression named "lr" to estimators
Add RandomForestRegressor named "rf" to estimators
Add GradientBoostingRegressor named "gb" to estimators

Create voting regressor with estimators
Train voting regressor using X_train to predict y_train

# Stacking
Create stacked model
    Set base_models to [LinearRegression, RandomForestRegressor,
SVR]
    Set final_estimator to Ridge
End model

Train stacked model using X_train to predict y_train
```

16.19 Model Deployment

```astra
Use python library joblib

# Save trained model
Save model to "production_model.pkl" using joblib
Write "Model saved for deployment"

# Load model in production
Load model from "production_model.pkl" using joblib save to loaded_model

# Make prediction
To do PredictPrice with inputs features
    Load model from "production_model.pkl" using joblib save to model
    Predict using model with inputs features save to prediction
    Return prediction
End function

# API endpoint (conceptual)
On API request "/predict" with method POST
    Get JSON data from request save to input_data
    Run PredictPrice with inputs input_data save to result
    Return JSON response with prediction result
End endpoint
```

16.20 Complete ML Project Example

```astra
Use python library pandas
Use python library sklearn

To do CompleteMLProject with inputs data_file
    Write "============================================="
    Write "    COMPLETE MACHINE LEARNING PROJECT"
    Write "============================================="

    # Phase 1: Data Loading
    Write "Phase 1: Loading and Exploring Data"
    Load dataset from data_file save to data

    Write "Dataset shape: "
    Get shape of data save to shape
    Write shape

    Write "First few rows:"
    Peek at data

    Write "Statistical summary:"
    Describe data

    # Phase 2: Data Cleaning
    Write "Phase 2: Data Cleaning"
```



```
    # Check for missing values
    Count missing in data save to missing_counts
    Write "Missing values per column:"
    Write missing_counts

    # Handle missing values
    Fill missing in "Age" with median in data save to data
    Drop rows with missing in columns ["Target"] from data save to
data

    # Remove duplicates
    Remove duplicates from data save to data

    # Phase 3: Feature Engineering
    Write "Phase 3: Feature Engineering"

    # Create new features
    Calculate "Feature1" multiplied by "Feature2" in data save as
"Interaction"
    Apply log transform to "Income" in data save as "LogIncome"

    # Encode categorical variables
    One-hot encode "Category" in data save to data

    # Phase 4: Data Splitting
    Write "Phase 4: Splitting Data"

    Get columns for features from data save to X
    Get column "Target" from data save to y
```



```
    Split data X, y into train and test with ratio 0.8 save to
X_train, X_test, y_train, y_test

    Write "Training set size: "
    Get shape of X_train save to train_shape
    Write train_shape

    # Phase 5: Model Training
    Write "Phase 5: Training Multiple Models"

    Create list named models
    Create list named model_names
    Create list named scores

    # Train multiple models
    Repeat for every algorithm in ["LinearRegression",
"RandomForest", "GradientBoosting"]
        Write "Training "
        Write algorithm
        Write "..."

        Create predictor model using algorithm
        Train model using X_train to predict y_train

        Predict using model with inputs X_test save to predictions
        Calculate R2 score between y_test and predictions save to
score
        Add model to models
        Add algorithm to model_names
        Add score to scores
```



```
        Write "  Score: "
        Write score
    End repeat

    # Phase 6: Model Selection
    Write "Phase 6: Selecting Best Model"

    Get maximum from scores save to best_score
    Get index of best_score in scores save to best_index
    Get item best_index from models save to best_model
    Get item best_index from model_names save to best_model_name

    Write "Best Model: "
    Write best_model_name
    Write " with R² = "
    Write best_score

    # Phase 7: Hyperparameter Tuning
    Write "Phase 7: Tuning Best Model"

    # (Tuning logic here)

    # Phase 8: Final Evaluation
    Write "Phase 8: Final Evaluation"

    Predict using best_model with inputs X_test save to
final_predictions
```



```
    Calculate R2 score between y_test and final_predictions save to
r2

    Calculate mean absolute error between y_test and
final_predictions save to mae

    Calculate root mean squared error between y_test and
final_predictions save to rmse


    Write "===== Final Model Performance ====="

    Write "R² Score: "

    Write r2

    Write "MAE: "

    Write mae

    Write "RMSE: "

    Write rmse


    # Phase 9: Visualization

    Write "Phase 9: Creating Visualizations"


    # Model comparison

    Visualize comparison as BarChart

        Set X to model_names

        Set Y to scores

        Set Title to "Model Performance Comparison"

    End visualization

    Save to "model_comparison.png"


    # Actual vs Predicted

    Visualize results as ScatterPlot

        Set X to y_test

        Set Y to final_predictions
```



```
        Add diagonal line
        Set Title to "Actual vs Predicted Values"
        Set X label to "Actual"
        Set Y label to "Predicted"
    End visualization
    Save to "predictions.png"

    # Residual plot
    Calculate y_test minus final_predictions save to residuals
    Visualize residuals as ScatterPlot
        Set X to final_predictions
        Set Y to residuals
        Add horizontal line at 0
        Set Title to "Residual Plot"
    End visualization
    Save to "residuals.png"

    # Phase 10: Save Model
    Write "Phase 10: Saving Model"
    Save best_model to "final_model.pkl"
    Write "Model saved successfully"

    Write "============================================"
    Write "         PROJECT COMPLETE"
    Write "============================================"

    Return best_model
End function
```



```
# Run the complete project
Run CompleteMLProject with inputs "dataset.csv" save to final_model
```

16.21 Professional Insights

Machine learning in production environments requires:

1. Data Pipeline Management

- Automated data collection and validation
- Continuous monitoring of data quality
- Version control for datasets (DVC, Git LFS)
- Data lineage tracking

2. Model Monitoring

- Track prediction accuracy over time
- Detect data drift (input distribution changes)
- Detect concept drift (relationship changes)
- Automated alerts for performance degradation
- Retrain triggers

3. A/B Testing

- Compare new models against production baseline
- Gradual rollout (5% → 20% → 50% → 100%)
- Statistical significance testing
- Rollback capability

4. Explainability & Interpretability

- Feature importance analysis
- SHAP (SHapley Additive exPlanations) values

- LIME (Local Interpretable Model-agnostic Explanations)
- Model cards documenting intended use

5. Ethical Considerations

- Bias detection across demographic groups
- Fairness metrics (demographic parity, equal opportunity)
- Privacy protection (differential privacy, federated learning)
- Transparent documentation of limitations
- Regular audits

6. MLOps Best Practices

- Containerization (Docker)
- Orchestration (Kubernetes, Kubeflow)
- CI/CD pipelines for models
- Feature stores for consistency
- Model registries with versioning

16.22 Key Takeaways

1. **ML learns patterns from data automatically—no explicit rules needed**
2. **Quality data matters more than sophisticated algorithms (garbage in = garbage out)**
3. **Always split data: train/validation/test or use cross-validation**
4. **Choose evaluation metrics appropriate for your problem domain**
5. **Feature engineering often provides the biggest performance gains**
6. **Overfitting is the most common pitfall—simpler models often generalize better**
7. **Start simple, add complexity only when justified by performance**
8. **Document everything for reproducibility and collaboration**

9. **Production ML requires monitoring, testing, and maintenance**

10. **Consider ethical implications and potential biases in your models**

16.23 Exercises

Beginner Level

Create your first ML model:

- Load the iris dataset (built-in sklearn dataset)
- Split into train/test (80/20)
- Train a Decision Tree classifier
- Calculate and report accuracy
- Try at least 2 other algorithms
- Identify which performs best

Deliverables: Working code, accuracy comparison table

Intermediate Level

Build a complete classification pipeline:

- Load Titanic survival dataset (Kaggle)
- Perform exploratory data analysis (5+ visualizations)
- Handle missing values appropriately
- Engineer 3+ new features (e.g., family size, title from name)
- Encode categorical variables
- Train and compare 5 different algorithms
- Use 5-fold cross-validation
- Create confusion matrix for best model
- Identify and visualize most important features

- Achieve >80% accuracy

Deliverables: Jupyter notebook with analysis, visualizations, and model comparison

Advanced Level

Create an end-to-end ML system:

- Choose a real-world dataset (Kaggle, UCI, or scrape your own)
- Perform comprehensive EDA with at least 10 visualizations
- Document data quality issues and solutions
- Engineer 10+ meaningful features with justification
- Handle class imbalance if present (SMOTE, class weights)
- Compare 6+ algorithms with hyperparameter tuning (GridSearch or RandomSearch)
- Implement ensemble methods (voting, stacking)
- Create detailed performance report with:
  - Precision/Recall/F1 for each class
  - ROC curves and AUC scores
  - Feature importance analysis
  - Error analysis (which examples are misclassified?)
- Save final model with versioning (v1.0, v1.1, etc.)
- Write API endpoint for predictions
- Create simple web interface (optional)
- Document entire pipeline in README

Deliverables: GitHub repository with complete pipeline, documentation, and deployment instructions

---

Chapter 17: Neural Networks & Deep Learning

Teaching Machines to Perceive and Understand

17.1 The Deep Learning Revolution

Traditional Programming:

```
Input → Hand-coded Rules → Output
```

Traditional Machine Learning:

```
Input → Hand-crafted Features → Simple Model → Output
```

Deep Learning:

```
Raw Input → Neural Network → Output
            (learns features automatically)
```

Example - Image Classification:

Traditional Approach:

```
Image → Extract edges (Sobel filter)
      → Extract corners (Harris detector)
```



```
      → Extract textures (Gabor filters)
      → Train SVM
      → Classification
```

Deep Learning Approach:

```
Image → Convolutional Neural Network → Classification
        (learns edges, corners, textures automatically)
```

17.2 The Artificial Neuron

Biological Inspiration:

```
Dendrites → Cell Body → Axon → Synapses
(inputs)    (process)   (output) (connections)
```

Artificial Neuron:

```
     x₁ ──w₁──┐
     x₂ ──w₂──┤
     x₃ ──w₃──┼──► Σ ──► f(·) ──► output
     x₄ ──w₄──┘
       + bias
```


Mathematical Formulation:

```
z = w₁x₁ + w₂x₂ + w₃x₃ + w₄x₄ + b
output = f(z)
```

Where:

- **x**: Inputs
- **w**: Weights (learned parameters)
- **b**: Bias term (learned parameter)
- **f**: Activation function (adds non-linearity)

17.3 Activation Functions

Why Non-linearity Matters:

Without activation functions, stacked layers collapse to single linear transformation.

ReLU (Rectified Linear Unit)

```
f(x) = max(0, x)
```

Graph:

```
     |    /
     |   /
     |  /
─────┼──────
     |
```


Properties:

- Most popular for hidden layers
- Fast computation
- Helps with vanishing gradient
- Can cause "dying ReLU" (neurons output 0 forever)

Variants:

- **Leaky ReLU**: f(x) = max(0.01x, x)
- **ELU**: Exponential Linear Unit
- **GELU**: Gaussian Error Linear Unit

Sigmoid

```
f(x) = 1 / (1 + e^(-x))

Graph:
   1 ┤     ────────
     │    /
 0.5 ┤   /
     │  /
   0 ┤─────
```

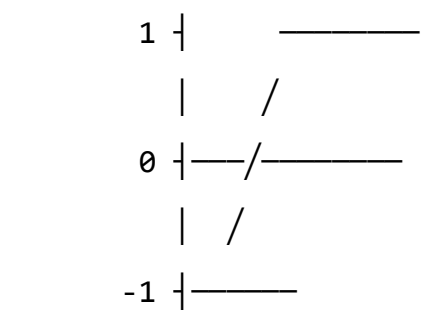
Properties:

- Squashes to [0, 1]
- Interpreted as probability
- Vanishing gradient problem
- Used for binary classification output

Tanh (Hyperbolic Tangent)

```
f(x) = (e^x - e^(-x)) / (e^x + e^(-x))
```

Graph:



Properties:

- Squashes to [-1, 1]
- Zero-centered (better than sigmoid)
- Still suffers from vanishing gradient

Softmax

```
f(x_i) = e^(x_i) / Σ_j e^(x_j)
```

Properties:

- Converts scores to probabilities
- Output sums to 1
- Used for multi-class classification
- Only in final layer

17.4 Building Your First Neural Network

```astra
Use python library tensorflow as tf
Use python library numpy as np

To do CreateAndTrainNetwork
    Write "=== Building Neural Network ==="

    # Define architecture
    Define NeuralNetwork named DigitClassifier
        Add layer Input with shape 784
        Add layer Dense with 128 neurons activation relu
        Add layer Dropout with rate 0.2
        Add layer Dense with 64 neurons activation relu
        Add layer Dropout with rate 0.2
        Add layer Dense with 10 neurons activation softmax
    End network

    # Display architecture
    Write "Network Architecture:"
    Display summary of DigitClassifier

    # Compile with optimizer and loss
    Compile DigitClassifier with
        Optimizer "adam" learning_rate 0.001
        Loss "sparse_categorical_crossentropy"
        Metrics ["accuracy"]
    End compile
```



```
    # Load and prepare MNIST data
    Write "Loading MNIST dataset..."
    Load dataset "mnist" save to train_images, train_labels,
test_images, test_labels

    # Normalize pixel values to [0, 1]
    Divide train_images by 255.0 save to train_images
    Divide test_images by 255.0 save to test_images

    # Flatten 28x28 images to 784-element vectors
    Reshape train_images to shape -1, 784 save to train_images
    Reshape test_images to shape -1, 784 save to test_images

    Write "Training set: "
    Get shape of train_images save to train_shape
    Write train_shape

    Write "Test set: "
    Get shape of test_images save to test_shape
    Write test_shape

    # Train the model
    Write "Starting training..."
    Train DigitClassifier using train_images, train_labels
        Set epochs to 10
        Set batch_size to 128
        Set validation_split to 0.2
        Set verbose to 1
```



```
    End training save to history

    # Evaluate on test set
    Write "Evaluating on test set..."
    Evaluate DigitClassifier using test_images, test_labels save to
test_loss, test_accuracy

    Write "===== Final Results ====="
    Write "Test Loss: "
    Write test_loss
    Write "Test Accuracy: "
    Write test_accuracy

    # Visualize training history
    Create figure with 1 row and 2 columns save to fig

    In subplot 1 of fig
        Visualize history as LineChart
            Get "accuracy" from history save to train_acc
            Get "val_accuracy" from history save to val_acc
            Set X to epochs
            Add series train_acc with label "Training"
            Add series val_acc with label "Validation"
            Set Title to "Model Accuracy"
            Set Y label to "Accuracy"
            Set X label to "Epoch"
        End visualization
    End subplot
```



```
    In subplot 2 of fig
        Visualize history as LineChart
            Get "loss" from history save to train_loss
            Get "val_loss" from history save to val_loss
            Set X to epochs
            Add series train_loss with label "Training"
            Add series val_loss with label "Validation"
            Set Title to "Model Loss"
            Set Y label to "Loss"
            Set X label to "Epoch"
        End visualization
    End subplot

    Save figure fig to "training_history.png"
    Write "Training history saved"

    # Make sample predictions
    Write "Sample Predictions:"
    Repeat 5 times using counter as i
        Get item i from test_images save to sample_image
        Get item i from test_labels save to true_label

        Reshape sample_image to 1, 784 save to sample_batch
        Predict using DigitClassifier with inputs sample_batch save
to prediction

        Get maximum index from prediction save to predicted_label

        Write "Image "
```



```
        Write i
        Write ": Predicted="
        Write predicted_label
        Write ", Actual="
        Write true_label
    End repeat

    # Save model
    Save DigitClassifier to "digit_classifier.h5"
    Write "Model saved successfully"

    Return DigitClassifier
End function

Run CreateAndTrainNetwork save to trained_model
```

Expected Output:

```
=== Building Neural Network ===
Network Architecture:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
dropout (Dropout)            (None, 128)               0
dense_1 (Dense)              (None, 64)                8256
dropout_1 (Dropout)          (None, 64)                0
dense_2 (Dense)              (None, 10)                650
=================================================================
```



```
Total params: 109,386

Trainable params: 109,386

Non-trainable params: 0

_________________________________________________________________


Loading MNIST dataset...

Training set: (60000, 784)

Test set: (10000, 784)


Starting training...

Epoch 1/10

375/375 [======] - 2s 5ms/step - loss: 0.3721 - accuracy: 0.8912 -
val_loss: 0.1876 - val_accuracy: 0.9448

Epoch 2/10

375/375 [======] - 2s 4ms/step - loss: 0.1689 - accuracy: 0.9503 -
val_loss: 0.1389 - val_accuracy: 0.9596

...

Epoch 10/10

375/375 [======] - 2s 4ms/step - loss: 0.0521 - accuracy: 0.9840 -
val_loss: 0.0982 - val_accuracy: 0.9726


Evaluating on test set...

===== Final Results =====

Test Loss: 0.0891

Test Accuracy: 0.9753


Sample Predictions:

Image 0: Predicted=7, Actual=7

Image 1: Predicted=2, Actual=2

Image 2: Predicted=1, Actual=1
```



```
Image 3: Predicted=0, Actual=0
Image 4: Predicted=4, Actual=4

Training history saved
Model saved successfully
```

17.5 Understanding Backpropagation

Forward Pass:

```
Input → Layer 1 → Layer 2 → ... → Output → Loss
```

Backward Pass (Gradient Flow):

```
Loss → ∂L/∂output → ∂L/∂Layer2 → ∂L/∂Layer1 → Update Weights
```

Weight Update:

```
w_new = w_old - learning_rate × gradient
```

Conceptual Example:

```astra
# Pseudocode for understanding backpropagation

To do TrainStep with inputs x, y_true
```



```
    # Forward pass
    Run forward pass with x save to y_pred
    Calculate loss between y_pred and y_true save to loss

    # Backward pass
    Calculate gradient of loss with respect to output save to grad_output

    # Propagate gradients backward through layers
    For each layer in reverse order
        Calculate gradient for layer using grad_output
        Update layer weights using gradient
        Pass gradient to previous layer
    End for

    Return loss
End function
```

17.6 Training Hyperparameters

Learning Rate

Controls step size during weight updates.

```astra
# Too high - diverges
Compile model with
    Optimizer "adam" learning_rate 1.0
End compile
```



```
# Too low - slow convergence
Compile model with
    Optimizer "adam" learning_rate 0.00001
End compile

# Good range
Compile model with
    Optimizer "adam" learning_rate 0.001  # Common default
End compile
```

Learning Rate Schedules:

```astra
# Decay over time
Create learning rate schedule
    Set initial_rate to 0.001
    Set decay_steps to 1000
    Set decay_rate to 0.96
End schedule save to lr_schedule

Compile model with
    Optimizer "adam" learning_rate lr_schedule
End compile
```

Batch Size

Number of samples processed before updating weights.


```astra
# Small batch (32-64)
# Pros: More frequent updates, less memory
# Cons: Noisy gradients, slower per epoch

# Medium batch (128-256)
# Good balance for most tasks

# Large batch (512-1024)
# Pros: Stable gradients, faster per epoch
# Cons: More memory, might converge to poor local minimum
```

Epochs

Complete passes through the entire dataset.

```astra
# Use early stopping to prevent overfitting
Create early stopping callback
    Set monitor to "val_loss"
    Set patience to 5
    Set restore_best_weights to True
End callback save to early_stop

Train model using X_train, y_train
    Set epochs to 100
    Set callbacks to [early_stop]
    Set validation_split to 0.2
End training
```

17.7 Regularization Techniques

Dropout

Randomly disable neurons during training.

```astra
Define NeuralNetwork named RegularizedNet
    Add layer Dense with 128 neurons activation relu
    Add layer Dropout with rate 0.5  # Drop 50% of neurons
    Add layer Dense with 64 neurons activation relu
    Add layer Dropout with rate 0.3  # Drop 30% of neurons
    Add layer Dense with 10 neurons activation softmax
End network
```

Effect:

- Prevents co-adaptation of neurons
- Acts as ensemble of networks
- Only active during training (automatically disabled during inference)

L2 Regularization (Weight Decay)

Penalizes large weights.

```astra
Use python library tensorflow.keras.regularizers as regularizers
```



```
Define NeuralNetwork named L2Net
    Add layer Dense with 128 neurons activation relu
        Set kernel_regularizer to L2 with lambda 0.01
    End layer
    Add layer Dense with 10 neurons activation softmax
End network
```

Batch Normalization

Normalizes layer inputs.

```astra
Define NeuralNetwork named BatchNormNet
    Add layer Dense with 128 neurons activation relu
    Add layer BatchNormalization
    Add layer Dense with 64 neurons activation relu
    Add layer BatchNormalization
    Add layer Dense with 10 neurons activation softmax
End network
```

Benefits:

- Allows higher learning rates
- Reduces sensitivity to initialization
- Acts as regularization

17.8 Convolutional Neural Networks (CNNs)

For Image Data:

```astra
Define NeuralNetwork named ImageClassifier
    # Input: 28x28x1 (grayscale image)
    Add layer Input with shape 28, 28, 1

    # Convolutional Block 1
    Add layer Conv2D with 32 filters size 3x3 activation relu padding same
    Add layer BatchNormalization
    Add layer Conv2D with 32 filters size 3x3 activation relu padding same
    Add layer BatchNormalization
    Add layer MaxPooling2D with pool_size 2x2
    Add layer Dropout with rate 0.25

    # Convolutional Block 2
    Add layer Conv2D with 64 filters size 3x3 activation relu padding same
    Add layer BatchNormalization
    Add layer Conv2D with 64 filters size 3x3 activation relu padding same
    Add layer BatchNormalization
    Add layer MaxPooling2D with pool_size 2x2
    Add layer Dropout with rate 0.25

    # Convolutional Block 3
```



```
    Add layer Conv2D with 128 filters size 3x3 activation relu
padding same
    Add layer BatchNormalization
    Add layer MaxPooling2D with pool_size 2x2
    Add layer Dropout with rate 0.25

    # Fully Connected Layers
    Add layer Flatten
    Add layer Dense with 256 neurons activation relu
    Add layer BatchNormalization
    Add layer Dropout with rate 0.5
    Add layer Dense with 10 neurons activation softmax
End network
```

CNN Architecture Visualization:

```
Input (28×28×1)
    │
    ▼
Conv2D (32 filters) → 28×28×32
    │
    ▼
MaxPool (2×2) → 14×14×32
    │
    ▼
Conv2D (64 filters) → 14×14×64
    │
    ▼
MaxPool (2×2) → 7×7×64
```



```
    │
    ▼
Conv2D (128 filters) → 7×7×128
    │
    ▼
MaxPool (2×2) → 3×3×128
    │
    ▼
Flatten → 1152
    │
    ▼
Dense (256) → 256
    │
    ▼
Output (10) → 10
```

17.9 CNN Operations Explained

Convolution Operation

Detects local patterns.

```
Input (5×5):          Filter (3×3):        Output:
[1 2 3 4 5]          [1 0 -1]            [conv
 [2 3 4 5 6]          [2 0 -2]             result]
 [3 4 5 6 7]    ⊛     [1 0 -1]        →
 [4 5 6 7 8]
```



```
 [5 6 7 8 9]]
```

Sliding Window:

1. Place filter on top-left of input
2. Multiply element-wise and sum
3. Slide filter right
4. Repeat until entire input covered

Pooling Operation

Reduces spatial dimensions.

```
Max Pooling (2×2):

Input:               Output:
[1  3  2  4]        [3  4]
[5  6  7  8]    →   [9  10]
[9  2  10 3]
[1  5  3  7]
```

Takes maximum value in each 2×2 region.

Benefits:

- Reduces parameters
- Translation invariance
- Computational efficiency

17.10 Data Augmentation

Artificially expand training data:

```astra
Use python library tensorflow.keras.preprocessing.image as image_processing

Create image data generator
    Set rotation_range to 15
    Set width_shift_range to 0.1
    Set height_shift_range to 0.1
    Set horizontal_flip to True
    Set zoom_range to 0.1
    Set shear_range to 0.1
    Set fill_mode to "nearest"
End generator save to data_gen

# Train with augmented data
Train model using data_gen with data X_train, y_train
    Set epochs to 50
    Set steps_per_epoch to 100
    Set validation_data to X_val, y_val
End training
```

Augmentation Examples:

- Original image
- Rotated 10°
- Shifted horizontally

- Flipped horizontally
- Zoomed 20%
- Sheared slightly

17.11 Transfer Learning

Leverage pre-trained models:

```astra
Use python library tensorflow.keras.applications as applications

To do BuildTransferModel
    # Load pre-trained model (trained on ImageNet)
    Load pretrained model "ResNet50" with parameters
        Set weights to "imagenet"
        Set include_top to False  # Remove classification layer
        Set input_shape to 224, 224, 3
    End parameters save to base_model

    # Freeze pre-trained layers
    Set trainable of base_model to False

    # Add custom classification layers
    Define NeuralNetwork named CustomClassifier
        Add base_model as layer
        Add layer GlobalAveragePooling2D
        Add layer Dense with 256 neurons activation relu
        Add layer Dropout with rate 0.5
        Add layer Dense with num_classes neurons activation softmax
```



```
End network

# Compile
Compile CustomClassifier with
    Optimizer "adam" learning_rate 0.001
    Loss "categorical_crossentropy"
    Metrics ["accuracy"]
End compile

# Train only new layers
Write "Training custom layers..."
Train CustomClassifier using train_data
    Set epochs to 10
    Set validation_data to val_data
End training

# Fine-tuning: Unfreeze some base layers
Write "Fine-tuning..."
Unfreeze last 20 layers of base_model

# Re-compile with lower learning rate
Compile CustomClassifier with
    Optimizer "adam" learning_rate 0.0001
    Loss "categorical_crossentropy"
    Metrics ["accuracy"]
End compile

# Train entire network
Train CustomClassifier using train_data
```



```
        Set epochs to 10
        Set validation_data to val_data
    End training

    Return CustomClassifier
End function
```

Benefits:
- Train with fewer examples
- Faster convergence
- Better accuracy
- Leverages knowledge from millions of images

17.12 Recurrent Neural Networks (RNNs)

For Sequential Data:

```astra
Define NeuralNetwork named TextClassifier
    # Input: sequences of word indices
    Add layer Input with shape sequence_length

    # Embedding layer (convert indices to dense vectors)
    Add layer Embedding with parameters
        Set input_dim to vocab_size
        Set output_dim to 128
        Set mask_zero to True
    End parameters
```



```
    # Recurrent layers
    Add layer LSTM with 128 units return_sequences True
    Add layer Dropout with rate 0.3
    Add layer LSTM with 64 units
    Add layer Dropout with rate 0.3

    # Output
    Add layer Dense with 1 neuron activation sigmoid
End network
```

RNN Architecture:

```
Input Sequence:    x₁   →   x₂   →   x₃   →   x₄
                   |        |        |        |
Hidden States:    [h₁] →  [h₂] →  [h₃] →  [h₄]
                   |        |        |        |
Outputs:          y₁       y₂       y₃       y₄
```

LSTM Cell (Long Short-Term Memory):

Solves vanishing gradient problem in standard RNNs.

Components:

- **Forget Gate**: What to discard from memory
- **Input Gate**: What new information to store
- **Output Gate**: What to output based on memory

17.13 Natural Language Processing

Sentiment Analysis

```astra
Use python library tensorflow as tf
Use python library tensorflow.keras.preprocessing.text as text_processing

To do TrainSentimentModel with inputs texts, labels
    # Tokenization
    Create tokenizer with max_words 10000
    Fit tokenizer on texts

    # Convert texts to sequences
    Convert texts to sequences using tokenizer save to sequences

    # Pad to same length
    Pad sequences to length 200 save to padded_sequences

    # Build model
    Define NeuralNetwork named SentimentAnalyzer
        Add layer Input with shape 200
        Add layer Embedding with 10000 input_dim 128 output_dim
        Add layer SpatialDropout1D with rate 0.2
        Add layer LSTM with 64 units dropout 0.2 recurrent_dropout 0.2
        Add layer Dense with 1 neuron activation sigmoid
    End network
```



```
    # Compile
    Compile SentimentAnalyzer with
        Optimizer "adam"
        Loss "binary_crossentropy"
        Metrics ["accuracy"]
    End compile

    # Train
    Train SentimentAnalyzer using padded_sequences, labels
        Set epochs to 5
        Set batch_size to 64
        Set validation_split to 0.2
    End training

    Return SentimentAnalyzer, tokenizer
End function

# Use the model
Run TrainSentimentModel with inputs movie_reviews, sentiments save
to model, tok

To do PredictSentiment with inputs text, model, tokenizer
    Convert text to sequence using tokenizer save to seq
    Pad seq to length 200 save to padded
    Predict using model with inputs padded save to score

    If score is greater than 0.5 then
        Return "Positive"
```



```
    Else
        Return "Negative"
    End check
End function

Set review to "This movie was absolutely fantastic!"
Run PredictSentiment with inputs review, model, tok save to
sentiment
Write sentiment  # Output: Positive
```

17.14 Using Pre-trained Language Models

Transformers (BERT, GPT, etc.):

```astra
Use python library transformers

# Sentiment Analysis with BERT
Load pretrained model "bert-base-uncased" for "sentiment-analysis"
save to classifier

To do AnalyzeSentiment with inputs text
    Predict using classifier with inputs text save to result
    Return result
End function

Run AnalyzeSentiment with inputs "I love this product!" save to
result
```



```
Write result
# Output: [{'label': 'POSITIVE', 'score': 0.9998}]

# Text Generation with GPT-2
Load pretrained model "gpt2" save to generator
Load tokenizer "gpt2" save to tokenizer

To do GenerateText with inputs prompt, max_length
    Encode prompt using tokenizer save to input_ids

    Generate from generator
        Set input_ids to input_ids
        Set max_length to max_length
        Set temperature to 0.7
        Set top_k to 50
        Set top_p to 0.95
        Set num_return_sequences to 1
        Set pad_token_id to tokenizer.eos_token_id
    End generation save to output_ids

    Decode output_ids using tokenizer save to generated_text
    Return generated_text
End function

Set prompt to "In a world where artificial intelligence"
Run GenerateText with inputs prompt, 100 save to story
Write story
```

17.15 Computer Vision Applications

Image Classification

```astra
Use python library tensorflow.keras.applications as apps

# Load pre-trained MobileNetV2
Load pretrained model "MobileNetV2" with weights "imagenet" save to model

To do ClassifyImage with inputs image_path
    # Load and preprocess
    Load image image_path with target_size 224, 224 save to img
    Convert img to array save to img_array
    Expand dimensions of img_array save to img_batch
    Preprocess img_batch for "MobileNetV2" save to processed

    # Predict
    Predict using model with inputs processed save to predictions

    # Decode top 5 predictions
    Decode predictions for "imagenet" top 5 save to results

    Write "Predictions:"
    Repeat for every result in results
        Get "class" from result save to class_name
        Get "probability" from result save to prob
```



```
        Write "  "
        Write class_name
        Write ": "
        Multiply prob by 100 save to percent
        Write percent
        Write "%"
    End repeat

    Return results
End function

Run ClassifyImage with inputs "dog.jpg"
```

Output:

```
Predictions:
  golden_retriever: 89.2%
  labrador_retriever: 7.3%
  cocker_spaniel: 1.8%

Object Detection
Use python library tensorflow_hub as hub

# Load object detection model
Load model "https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2" save to
detector

To do DetectObjects with inputs image_path
    Load image image_path save to img
```



```
    Resize img to 640, 640 save to img
    Convert img to tensor save to input_tensor

    # Detect
    Run detector with inputs input_tensor save to result

    # Extract detections
    Get "detection_boxes" from result save to boxes
    Get "detection_classes" from result save to classes
    Get "detection_scores" from result save to scores

    Write "Detected Objects:"
    Repeat for every i in range 0 to length of scores
        Get item i from scores save to score

        If score is greater than 0.5 then
            Get item i from classes save to class_id
            Get item i from boxes save to box

            Get class name for class_id save to class_name

            Write "  "
            Write class_name
            Write " ("
            Multiply score by 100 save to confidence
            Write confidence
            Write "%) at "
            Write box
        End check
```



```
    End repeat

    # Draw boxes on image
    Draw boxes on img from boxes, classes, scores save to annotated
    Save annotated to "detected.jpg"

    Return boxes, classes, scores
End function

Run DetectObjects with inputs "street_scene.jpg"
```

17.16 Generative Models

```
Variational Autoencoders (VAEs)
Define NeuralNetwork named VAE_Encoder
    Add layer Input with shape 28, 28, 1
    Add layer Conv2D with 32 filters size 3x3 activation relu
strides 2 padding same
    Add layer Conv2D with 64 filters size 3x3 activation relu
strides 2 padding same
    Add layer Flatten
    Add layer Dense with 16 neurons  # z_mean
    Add layer Dense with 16 neurons  # z_log_var
End network

Define NeuralNetwork named VAE_Decoder
    Add layer Input with shape 16
    Add layer Dense with 7*7*64 neurons activation relu
    Add layer Reshape to shape 7, 7, 64
```



```
    Add layer Conv2DTranspose with 64 filters size 3x3 activation
relu strides 2 padding same

    Add layer Conv2DTranspose with 32 filters size 3x3 activation
relu strides 2 padding same

    Add layer Conv2DTranspose with 1 filter size 3x3 activation
sigmoid padding same

End network


# Train VAE to learn latent representation

# Can then generate new images by sampling from latent space
```

Generative Adversarial Networks (GANs)

```
# Generator: Creates fake images

Define NeuralNetwork named Generator

    Add layer Input with shape 100  # Random noise

    Add layer Dense with 7*7*256 neurons

    Add layer BatchNormalization

    Add layer LeakyReLU with alpha 0.2

    Add layer Reshape to shape 7, 7, 256


    Add layer Conv2DTranspose with 128 filters size 5x5 strides 2
padding same

    Add layer BatchNormalization

    Add layer LeakyReLU with alpha 0.2


    Add layer Conv2DTranspose with 64 filters size 5x5 strides 2
padding same

    Add layer BatchNormalization

    Add layer LeakyReLU with alpha 0.2


    Add layer Conv2D with 1 filter size 7x7 activation tanh padding
same
```



```
End network


# Discriminator: Real or Fake?
Define NeuralNetwork named Discriminator
    Add layer Input with shape 28, 28, 1
    Add layer Conv2D with 64 filters size 5x5 strides 2 padding same
    Add layer LeakyReLU with alpha 0.2
    Add layer Dropout with rate 0.3

    Add layer Conv2D with 128 filters size 5x5 strides 2 padding
same
    Add layer LeakyReLU with alpha 0.2
    Add layer Dropout with rate 0.3

    Add layer Flatten
    Add layer Dense with 1 neuron activation sigmoid
End network


# Training loop (adversarial process)
To do TrainGAN with inputs epochs, batch_size
    Repeat epochs times using counter as epoch
        # Train discriminator
        Sample random noise save to noise
        Generate fake images using Generator with inputs noise save
to fake_images
        Get real images from dataset save to real_images

        # Label: 1 for real, 0 for fake
        Train Discriminator on real_images with labels 1
```



```
        Train Discriminator on fake_images with labels 0

        # Train generator (try to fool discriminator)
        Sample random noise save to noise
        Train Generator to make Discriminator output 1

        If epoch modulo 100 is 0 then
            Write "Epoch "
            Write epoch
            Write " complete"
            Save sample generated images
        End check
    End repeat
End function
```

17.17 Model Evaluation and Debugging

Common Issues:

1. Loss Not Decreasing

Learning rate too high or too low

Wrong loss function

Data not normalized

Vanishing/exploding gradients

2. Overfitting

Training loss decreases, validation loss increases

Solutions: Dropout, regularization, more data, data augmentation

3. Underfitting

Both losses high and not improving

Solutions: Deeper network, train longer, remove regularization

4. NaN Loss

Exploding gradients

Solutions: Lower learning rate, gradient clipping, batch normalization

Debugging Checklist:

```
To do DebugModel with inputs model, X_train, y_train
    Write "===== Model Debugging ====="

    # Check data
    Write "Data shapes:"
    Get shape of X_train save to x_shape
    Get shape of y_train save to y_shape
    Write "  X: "
    Write x_shape
    Write "  y: "
    Write y_shape

    # Check data range
    Write "Data statistics:"
    Get minimum of X_train save to x_min
    Get maximum of X_train save to x_max
    Get mean of X_train save to x_mean
    Write "  Min: "
    Write x_min
    Write "  Max: "
```



```
    Write x_max
    Write "  Mean: "
    Write x_mean

    # Check for NaN or Inf
    Count NaN in X_train save to nan_count
    If nan_count is greater than 0 then
        Write "WARNING: "
        Write nan_count
        Write " NaN values found in data!"
    End check

    # Model architecture
    Write "Model Summary:"
    Display summary of model

    # Count parameters
    Get total parameters of model save to total_params
    Get trainable parameters of model save to trainable_params
    Write "Total parameters: "
    Write total_params
    Write "Trainable parameters: "
    Write trainable_params

    # Test forward pass
    Write "Testing forward pass..."
    Get first 10 samples from X_train save to sample_batch
    Attempt
```



```
        Predict using model with inputs sample_batch save to
test_output
        Write "Forward pass successful"
        Write "Output shape: "
        Get shape of test_output save to output_shape
        Write output_shape
    Recover
        Write "ERROR in forward pass: "
        Write error_message
    End attempt

    Write "===== Debugging Complete ====="
End function
```

17.18 Model Optimization

```
Reduce Model Size:
Use python library tensorflow as tf

# Quantization (INT8)
To do QuantizeModel with inputs model_path
    Load model from model_path save to model

    Convert model to TFLite
        Set optimization to "DEFAULT"
        Set inference_type to "INT8"
    End conversion save to quantized_model

    Get size of model save to original_size
```



```
    Get size of quantized_model save to quantized_size

    Write "Original size: "
    Divide original_size by 1024 * 1024 save to mb
    Write mb
    Write " MB"

    Write "Quantized size: "
    Divide quantized_size by 1024 * 1024 save to qmb
    Write qmb
    Write " MB"

    Write "Compression ratio: "
    Divide original_size by quantized_size save to ratio
    Write ratio
    Write "x"

    Save quantized_model to "quantized_model.tflite"
End function

# Pruning
To do PruneModel with inputs model, sparsity
    Use python library tensorflow_model_optimization as tfmot

    Define pruning parameters
        Set pruning_schedule to "PolynomialDecay"
        Set initial_sparsity to 0.0
        Set final_sparsity to sparsity
        Set begin_step to 0
```



```
        Set end_step to 1000
    End parameters

    Apply pruning to model with parameters save to pruned_model

    Return pruned_model
End function
```

17.19 Production Deployment

```
Serving Models:
Use python library tensorflow as tf

# Save model for production
Save model to "production_model" format "SavedModel"

# Load and serve
Load model from "production_model" save to loaded_model

To do PredictionAPI with inputs image_bytes
    # Decode image
    Decode image_bytes save to img

    # Preprocess
    Resize img to 224, 224 save to img
    Convert img to array save to img_array
    Expand dimensions of img_array save to batch
    Preprocess batch save to processed
```



```
    # Predict
    Predict using loaded_model with inputs processed save to
prediction

    # Get top class
    Get maximum index from prediction save to class_id
    Get maximum from prediction save to confidence

    Create response dictionary
        Set "class_id" to class_id
        Set "confidence" to confidence
        Set "timestamp" to current_time
    End dictionary

    Return response
End function

# REST API (Flask example - conceptual)
Create API app

On route "/predict" method POST
    Get file from request save to file
    Read file bytes save to img_bytes

    Run PredictionAPI with inputs img_bytes save to result
    Return JSON result
End route
Start API app on port 5000
```


17.20 Ethical Considerations

Key Concerns:

-Bias in Training Data

Face recognition failing on certain demographics

Language models perpetuating stereotypes

Recommendation systems creating filter bubbles

-Privacy

Models memorizing training data

Face recognition without consent

Data aggregation risks

-Transparency

Black-box decisions affecting lives

Right to explanation

Audit trails

-Safety

Adversarial examples fooling models

Deepfakes and misinformation

Autonomous system failures

Best Practices:

```
To do EthicalMLChecklist with inputs model, dataset
    Write "===== Ethical ML Review ====="

    # Dataset diversity
```



```
    Analyze demographic distribution in dataset
    Check for balanced representation
    Document data sources and collection methods

    # Fairness testing
    Test model performance across demographic groups
    Calculate disparity metrics
    Document any disparities

    # Privacy
    Check for potential data leakage
    Implement differential privacy if needed
    Document data retention policies

    # Transparency
    Generate model card documenting:
        - Intended use
        - Training data
        - Performance metrics
        - Known limitations
        - Ethical considerations

    # Safety
    Test adversarial robustness
    Implement input validation
    Set up monitoring for drift

    Write "===== Review Complete ====="
End function
```

17.21 Key Takeaways

1. **Neural networks automatically learn features from raw data.**
2. **Deep learning requires substantial data and compute.**
3. **CNNs excel at images, RNNs at sequences, Transformers at language.**
4. **Transfer learning enables training with fewer examples.**
5. **Regularization (dropout, batch norm) prevents overfitting.**
6. **Hyperparameter tuning significantly impacts performance.**
7. **Pre-trained models (BERT, GPT, ResNet) provide massive head start.**
8. **Production deployment requires optimization and monitoring.**
9. **Ethical considerations must be addressed proactively.**
10. **Deep learning is powerful but not always necessary.**

17.22 Exercises

Beginner Level

Build your first neural network:

- Load MNIST or Fashion-MNIST dataset
- Create 3-layer fully connected network
- Train for 10 epochs
- Achieve >90% accuracy
- Plot training history
- Make predictions on test images
- Visualize 10 correct and 10 incorrect predictions

Intermediate Level

Create a CNN for image classification:

- Use CIFAR-10 dataset (10 classes, 32x32 color images)
- Build CNN with at least 3 convolutional blocks
- Implement data augmentation
- Use batch normalization and dropout
- Train until convergence
- Create confusion matrix
- Visualize learned filters in first layer
- Achieve >75% test accuracy
- Save and reload model to make new predictions

Advanced Level

Implement transfer learning system:

- Choose dataset (cats/dogs, medical images, custom)
- Use pre-trained model (ResNet50, EfficientNet, or VGG)
- Freeze base layers, train custom top layers
- Implement fine-tuning
- Compare performance with/without transfer learning
- Create comprehensive evaluation report:
- Per-class metrics
- Confusion matrix
- ROC curves
- Error analysis with examples
- Optimize model (quantization or pruning)
- Deploy as REST API with Flask/FastAPI

Write model card documenting ethical considerations

Chapter 18: Trees & Hierarchical Data

Organizing Information in Levels

18.1 Why Trees Matter

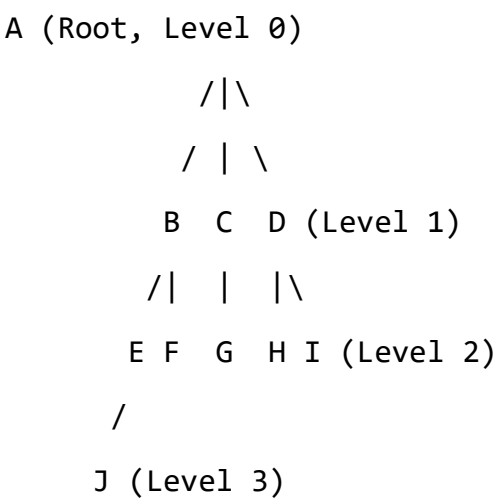
Trees are everywhere in computing:

- File systems (folders within folders)
- Organization charts (CEO → VPs → Managers → Employees)
- HTML DOM (html → body → div → p)
- XML/JSON structures
- Database indexes (B-trees)
- Decision processes
- Artificial intelligence (decision trees, game trees)
- Compilation (abstract syntax trees)

Key Properties:

- Hierarchical structure
- Single root node
- Parent-child relationships
- No cycles (paths don't loop back)

18.2 Tree Terminology



Essential Terms:

Root: A (top node, no parent)

Internal Nodes: A, B, C, D, E (have children)

Leaves: F, G, H, I, J (no children)

Parent: B is parent of E and F

Children: E and F are children of B

Siblings: B, C, D are siblings

Ancestors of J: E, B, A

Descendants of B: E, F, J

Depth of Node: Distance from root (J has depth 3)

Height of Tree: Maximum depth (height = 3)

Subtree: Any node and all its descendants

18.3 Creating Trees in Astra

```
# Create company organization chart
Create tree named CompanyOrg with root "CEO"

# Level 1: C-suite
Add child "CTO" to "CEO" in CompanyOrg
Add child "CFO" to "CEO" in CompanyOrg
Add child "CMO" to "CEO" in CompanyOrg
Add child "COO" to "CEO" in CompanyOrg

# Level 2: Engineering under CTO
Add child "VP Engineering" to "CTO" in CompanyOrg
Add child "VP Product" to "CTO" in CompanyOrg

# Level 3: Engineering teams
Add child "Backend Lead" to "VP Engineering" in CompanyOrg
```



```
Add child "Frontend Lead" to "VP Engineering" in CompanyOrg
Add child "DevOps Lead" to "VP Engineering" in CompanyOrg

# Level 4: Individual engineers
Add child "Senior Backend Dev 1" to "Backend Lead" in CompanyOrg
Add child "Senior Backend Dev 2" to "Backend Lead" in CompanyOrg
Add child "Junior Backend Dev" to "Backend Lead" in CompanyOrg

# Display structure
Display tree CompanyOrg
```

Output:

```
CEO
├── CTO
│   ├── VP Engineering
│   │   ├── Backend Lead
│   │   │   ├── Senior Backend Dev 1
│   │   │   ├── Senior Backend Dev 2
│   │   │   └── Junior Backend Dev
│   │   ├── Frontend Lead
│   │   └── DevOps Lead
│   └── VP Product
├── CFO
├── CMO
└── COO
```

18.4 Tree Traversal Algorithms

Three primary depth-first strategies:

1.Pre-Order (Root → Left → Right)

-Visit parent before children.


```
-Traverse CompanyOrg pre-order save to nodes

Write "Pre-order traversal: "
Repeat for every node in nodes
    Write node
End repeat
```

Output:

```
CEO
CTO
VP Engineering
Backend Lead
Senior Backend Dev 1
Senior Backend Dev 2
Junior Backend Dev
Frontend Lead
DevOps Lead
VP Product
CFO
CMO
COO
```

Use Cases:

-Copy a tree structure

-Serialize tree to file

-Print directory structure with indentation

-Expression tree evaluation (prefix notation)

2.In-Order (Left → Root → Right)

-Visit left subtree, then root, then right subtree.

-Traverse BinarySearchTree in-order save to nodes

Output (for BST): Sorted sequence

Use Cases:

-Get sorted output from Binary Search Tree

-Expression trees (infix notation)

Note: Only meaningful for binary trees.

3.Post-Order (Left → Right → Root)

-Visit children before parent.

-Traverse FileSystem post-order save to nodes

Use Cases:

-Delete tree (delete children before parent)

-Calculate directory sizes (need child sizes first)

-Expression trees (postfix notation)

-Dependency resolution

-Level-Order (Breadth-First)

-Visit all nodes at each level before deeper levels.

-Traverse CompanyOrg level-order save to nodes

```
Write "Level-order traversal: "
Repeat for every node in nodes
    Write node
End repeat
```


Output:

```
CEO
CTO, CFO, CMO, COO
VP Engineering, VP Product
Backend Lead, Frontend Lead, DevOps Lead
Senior Backend Dev 1, Senior Backend Dev 2, Junior Backend Dev
```

Use Cases:

-Find shortest path in unweighted tree

-Level-wise processing

-Print tree level by level

-Find nodes at specific depth

18.5 Binary Trees

Binary Tree: Each node has at most 2 children (left and right).

```
Create binary tree named ExpressionTree with root "+"

# Build expression: (3 + 5) * (2 - 1)
Add left child "*" to "+" in ExpressionTree
Add right child "/" to "+" in ExpressionTree

Add left child "3" to first "*" in ExpressionTree
Add right child "5" to first "*" in ExpressionTree

Add left child "2" to "/" in ExpressionTree
Add right child "1" to "/" in ExpressionTree
```


Tree Structure:

```
         +
        / \
       *   /
      / \ / \
     3  5 2  1
```

Evaluate Expression:

```
To do EvaluateExpressionTree with inputs node
    If node is leaf then
        Convert node value to number save to result
        Return result
    End check

    # Recursively evaluate children
    Get left child of node save to left_node
    Get right child of node save to right_node

    Run EvaluateExpressionTree with inputs left_node save to
left_value
    Run EvaluateExpressionTree with inputs right_node save to
right_value

    # Apply operator
    Get value of node save to operator

    If operator is "+" then
        Add right_value to left_value save to result
    Else if operator is "-" then
```



```
        Subtract right_value from left_value save to result
    Else if operator is "*" then
        Multiply left_value by right_value save to result
    Else if operator is "/" then
        Divide left_value by right_value save to result
    End check

    Return result
End function

Get root of ExpressionTree save to root
Run EvaluateExpressionTree with inputs root save to answer
Write "Result":
Write answer  # Output: 8.5
```

18.6 Binary Search Trees (BST)

Property: For every node, left descendants < node < right descendants.

```
Create binary search tree named BST

# Insert values
Insert 50 into BST
Insert 30 into BST
Insert 70 into BST
Insert 20 into BST
Insert 40 into BST
Insert 60 into BST
Insert 80 into BST
```


Resulting Structure:

```
        50
       /  \
     30    70
    /  \  /  \
   20  40 60  80
```

Searching:

```
Search BST for 40 save to found

If found is not Nothing then
    Write "Found: 40"
Else
    Write "Not found"
End check
```

Search Process:

Start at 50: 40 < 50, go left

At 30: 40 > 30, go right

At 40: Found!

Time Complexity:

Balanced tree: O(log n)

Worst case (skewed): O(n)

18.7 Tree Operations

Finding Height

```
To do FindHeight with inputs node
```



```
    If node is Nothing then
        Return 0
    End check

    Get left child of node save to left
    Get right child of node save to right

    Run FindHeight with inputs left save to left_height
    Run FindHeight with inputs right save to right_height

    If left_height is greater than right_height then
        Add 1 to left_height save to height
    Else
        Add 1 to right_height save to height
    End check

    Return height
End function
```

Counting Nodes

```
To do CountNodes with inputs node
    If node is Nothing then
        Return 0
    End check

    Get left child of node save to left
    Get right child of node save to right

    Run CountNodes with inputs left save to left_count
```



```
    Run CountNodes with inputs right save to right_count

    Add left_count to right_count save to total
    Add 1 to total save to total

    Return total
End function
```

Finding Maximum Value

```
To do FindMax with inputs node
    If node is Nothing then
        Return negative infinity
    End check

    Get value of node save to node_val
    Get left child of node save to left
    Get right child of node save to right

    Run FindMax with inputs left save to left_max
    Run FindMax with inputs right save to right_max

    Set max to node_val
    If left_max is greater than max then
        Set max to left_max
    End check
    If right_max is greater than max then
        Set max to right_max
    End check
```



```
    Return max
End function
```

18.8 Practical Applications

File System Navigation

```
Create tree named FileSystem with root "/"

Add child "home" to "/" in FileSystem
Add child "usr" to "/" in FileSystem
Add child "var" to "/" in FileSystem

Add child "user1" to "home" in FileSystem
Add child "user2" to "home" in FileSystem

Add child "documents" to "user1" in FileSystem
Add child "downloads" to "user1" in FileSystem

Add child "report.pdf" to "documents" in FileSystem
Add child "data.csv" to "documents" in FileSystem

To do CalculateDirectorySize with inputs path
    Get node at path in FileSystem save to node

    If node is file then
        Get size of node save to size
        Return size
    End check
```



```
    # It's a directory, sum children
    Set total_size to 0
    Get children of node save to children

    Repeat for every child in children
        Run CalculateDirectorySize with inputs child save to
child_size
        Add child_size to total_size
    End repeat

    Return total_size
End function

Run CalculateDirectorySize with inputs "/home/user1" save to size
Write "Total size: "
Write size
Write " bytes"
Decision Trees for AI
Create tree named DecisionTree with root "Outlook"

# Root: Outlook
Add child "Sunny" to "Outlook" in DecisionTree
Add child "Overcast" to "Outlook" in DecisionTree
Add child "Rain" to "Outlook" in DecisionTree

# Sunny branch: Check Humidity
Add child "Humidity" to "Sunny" in DecisionTree
Add child "High" to "Humidity" in DecisionTree  # → Don't Play
Add child "Normal" to "Humidity" in DecisionTree  # → Play
```



```
# Rain branch: Check Wind
Add child "Wind" to "Rain" in DecisionTree
Add child "Strong" to "Wind" in DecisionTree  # → Don't Play
Add child "Weak" to "Wind" in DecisionTree  # → Play

# Overcast always Play
Add child "Play" to "Overcast" in DecisionTree

To do MakeDecision with inputs conditions
    Set current_node to root of DecisionTree

    Loop while current_node is not leaf
        Get value of current_node save to attribute
        Get attribute from conditions save to value

        # Navigate to child matching value
        Get child with value value from current_node save to
current_node
    End loop

    Get value of current_node save to decision
    Return decision
End function

Create dictionary named today_weather
Set "Outlook" in today_weather to "Sunny"
Set "Humidity" in today_weather to "Normal"
```



```
Run MakeDecision with inputs today_weather save to should_play
Write "Decision: "
Write should_play  # Output: Play
```

18.9 Balanced Trees

Problem: Skewed BST degrades to O(n) search.

Solution: Self-balancing trees.

AVL Tree: Maintains height difference ≤ 1 between left and right subtrees.

Red-Black Tree: Uses coloring rules to maintain balance.

```
# Astra provides balanced BST automatically
Create balanced binary search tree named BalancedBST

# These insertions automatically maintain balance
Repeat 100 times using counter as i
    Insert i into BalancedBST
End repeat

# Still O(log n) search
Search BalancedBST for 42 save to result
```

18.10 Professional Insights

Trees are fundamental to:

1. Databases

-B-trees for disk-based indexing

-B+ trees for range queries

-R-trees for spatial data

2. Compilers

-Abstract Syntax Trees (AST) represent code structure

-Parse trees for syntax analysis

3. Operating Systems

-File system hierarchies

-Process trees

-Memory management (segment trees)

4. AI & Games

-Decision trees for classification

-Game trees for minimax algorithm

-Monte Carlo Tree Search (MCTS)

5. Networking

-Routing tables (trie structures)

-DNS hierarchy

-Network topologies

18.11 Key Takeaways

1.**Trees organize data hierarchically with parent-child relationships.**

2.**Traversal order determines processing sequence (pre/in/post/level-order).**

3.**Binary Search Trees enable O(log n) search when balanced.**

4.**Recursive algorithms naturally solve tree problems.**

5.**Trees appear throughout computer science and real-world systems.**

6.**Understanding trees is essential for algorithms and data structures.**

18.12 Exercises

Beginner Level

Implement basic tree operations:

- Create tree representing your family (3 generations)
- Implement pre-order traversal manually
- Count total number of nodes
- Find height of tree
- Print all leaf nodes

Intermediate Level

Build a file system simulator:

- Create tree representing directory structure
- Implement commands: ls, cd, mkdir, rm
- Calculate total size of directory (sum of all files)
- Find all files matching extension (.pdf, .txt)
- Print full path to each file
- Implement search by filename

Advanced Level

Create decision tree classifier:

- Load dataset (iris, titanic, or custom)
- Implement ID3 or CART algorithm to build tree
- Use information gain or Gini impurity for splits
- Visualize resulting tree structure
- Make predictions on new data
- Calculate accuracy
- Compare with sklearn Decision Tree
- Implement pruning to prevent overfitting

Chapter 19: Graphs & Networks

Modeling Connections and Relationships

19.1 What Is a Graph?

A graph consists of:

-Vertices (Nodes): The entities

-Edges: Connections between entities

Unlike trees:

-No single root

-Cycles are allowed

-Multiple paths between nodes

-Nodes can have any number of connections

Real-World Graphs:

-Social networks (people and friendships)

-Road networks (cities and roads)

-Internet (computers and connections)

-Molecular structures (atoms and bonds)

-Recommendation systems (users and products)

-Knowledge graphs (concepts and relationships)

-Flight routes (airports and flights)

19.2 Types of Graphs

Undirected Graph

Edges have no direction (symmetric relationships).

A ——— B

| |


```
|     |
C ——— D
```

Examples: Friendships, physical connections

Directed Graph (Digraph)

Edges have direction (asymmetric relationships).

```
A ——→ B
↑     |
|     ↓
C ←—— D
```

Examples: Twitter follows, web links, dependencies

Weighted Graph

Edges have associated values.

```
    5
A ————  B
|      /|
|8   /3 |7
|  /    |
C ————  D
    4
```

Examples: Road distances, costs, probabilities

19.3 Creating Graphs in Astra

```
# Create city road network
Create graph named CityMap

# Add cities (vertices)
```



```
Add vertex "New York" to CityMap

Add vertex "Boston" to CityMap

Add vertex "Philadelphia" to CityMap

Add vertex "Washington DC" to CityMap

Add vertex "Pittsburgh" to CityMap


# Add roads (weighted edges)

Add edge from "New York" to "Boston" with weight 215 in CityMap

Add edge from "New York" to "Philadelphia" with weight 95 in CityMap

Add edge from "Philadelphia" to "Washington DC" with weight 140 in
CityMap

Add edge from "Philadelphia" to "Pittsburgh" with weight 305 in
CityMap

Add edge from "Pittsburgh" to "Washington DC" with weight 245 in
CityMap


# Display graph

Display graph CityMap, 128)               100480

dropout (Dropout)           (None, 128)              0

dense_1 (Dense)             (None, 64)               8256

dropout_1 (Dropout)         (None, 64)               0

dense_2 (Dense)             (None, 10)               650

=================================================================

Total params: 109,386

Trainable params: 109,386

Non-trainable params: 0

_________________________________________________________________


Loading MNIST dataset...

Training set: (60000, 784)
```



```
Test set: (10000, 784)


Starting training...

Epoch 1/10

375/375 [======] - 2s 5ms/step - loss: 0.3721 - accuracy: 0.8912 -
val_loss: 0.1876 - val_accuracy: 0.9448

Epoch 2/10

375/375 [======] - 2s 4ms/step - loss: 0.1689 - accuracy: 0.9503 -
val_loss: 0.1389 - val_accuracy: 0.9596

...

Epoch 10/10

375/375 [======] - 2s 4ms/step - loss: 0.0521 - accuracy: 0.9840 -
val_loss: 0.0982 - val_accuracy: 0.9726


Evaluating on test set...

===== Final Results =====

Test Loss: 0.0891

Test Accuracy: 0.9753


Sample Predictions:

Image 0: Predicted=7, Actual=7

Image 1: Predicted=2, Actual=2

Image 2: Predicted=1, Actual=1

Image 3: Predicted=0, Actual=0

Image 4: Predicted=4, Actual=4


Training history saved

Model saved successfully
```

19.4 Graph Representation

Two Common Representations:

Adjacency Matrix

2D array where matrix[i][j] represents edge from node i to node j.

```
      A  B  C  D
A  [  0  5  8  0 ]
B  [  5  0  3  7 ]
C  [  8  3  0  4 ]
D  [  0  7  4  0 ]
```

Properties:

- O(1) edge lookup
- O(V²) space
- Good for dense graphs

Adjacency List

Each vertex stores list of connected vertices.

```
A → [B(5), C(8)]
B → [A(5), C(3), D(7)]
C → [A(8), B(3), D(4)]
D → [B(7), C(4)]
```


Properties:

- O(V + E) space
- Better for sparse graphs
- Faster iteration over neighbors

```astra
# Astra uses adjacency list by default
Get neighbors of "New York" in CityMap save to neighbors

Write "Cities connected to New York:"
Repeat for every neighbor in neighbors
    Write neighbor
End repeat
```

19.5 Graph Traversal Algorithms

Breadth-First Search (BFS)

Explore level by level (layer by layer).

```astra
Search CityMap from "New York" using BFS save to visited_order

Write "BFS Traversal Order:"
Repeat for every city in visited_order
    Write city
End repeat
```


BFS Process:

```
Queue: [New York]
Visit: New York
Queue: [Boston, Philadelphia]
Visit: Boston
Queue: [Philadelphia]
Visit: Philadelphia
Queue: [Washington DC, Pittsburgh]
...
```

Visual:

```
Level 0:  New York
Level 1:  Boston, Philadelphia
Level 2:  Washington DC, Pittsburgh
```

Use Cases:

- Shortest path in unweighted graph
- Social network connections (friends of friends)
- Web crawlers
- Network broadcasting

Depth-First Search (DFS)

Explore as deep as possible before backtracking.

```astra
Search CityMap from "New York" using DFS save to visited_order
```



```
Write "DFS Traversal Order:"
Repeat for every city in visited_order
    Write city
End repeat
```

DFS Process:

```
Stack: [New York]
Visit: New York
Stack: [Boston, Philadelphia]
Visit: Boston (go deep)
Stack: [Philadelphia, Washington DC]
Visit: Washington DC (continue deep)
...
```

Use Cases:

- Maze solving
- Topological sorting
- Detecting cycles
- Path finding

19.6 Shortest Path Algorithms

Dijkstra's Algorithm

Find shortest path in weighted graph with non-negative weights.

```astra
Find shortest path from "New York" to "Washington DC" in CityMap
save to path, distance
```



```
Write "Shortest path: "
Repeat for every city in path
    Write city
    Write " → "
End repeat

Write "Total distance: "
Write distance
Write " miles"
```

Output:

```
Shortest path: New York → Philadelphia → Washington DC
Total distance: 235 miles
```

Algorithm Visualization:

```
Start: New York (distance: 0)

Step 1: Update neighbors
  Boston: 215
  Philadelphia: 95

Step 2: Visit Philadelphia (95)
  Update Washington DC: 95 + 140 = 235
  Update Pittsburgh: 95 + 305 = 400
```



```
Step 3: Visit Boston (215)
  No improvements

Step 4: Visit Washington DC (235)
  Found destination!

Path: New York → Philadelphia → Washington DC
```

Complete Implementation:

```astra
To do DijkstraShortestPath with inputs graph, start, end
    # Initialize distances
    Create dictionary named distances
    Create dictionary named previous
    Create priority queue named unvisited

    Get all vertices from graph save to vertices
    Repeat for every vertex in vertices
        If vertex is start then
            Set vertex in distances to 0
        Else
            Set vertex in distances to infinity
        End check
        Set vertex in previous to Nothing
        Add vertex to unvisited with priority from distances
    End repeat

    # Main algorithm loop
```



```
    Loop while unvisited is not empty
        Get minimum from unvisited save to current
        Remove current from unvisited

        If current is end then
            Stop loop
        End check

        Get neighbors of current in graph save to neighbors
        Repeat for every neighbor in neighbors
            Get edge weight from current to neighbor save to weight
            Get current from distances save to current_dist
            Add weight to current_dist save to new_dist
            Get neighbor from distances save to neighbor_dist

            If new_dist is less than neighbor_dist then
                Set neighbor in distances to new_dist
                Set neighbor in previous to current
                Update priority of neighbor in unvisited to new_dist
            End check
        End repeat
    End loop

    # Reconstruct path
    Create list named path
    Set current to end

    Loop while current is not Nothing
        Add current to beginning of path
```



```
        Get current from previous save to current
    End loop

    Get end from distances save to total_distance

    Return path, total_distance
End function

Run DijkstraShortestPath with inputs CityMap, "New York",
"Washington DC" save to route, dist
Write "Optimal route: "
Write route
Write "Distance: "
Write dist
```

A* Algorithm

Optimized shortest path using heuristics.

```astra
To do AStarSearch with inputs graph, start, goal, heuristic
    Create priority queue named open_set
    Add start to open_set with priority 0

    Create dictionary named came_from
    Create dictionary named g_score
    Create dictionary named f_score

    Set start in g_score to 0
```



```
    Run heuristic with inputs start, goal save to h
    Set start in f_score to h

    Loop while open_set is not empty
        Get minimum from open_set save to current

        If current is goal then
            # Reconstruct path
            Create list named path
            Loop while current in came_from
                Add current to beginning of path
                Get current from came_from save to current
            End loop
            Add start to beginning of path
            Return path
        End check

        Remove current from open_set
        Get neighbors of current in graph save to neighbors

        Repeat for every neighbor in neighbors
            Get edge weight from current to neighbor save to
edge_weight
            Get current from g_score save to current_g
            Add edge_weight to current_g save to tentative_g

            Get neighbor from g_score with default infinity save to
neighbor_g
```



```
            If tentative_g is less than neighbor_g then
                Set neighbor in came_from to current
                Set neighbor in g_score to tentative_g
                Run heuristic with inputs neighbor, goal save to h
                Add tentative_g to h save to f
                Set neighbor in f_score to f

                If neighbor not in open_set then
                    Add neighbor to open_set with priority f
                End check
            End check
        End repeat
    End loop

    Return Nothing  # No path found
End function
```

19.7 Minimum Spanning Tree

Find subset of edges connecting all vertices with minimum total weight.

Kruskal's Algorithm

```astra
To do KruskalMST with inputs graph
    # Get all edges sorted by weight
    Get all edges from graph save to edges
    Sort edges by weight ascending save to sorted_edges
```



```
    # Initialize disjoint sets (union-find)
    Create disjoint set from vertices in graph save to uf

    Create list named mst_edges
    Set total_weight to 0

    Repeat for every edge in sorted_edges
        Get source from edge save to u
        Get destination from edge save to v
        Get weight from edge save to weight

        # Check if adding edge creates cycle
        If find u in uf is not equal to find v in uf then
            Add edge to mst_edges
            Add weight to total_weight
            Union u and v in uf
        End check

        # Stop when we have V-1 edges
        If length of mst_edges is equal to num_vertices minus 1 then
            Stop loop
        End check
    End repeat

    Return mst_edges, total_weight
End function

Run KruskalMST with inputs CityMap save to mst, total_cost
```



```
Write "Minimum Spanning Tree edges:"
Repeat for every edge in mst
    Write edge
End repeat
Write "Total cost: "
Write total_cost
```

Use Cases:

- Network design (minimize cable length)
- Cluster analysis
- Image segmentation
- Approximation algorithms

19.8 Cycle Detection

Check if graph contains cycles:

```astra
To do HasCycle with inputs graph
    Create set named visited
    Create set named recursion_stack

    Get all vertices from graph save to vertices

    Repeat for every vertex in vertices
        If vertex not in visited then
            Run DFSCycleDetect with inputs graph, vertex, visited, recursion_stack save to has_cycle
```



```
            If has_cycle is True then
                Return True
            End check
        End check
    End repeat

    Return False
End function

To do DFSCycleDetect with inputs graph, node, visited, rec_stack
    Add node to visited
    Add node to rec_stack

    Get neighbors of node in graph save to neighbors
    Repeat for every neighbor in neighbors
        If neighbor not in visited then
            Run DFSCycleDetect with inputs graph, neighbor, visited,
rec_stack save to found
            If found is True then
                Return True
            End check
        Else if neighbor in rec_stack then
            Return True  # Back edge = cycle
        End check
    End repeat

    Remove node from rec_stack
    Return False
End function
```



```
Run HasCycle with inputs CityMap save to has_cycle
If has_cycle then
    Write "Graph contains cycle"
Else
    Write "Graph is acyclic"
End check
```

19.9 Topological Sorting

Order vertices such that for every edge u→v, u comes before v.

```astra
To do TopologicalSort with inputs graph
    Create dictionary named in_degree
    Create queue named zero_degree
    Create list named result

    # Calculate in-degree for each vertex
    Get all vertices from graph save to vertices
    Repeat for every vertex in vertices
        Set vertex in in_degree to 0
    End repeat

    Repeat for every vertex in vertices
        Get neighbors of vertex in graph save to neighbors
        Repeat for every neighbor in neighbors
            Get neighbor from in_degree save to degree
            Add 1 to degree
```



```
            Set neighbor in in_degree to degree
        End repeat
    End repeat

    # Add vertices with in-degree 0 to queue
    Repeat for every vertex in vertices
        Get vertex from in_degree save to degree
        If degree is 0 then
            Add vertex to zero_degree
        End check
    End repeat

    # Process vertices
    Loop while zero_degree is not empty
        Dequeue from zero_degree save to current
        Add current to result

        Get neighbors of current in graph save to neighbors
        Repeat for every neighbor in neighbors
            Get neighbor from in_degree save to degree
            Subtract 1 from degree
            Set neighbor in in_degree to degree

            If degree is 0 then
                Add neighbor to zero_degree
            End check
        End repeat
    End loop
```



```
    # Check if all vertices processed (no cycles)
    If length of result is not equal to length of vertices then
        Raise error "Graph contains cycle"
    End check

    Return result
End function

# Example: Course prerequisites
Create directed graph named Courses

Add edge from "Intro CS" to "Data Structures" in Courses
Add edge from "Data Structures" to "Algorithms" in Courses
Add edge from "Intro CS" to "Databases" in Courses
Add edge from "Databases" to "Advanced Databases" in Courses
Add edge from "Data Structures" to "Operating Systems" in Courses

Run TopologicalSort with inputs Courses save to course_order

Write "Recommended course order:"
Repeat for every course in course_order
    Write course
End repeat
```

Output:

```
Recommended course order:
Intro CS
Data Structures
```



```
Databases
Algorithms
Operating Systems
Advanced Databases
```

19.10 Graph Coloring

Assign colors to vertices such that no adjacent vertices share same color.

```astra
To do GraphColoring with inputs graph, num_colors
    Create dictionary named coloring
    Get all vertices from graph save to vertices

    Repeat for every vertex in vertices
        # Try each color
        Create set named neighbor_colors
        Get neighbors of vertex in graph save to neighbors

        Repeat for every neighbor in neighbors
            If neighbor in coloring then
                Get neighbor from coloring save to color
                Add color to neighbor_colors
            End check
        End repeat

        # Find first available color
```



```
        Repeat for every color in range 0 to num_colors
            If color not in neighbor_colors then
                Set vertex in coloring to color
                Stop loop
            End check
        End repeat
    End repeat

    Return coloring
End function

Run GraphColoring with inputs CityMap, 4 save to colors
Write "Graph coloring:"
Write colors
```

Applications:

- Map coloring
- Register allocation in compilers
- Scheduling problems
- Frequency assignment in wireless networks

19.11 Network Flow

Find maximum flow through network from source to sink.

Ford-Fulkerson Algorithm

```astra
To do MaxFlow with inputs graph, source, sink
    # Create residual graph (copy of original)
    Copy graph save to residual_graph
```



```
    Set max_flow to 0

    # While there exists augmenting path
    Loop while True
        # Find path using BFS
        Run BFSPath with inputs residual_graph, source, sink save to
path

        If path is Nothing then
            Stop loop
        End check

        # Find minimum capacity along path
        Set min_capacity to infinity
        Repeat for every edge in path
            Get capacity of edge in residual_graph save to capacity
            If capacity is less than min_capacity then
                Set min_capacity to capacity
            End check
        End repeat

        # Update residual capacities
        Repeat for every edge in path
            Get source from edge save to u
            Get destination from edge save to v

            # Forward edge: decrease capacity
            Get capacity from u to v save to forward_cap
            Subtract min_capacity from forward_cap
```



```
            Set capacity from u to v to forward_cap

            # Backward edge: increase capacity
            Get capacity from v to u save to backward_cap
            Add min_capacity to backward_cap
            Set capacity from v to u to backward_cap
        End repeat

        # Increase total flow
        Add min_capacity to max_flow
    End loop

    Return max_flow
End function
```

Applications:

- Transportation networks
- Computer networks
- Bipartite matching
- Project selection

19.12 Social Network Analysis

Practical Example:

```astra
# Build social network
Create graph named SocialNetwork

# Add people
Add vertex "Alice" to SocialNetwork
```



```
Add vertex "Bob" to SocialNetwork
Add vertex "Charlie" to SocialNetwork
Add vertex "Diana" to SocialNetwork
Add vertex "Eve" to SocialNetwork

# Add friendships
Add edge from "Alice" to "Bob" in SocialNetwork
Add edge from "Alice" to "Charlie" in SocialNetwork
Add edge from "Bob" to "Diana" in SocialNetwork
Add edge from "Charlie" to "Diana" in SocialNetwork
Add edge from "Diana" to "Eve" in SocialNetwork

# Analyze network
To do CalculateDegree with inputs graph, vertex
    Get neighbors of vertex in graph save to neighbors
    Get length of neighbors save to degree
    Return degree
End function

To do FindInfluencers with inputs graph
    Get all vertices from graph save to people
    Create list named influencers

    Repeat for every person in people
        Run CalculateDegree with inputs graph, person save to
connections
        Create dictionary named person_data
        Set "name" in person_data to person
        Set "connections" in person_data to connections
```



```
        Add person_data to influencers
    End repeat

    Sort influencers by "connections" descending
    Return influencers
End function

Run FindInfluencers with inputs SocialNetwork save to
top_influencers

Write "Most Connected People:"
Repeat first 3 from top_influencers using person
    Get "name" from person save to name
    Get "connections" from person save to count
    Write name
    Write " has "
    Write count
    Write " connections"
End repeat

# Find shortest connection path
Run DijkstraShortestPath with inputs SocialNetwork, "Alice", "Eve"
save to path, steps

Write "Connection path from Alice to Eve:"
Write path
Write "Degrees of separation: "
Subtract 1 from length of path save to separation
Write separation
```



```

---

## 19.13 Real-World Applications

### 1. Route Planning (GPS Navigation)

```astra
Create weighted graph named RoadNetwork

# Add intersections and roads with travel times
Add vertex "Home" to RoadNetwork
Add vertex "Grocery Store" to RoadNetwork
Add vertex "Gas Station" to RoadNetwork
Add vertex "Work" to RoadNetwork

Add edge from "Home" to "Grocery Store" with weight 5 in RoadNetwork
Add edge from "Home" to "Gas Station" with weight 3 in RoadNetwork
Add edge from "Gas Station" to "Work" with weight 7 in RoadNetwork
Add edge from "Grocery Store" to "Work" with weight 4 in RoadNetwork

Run DijkstraShortestPath with inputs RoadNetwork, "Home", "Work"
save to route, time

Write "Fastest route: "
Write route
Write "Estimated time: "
Write time
Write " minutes"
```


```

**2. Recommendation Systems**

```astra
Create graph named ProductGraph

# Users and products they liked
Add edge from "User1" to "ProductA" in ProductGraph
Add edge from "User1" to "ProductB" in ProductGraph
Add edge from "User2" to "ProductB" in ProductGraph
Add edge from "User2" to "ProductC" in ProductGraph
Add edge from "User3" to "ProductA" in ProductGraph
Add edge from "User3" to "ProductC" in ProductGraph

To do RecommendProducts with inputs graph, user
    # Find products liked by similar users
    Get neighbors of user in graph save to liked_products
    Create set named recommendations

    Repeat for every product in liked_products
        # Find other users who liked this product
        Get neighbors of product in graph save to similar_users

        Repeat for every similar_user in similar_users
            If similar_user is not user then
                # Get their other liked products
                Get neighbors of similar_user in graph save to
their_products
                Add their_products to recommendations
            End check
```



```
        End repeat
    End repeat

    # Remove products user already liked
    Remove liked_products from recommendations
    Return recommendations
End function

Run RecommendProducts with inputs ProductGraph, "User1" save to
suggestions

Write "Recommended products: "

Write suggestions
```

19.14 Graph Algorithms Summary

Algorithm	**Purpose**	**Time Complexity**
BFS	Shortest path (unweighted)	O(V + E)
DFS	Traversal, cycle detection	O(V + E)
Dijkstra	Shortest path (weighted, non-negative)	O((V + E) log V)
A*	Shortest path (with heuristic)	O(E) best case.
Kruskal	Minimum spanning tree	O(E log E)
Prim	Minimum spanning tree	O((V + E) log V)
Topo Sort	Ordering with dependencies	O(V + E)
Ford-Fulk	Maximum flow	O(E × max_flow)

19.15 Professional Insights

Graphs are essential in:

1. Social Networks

- Friend suggestions (shortest path)
- Community detection (clustering)
- Influence analysis (centrality metrics)

2. Transportation & Logistics

- Route optimization
- Traffic flow analysis
- Supply chain networks

3. Computer Networks

- Routing protocols
- Network topology design
- Fault tolerance

4. Biology

- Protein interaction networks
- Gene regulatory networks
- Phylogenetic trees

5. Web

- PageRank algorithm
- Link analysis
- Web crawling strategies

19.16 Key Takeaways

1. **Graphs model relationships between entities**
2. **BFS finds shortest path in unweighted graphs**
3. **Dijkstra finds shortest path in weighted graphs**
4. **Topological sort orders vertices with dependencies**
5. **MST connects all vertices with minimum total weight**
6. **Graph algorithms solve real-world optimization problems**
7. **Choose representation (matrix vs list) based on graph density**

19.17 Exercises

Beginner Level

Implement basic graph operations:

- Create undirected graph of your friend network
- Implement BFS and DFS traversal
- Find path between two people
- Calculate degree of each person
- Identify most connected person

Intermediate Level

Build a course prerequisite system:

- Create directed graph of courses and prerequisites
- Implement topological sort to find valid course order
- Detect if prerequisite cycle exists
- Find all courses that can be taken in first semester
- Calculate minimum semesters needed to complete all courses

Advanced Level

Create a navigation system:

- Build weighted graph of city street network
- Implement Dijkstra's algorithm from scratch
- Implement A* with Euclidean distance heuristic
- Compare performance (time and path quality)
- Add real-time traffic (edge weights change)
- Find k shortest paths (not just shortest)
- Visualize graph and solution paths
- Handle one-way streets (directed edges)

Chapter 20: Testing, Debugging & Code Quality

Ensuring Correctness and Maintainability

20.1 Why Testing Matters

Reality of Software Development:

- Bugs cost companies billions annually
- Untested code breaks in production
- Manual testing doesn't scale
- Code changes introduce regressions

Professional Truth: Code that isn't tested doesn't work, even if it appears to.

20.2 Types of Testing

Testing Pyramid:

```
        /\
       /  \  E2E Tests (few)
      /____\
     /      \
    / Integr.\  Integration Tests (some)
   /__________\
  /            \
 /   Unit Tests \  Unit Tests (many)
/________________\
```

20.3 Assertions: The Foundation

Built-in Safety Checks:

```astra
To do CalculateArea with inputs width, height
    # Pre-conditions
    Verify that width is greater than 0 with message "Width must be positive"
    Verify that height is greater than 0 with message "Height must be positive"

    Multiply width by height save to area

    # Post-condition
    Verify that area is greater than 0 with message "Area calculation error"

    Return area
End function

# This will fail
Run CalculateArea with inputs -5, 10
# Output: AssertionError: Width must be positive
```

When to Use Assertions:

- Validate function inputs
- Check intermediate calculations

- Verify function outputs
- Document assumptions

20.4 Unit Testing

Test individual functions in isolation:

```astra
To do Test_CalculateArea
    Write "Running unit tests for CalculateArea..."

    # Test 1: Normal rectangle
    Run CalculateArea with inputs 5, 3 save to result
    Verify that result is 15 with message "Test 1 failed: 5×3"
    Write "✓ Test 1 passed: Normal rectangle"

    # Test 2: Square
    Run CalculateArea with inputs 4, 4 save to result
    Verify that result is 16 with message "Test 2 failed: 4×4"
    Write "✓ Test 2 passed: Square"

    # Test 3: Large dimensions
    Run CalculateArea with inputs 1000, 500 save to result
    Verify that result is 500000 with message "Test 3 failed: Large dimensions"
    Write "✓ Test 3 passed: Large dimensions"

    # Test 4: Decimal dimensions
    Run CalculateArea with inputs 2.5, 4.0 save to result
```



```
    Verify that result is 10.0 with message "Test 4 failed:
Decimals"

    Write "✓ Test 4 passed: Decimal dimensions"


    # Test 5: Invalid input (negative width)

    Set test_passed to False

    Attempt

        Run CalculateArea with inputs -5, 10

        # Should not reach here

        Verify that False with message "Test 5 failed: Should reject
negative width"

    Recover

        Set test_passed to True

    End attempt

    Verify that test_passed with message "Test 5 failed: Exception
not raised"

    Write "✓ Test 5 passed: Negative width rejected"


    # Test 6: Invalid input (zero height)

    Set test_passed to False

    Attempt

        Run CalculateArea with inputs 10, 0

        Verify that False with message "Test 6 failed: Should reject
zero height"

    Recover

        Set test_passed to True

    End attempt

    Verify that test_passed with message "Test 6 failed: Exception
not raised"

    Write "✓ Test 6 passed: Zero height rejected"
```



```
    Write "===== All tests passed! ====="
End function

Run Test_CalculateArea
```

20.5 Test-Driven Development (TDD)

Process: Red → Green → Refactor

1. Write Test First (Red):

```astra
To do Test_Fibonacci
    Run Fibonacci with inputs 0 save to result
    Verify that result is 0

    Run Fibonacci with inputs 1 save to result
    Verify that result is 1

    Run Fibonacci with inputs 5 save to result
    Verify that result is 5

    Run Fibonacci with inputs 10 save to result
    Verify that result is 55
End function

# Running now will fail - function doesn't exist yet
```


2. Write Minimal Code to Pass (Green):

```astra
To do Fibonacci with inputs n
    If n is less than 2 then
        Return n
    End check

    Run Fibonacci with inputs n - 1 save to a
    Run Fibonacci with inputs n - 2 save to b
    Add a to b save to result
    Return result
End function

# Now tests pass!
```

3. Refactor (Improve):

```astra
# Optimized version with memoization
Create dictionary named fib_cache

To do Fibonacci with inputs n
    If n in fib_cache then
        Get n from fib_cache save to cached
        Return cached
    End check

    If n is less than 2 then
        Set n in fib_cache to n
```



```
        Return n
    End check

    Run Fibonacci with inputs n - 1 save to a
    Run Fibonacci with inputs n - 2 save to b
    Add a to b save to result
    Set n in fib_cache to result

    Return result
End function

# Tests still pass, but now much faster!
```

20.6 Testing Best Practices

Arrange-Act-Assert Pattern:

```astra
To do Test_UserRegistration
    # Arrange: Set up test data
    Create dictionary named user_data
    Set "username" in user_data to "testuser"
    Set "email" in user_data to "test@example.com"
    Set "password" in user_data to "SecurePass123"

    # Act: Execute the function
    Run RegisterUser with inputs user_data save to result
```



```
    # Assert: Verify outcomes
    Get "success" from result save to success
    Verify that success is True
    Get "user_id" from result save to user_id
    Verify that user_id is not Nothing

    Write "✓ User registration test passed"
End function
```

Test Edge Cases:

```astra
To do Test_DivideNumbers
    # Normal case
    Run Divide with inputs 10, 2 save to result
    Verify that result is 5.0

    # Division by 1
    Run Divide with inputs 10, 1 save to result
    Verify that result is 10.0

    # Division resulting in decimal
    Run Divide with inputs 7, 2 save to result
    Verify that result is 3.5

    # Division by zero (should raise error)
    Attempt
```



```
        Run Divide with inputs 10, 0
        Verify that False with message "Should raise error for
division by zero"
    Recover
        Write "✓ Division by zero correctly rejected"
    End attempt

    # Negative numbers
    Run Divide with inputs -10, 2 save to result
    Verify that result is -5.0

    Run Divide with inputs 10, -2 save to result
    Verify that result is -5.0

    Write "✓ All division tests passed"
End function
```

20.7 Integration Testing

Test multiple components together:

```astra
To do Test_OrderProcessing
    Write "Testing complete order processing flow..."

    # Create test order
    Create dictionary named order
    Set "product_id" in order to "PROD123"
```



```
    Set "quantity" in order to 2

    Set "customer_id" in order to "CUST456"


    # Test inventory check

    Run CheckInventory with inputs "PROD123", 2 save to available

    Verify that available is True with message "Inventory check
failed"

    Write "✓ Inventory check passed"


    # Test price calculation

    Run CalculateOrderTotal with inputs order save to total

    Verify that total is greater than 0 with message "Price
calculation failed"

    Write "✓ Price calculation passed"


    # Test order creation

    Run CreateOrder with inputs order save to order_id

    Verify that order_id is not Nothing with message "Order creation
failed"

    Write "✓ Order creation passed"


    # Test inventory deduction

    Run DeductInventory with inputs "PROD123", 2 save to success

    Verify that success is True with message "Inventory deduction
failed"

    Write "✓ Inventory deduction passed"


    # Verify final state

    Run GetOrder with inputs order_id save to saved_order

    Get "status" from saved_order save to status
```



```
    Verify that status is "pending" with message "Order status
incorrect"
    Write "✓ Order status verified"

    Write "===== Integration test passed! ====="
End function
```

20.8 Debugging Techniques

Print Debugging

```astra
To do ProcessData with inputs data
    Write "DEBUG: Entering ProcessData with "
    Get length of data save to len
    Write len
    Write " items"

    Create list named results

    Repeat for every item in data using index
        Write "DEBUG: Processing item "
        Write index
        Write ": "
        Write item

        # Process item
        Run TransformItem with inputs item save to transformed
```



```
        Write "DEBUG: Transformed to "
        Write transformed

        Add transformed to results
    End repeat

    Write "DEBUG: Returning "
    Get length of results save to result_len
    Write result_len
    Write " results"

    Return results
End function
```

Best Practice: Remove debug prints before production, or use conditional logging.

Conditional Debugging

```astra
Set DEBUG_MODE to True

To do DebugLog with inputs message
    If DEBUG_MODE is True then
        Write "[DEBUG] "
        Write message
    End check
End function
```



```
To do ProcessData with inputs data
    Run DebugLog with inputs "Starting data processing"

    # ... processing logic

    Run DebugLog with inputs "Processing complete"
    Return results
End function
```

Pause and Inspect

```astra
To do ComplexCalculation with inputs x, y
    Calculate x multiplied by 2 save to step1

    # Pause to inspect state
    Pause execution
    Write "Current state:"
    Write "  x = "
    Write x
    Write "  y = "
    Write y
    Write "  step1 = "
    Write step1

    Add step1 to y save to step2
    Divide step2 by 3 save to result
```



```
    Return result
End function
```

Binary Search Debugging

When bug is in large codebase:

```astra
# Comment out half the code
# Does bug still occur?
# If yes: bug is in remaining half
# If no: bug is in commented half
# Repeat until isolated
```

Rubber Duck Debugging

Explain your code line-by-line to:

- A colleague
- A rubber duck
- Yourself out loud

Often reveals the bug through verbalization.

20.9 Logging

Professional Logging System:

```astra
Create logger named AppLogger
```



```
To do SetupLogging
    Configure AppLogger with
        Set level to "INFO"
        Set format to "[{timestamp}] {level}: {message}"
        Set output to "app.log"
    End configuration
End function

To do ProcessOrder with inputs order_id
    Log INFO to AppLogger: "Processing order {order_id}"

    Attempt
        Run ValidateOrder with inputs order_id save to valid

        If not valid then
            Log WARNING to AppLogger: "Invalid order {order_id}"
            Return False
        End check

        Run ChargePayment with inputs order_id save to charged

        If not charged then
            Log ERROR to AppLogger: "Payment failed for order
{order_id}"
            Return False
        End check

        Run FulfillOrder with inputs order_id
```



```
        Log INFO to AppLogger: "Order {order_id} fulfilled
successfully"


        Return True


    Recover
        Log ERROR to AppLogger: "Exception processing order
{order_id}: {error_message}"
        Return False
    End attempt
End function
```

Log Levels:

- **DEBUG:** Detailed diagnostic information
- **INFO:** General informational messages
- **WARNING:** Warning messages (non-critical issues)
- **ERROR:** Error messages (functionality affected)
- **CRITICAL:** Critical errors (system failure)

20.10 Code Quality Metrics

Cyclomatic Complexity

Measure of code complexity (number of independent paths).

```astra
# Low complexity (good)
To do SimpleFunction with inputs x
    If x is greater than 0 then
```



```
        Return "positive"
    Else
        Return "non-positive"
    End check
End function
# Complexity: 2

# High complexity (bad)
To do ComplexFunction with inputs a, b, c, d
    If a is greater than 0 then
        If b is greater than 0 then
            If c is greater than 0 then
                If d is greater than 0 then
                    Return 1
                Else
                    Return 2
                End check
            Else
                Return 3
            End check
        Else
            Return 4
        End check
    Else
        Return 5
    End check
End function
# Complexity: 5+ (too high!)
```


Goal: Keep complexity under 10 per function.

Code Coverage

Percentage of code executed by tests.

```astra
To do CalculateGrade with inputs score
    If score is greater than or equal to 90 then
        Return "A"
    Else if score is greater than or equal to 80 then
        Return "B"
    Else if score is greater than or equal to 70 then
        Return "C"
    Else if score is greater than or equal to 60 then
        Return "D"
    Else
        Return "F"
    End check
End function

# Tests should cover all branches
To do Test_CalculateGrade
    Verify that CalculateGrade(95) is "A"  # Branch 1
    Verify that CalculateGrade(85) is "B"  # Branch 2
    Verify that CalculateGrade(75) is "C"  # Branch 3
    Verify that CalculateGrade(65) is "D"  # Branch 4
    Verify that CalculateGrade(55) is "F"  # Branch 5
    # 100% coverage!
End function
```



```

**Target:** 80%+ code coverage (100% for critical paths).

---

## 20.11 Code Style and Standards

**Naming Conventions:**

```astra
# Good: Descriptive names
To do CalculateMonthlyPayment with inputs loan_amount, interest_rate, num_months
    # Clear what function does and what inputs mean
End function

# Bad: Cryptic names
To do Calc with inputs amt, rate, n
    # What does this calculate?
End function

# Variables
Set user_age to 25                    # Good: snake_case for variables
Set total_price to 99.99              # Good: descriptive
Set x to 5                            # Bad: meaningless (unless in math context)

# Constants
Set MAX_RETRIES to 3                  # Good: UPPER_CASE for constants
Set API_KEY to "abc123"               # Good: clearly a constant
```


```

```
Functions
To do CalculateTotal   # Good: verb describing action
To do GetUserData   # Good: clear purpose
To do Process   # Bad: too vague
```

Function Length:

```astra
# Good: Focused, single responsibility
To do ValidateEmail with inputs email
    If length of email is less than 5 then
        Return False
    End check

    If email does not contain "@" then
        Return False
    End check

    Return True
End function

# Bad: Too long, multiple responsibilities
To do ProcessUserRegistration with inputs data
    # 100 lines of validation
    # 50 lines of database logic
    # 30 lines of email sending
    # 40 lines of logging
    # ... (too much in one function!)
```



```
End function

# Better: Split into focused functions
To do ProcessUserRegistration with inputs data
    Run ValidateUserData with inputs data
    Run CreateUserAccount with inputs data
    Run SendWelcomeEmail with inputs data
    Run LogRegistration with inputs data
End function
```

Rule of Thumb: Functions should fit on one screen (< 50 lines).

Comments:

```astra
# Good: Explain WHY, not WHAT
To do CalculateDiscount with inputs price, customer_type
    # Enterprise customers get 20% discount due to contract terms
    If customer_type is "Enterprise" then
        Multiply price by 0.8 save to discounted_price
        Return discounted_price
    End check

    Return price
End function

# Bad: Obvious comments
To do Add with inputs a, b
```



```
    # Add a and b  (obvious from code!)
    Add b to a save to sum
    # Return sum  (obvious from code!)
    Return sum
End function

# Good: Document complex algorithms
To do QuickSort with inputs array, low, high
    # Using Hoare partition scheme for better performance
    # on arrays with many duplicate elements
    If low is less than high then
        Run Partition with inputs array, low, high save to pivot
        Run QuickSort with inputs array, low, pivot
        Run QuickSort with inputs array, pivot + 1, high
    End check
End function
```

20.12 Code Reviews

Checklist for Reviewers:

```astra
# Code Review Checklist

To do ReviewCode with inputs code_changes
    Create list named issues

    # 1. Correctness
```



```
    Check if logic is correct
    Check if edge cases are handled
    Check if error handling is present

    # 2. Testing
    Verify tests exist for new code
    Verify tests cover edge cases
    Check if tests actually test the logic

    # 3. Code Style
    Check naming conventions
    Check function length
    Verify comments explain WHY not WHAT

    # 4. Performance
    Check for inefficient algorithms
    Check for unnecessary loops
    Verify database queries are optimized

    # 5. Security
    Check for SQL injection vulnerabilities
    Check for XSS vulnerabilities
    Verify input validation
    Check for hardcoded credentials

    # 6. Maintainability
    Check if code is modular
    Verify single responsibility principle
    Check for code duplication
```



```
    Return issues
End function
```

20.13 Performance Profiling

Measure Execution Time:

```astra
To do ProfileFunction with inputs function_name
    Get current time save to start_time

    Run function_name

    Get current time save to end_time
    Subtract start_time from end_time save to elapsed

    Write "Function "
    Write function_name
    Write " took "
    Write elapsed
    Write " seconds"

    Return elapsed
End function

# Profile different implementations
Run ProfileFunction with inputs "SlowSortAlgorithm" save to time1
```



```
Run ProfileFunction with inputs "FastSortAlgorithm" save to time2

Write "Speedup: "
Divide time1 by time2 save to speedup
Write speedup
Write "x"
```

Detailed Profiling:

```astra
Create profiler named PerformanceProfiler

To do ProfiledFunction with inputs data
    Start profiling PerformanceProfiler

    # Section 1
    Start timer "data_loading"
    Run LoadData with inputs data
    Stop timer "data_loading"

    # Section 2
    Start timer "processing"
    Run ProcessData with inputs data
    Stop timer "processing"

    # Section 3
    Start timer "saving"
    Run SaveResults with inputs data
```



```
    Stop timer "saving"

    Stop profiling PerformanceProfiler

    Get profile report from PerformanceProfiler save to report
    Write report
End function
```

Output:

```
Profiling Report:
  data_loading: 0.5s (25%)
  processing: 1.2s (60%)
  saving: 0.3s (15%)
  Total: 2.0s
```

20.14 Memory Profiling

Track Memory Usage:

```astra
To do CheckMemoryUsage
    Get current memory usage save to mem_before

    # Create large data structure
    Create list named large_list
    Repeat 1000000 times
```



```
        Add "item" to large_list
    End repeat

    Get current memory usage save to mem_after
    Subtract mem_before from mem_after save to mem_used

    Write "Memory used: "
    Divide mem_used by 1024 * 1024 save to mb
    Write mb
    Write " MB"
End function
```

20.15 Continuous Integration

Automated Testing Pipeline:

```astra
# CI/CD Pipeline Configuration (Conceptual)

On code push to repository:
    # Stage 1: Build
    Run BuildProject

    # Stage 2: Lint
    Run CodeStyleCheck
    If style_issues found then
        Fail pipeline with message "Code style violations"
    End check
```



```
    # Stage 3: Unit Tests
    Run AllUnitTests
    If any tests fail then
        Fail pipeline with message "Unit tests failed"
    End check

    # Stage 4: Integration Tests
    Run IntegrationTests
    If any tests fail then
        Fail pipeline with message "Integration tests failed"
    End check

    # Stage 5: Code Coverage
    Calculate code coverage
    If coverage is less than 80 then
        Fail pipeline with message "Insufficient code coverage"
    End check

    # Stage 6: Security Scan
    Run SecurityScan
    If vulnerabilities found then
        Fail pipeline with message "Security vulnerabilities
detected"
    End check

    # Stage 7: Deploy (if all pass)
    Deploy to staging environment
```



```
    Write "✓ All checks passed - ready for production"
End pipeline
```

20.16 Documentation

Function Documentation:

```astra
To do CalculateInterest with inputs principal, rate, time
    # Calculate compound interest
    #
    # Purpose:
    #   Calculates the compound interest on a principal amount
    #
    # Parameters:
    #   principal (float): The initial amount of money
    #   rate (float): Annual interest rate (as decimal, e.g., 0.05 for 5%)
    #   time (int): Time period in years
    #
    # Returns:
    #   float: The total amount after interest
    #
    # Example:
    #   Run CalculateInterest with inputs 1000, 0.05, 2 save to amount
    #   # Returns: 1102.50
    #
```



```
    # Raises:
    #   Error if principal is negative
    #   Error if rate is negative
    #   Error if time is negative

    Verify that principal is greater than or equal to 0
    Verify that rate is greater than or equal to 0
    Verify that time is greater than or equal to 0

    Add 1 to rate save to multiplier
    Raise multiplier to power time save to factor
    Multiply principal by factor save to amount

    Return amount
End function
```

20.17 Error Messages

Good vs Bad Error Messages:

```astra
# Bad: Cryptic
"Error: Invalid input"
"Failed"
"Exception at line 42"

# Good: Specific and actionable
"Error: Email address must contain '@' symbol"
```



```
"File not found: config.json (expected in /etc/app/)"
"Invalid age: Must be between 0 and 150, got -5"
```

Implementing Good Errors:

```astra
To do ValidateAge with inputs age
    If age is less than 0 then
        Raise error "Invalid age: {age}. Age cannot be negative."
    End check

    If age is greater than 150 then
        Raise error "Invalid age: {age}. Age must be 150 or less."
    End check

    Return True
End function
```

20.18 Defensive Programming

Validate All Inputs:

```astra
To do ProcessPayment with inputs amount, card_number, cvv
    # Validate amount
    If amount is Nothing then
        Raise error "Amount is required"
```



```
    End check

    If amount is less than or equal to 0 then
        Raise error "Amount must be positive, got {amount}"
    End check

    If amount is greater than 10000 then
        Raise error "Amount exceeds maximum transaction limit"
    End check

    # Validate card number
    If card_number is Nothing then
        Raise error "Card number is required"
    End check

    Get length of card_number save to card_length
    If card_length is not 16 then
        Raise error "Card number must be 16 digits"
    End check

    # Validate CVV
    If cvv is Nothing then
        Raise error "CVV is required"
    End check

    Get length of cvv save to cvv_length
    If cvv_length is not 3 and cvv_length is not 4 then
        Raise error "CVV must be 3 or 4 digits"
    End check
```



```
    # Process payment (all inputs validated)
    Run ChargeCard with inputs amount, card_number, cvv
End function
```

20.19 Technical Debt

Managing Debt:

```astra
# Mark technical debt with TODO comments
To do LegacyFunction with inputs data
    # TODO: Refactor this function - too complex
    # TODO: Add error handling
    # TODO: Optimize performance (currently O(n²))
    # ... old code ...
End function

# Track debt in backlog
Create technical debt item
    Set title to "Refactor LegacyFunction"
    Set priority to "Medium"
    Set estimated_effort to "4 hours"
    Set impact to "Improves maintainability and performance"
End item
```

Avoid Accumulating Debt:

- Refactor as you go

- Don't skip tests "temporarily"
- Address code review comments
- Pay down debt regularly

20.20 Professional Insights

Testing in Production Systems:

1. Test Automation

- Unit tests run on every commit
- Integration tests run nightly
- End-to-end tests before deployment

2. Monitoring

- Error tracking (Sentry, Rollbar)
- Performance monitoring (New Relic, DataDog)
- User analytics

3. Feature Flags

- Deploy code without activating features
- A/B test new features
- Quick rollback if issues detected

4. Canary Deployments

- Deploy to 5% of servers first
- Monitor error rates
- Gradually increase if stable

20.21 Key Takeaways

1. **Testing is not optional—untested code doesn't work**
2. **Write tests first (TDD) leads to better design**
3. **Assertions catch bugs early in development**
4. **Code coverage should exceed 80% for critical paths**
5. **Debugging is systematic, not random trial and error**
6. **Code reviews catch bugs and share knowledge**
7. **Performance profiling identifies bottlenecks**
8. **Good documentation saves countless hours**
9. **Defensive programming prevents edge case failures**
10. **Continuous integration catches regressions immediately**

20.22 Exercises

Beginner Level

Create a test suite:
- Write a function to check if string is palindrome
- Write 5+ test cases covering:
  - Normal palindromes ("racecar")
  - Single character ("a")
  - Empty string ("")
  - Non-palindromes ("hello")
  - Spaces and punctuation ("A man, a plan, a canal: Panama")
- Use assertions to validate
- Make all tests pass

Intermediate Level

Build a tested calculator:

- Implement basic operations (+, -, ×, ÷)
- Write comprehensive test suite:
  - Normal operations
  - Edge cases (divide by zero, overflow)
  - Negative numbers
  - Decimals
  - Chained operations
- Implement logging for all operations
- Measure code coverage (should be 100%)

Advanced Level

Create a complete tested system:

- Choose domain (shopping cart, banking, task manager)
- Implement core functionality
- Write unit tests for all functions
- Write integration tests for workflows
- Set up logging system with levels
- Implement error handling throughout
- Add performance profiling
- Write comprehensive documentation
- Create code review checklist
- Measure and report code coverage
- Implement CI/CD pipeline (conceptual)
- Achieve >90% test coverage

APPENDICES

Appendix A: Formal Language Specification

A.1 Lexical Elements

Keywords (Reserved Words)

```
Control Flow:
  If, Else, End, Then, Check, Loop, While, Repeat, Times
  For, Every, In, Stop, Skip, Iteration

Data Operations:
  Set, To, Add, Subtract, Multiply, Divide, Create, Get
  From, Save, Into, With, Using, By

Functions:
  To, Do, Run, Return, Inputs, Function

Logical:
  And, Or, Not, Is, Greater, Less, Than, Equal

Error Handling:
  Attempt, Recover, Always, Raise, Error

Objects:
  Define, Blueprint, Named, Has, Action, Object
```



```
Files:
  Read, Write, File, Append, Using, Usage

Types:
  List, Dictionary, Stack, Queue, Tree, Graph, Set

Values:
  True, False, Nothing

Async:
  Async, Await, On, Event

Data Science:
  Load, Dataset, Visualize, As, Train, Predict, Model

Testing:
  Verify, That, Message
```

Identifiers

- **Pattern:** `[a-zA-Z][a-zA-Z0-9_]*`
- **Case-sensitive:** `userName` ≠ `UserName`
- **Cannot start with digit:** `1user` is invalid
- **Cannot contain hyphens:** `user-name` is invalid

Valid Examples:

```
x
user_name
```



```
totalPrice
calculate2D
_internal
```

Invalid Examples:

```
1variable    # Starts with digit
user-name    # Contains hyphen
class        # Reserved keyword
my.var       # Contains dot
```

Literals

Integer:

```
42
-17
0
1000000
```

Float:

```
3.14
-0.5
2.0
1e6      # Scientific notation
```


String:

```
"Hello, World!"
'Single quotes work too'
"String with \"escaped\" quotes"
"Multi-line strings
are allowed"
```

Boolean:

```
True
False
```

Nothing:

```
Nothing
```

Operators (English Keywords)

Arithmetic:

```
Add ... to ...
Subtract ... from ...
Multiply ... by ...
Divide ... by ...
```


Comparison:

```
is

is not

is greater than

is less than

is greater than or equal to

is less than or equal to
```

Logical:

```
and

or

not
```

A.2 Syntax Grammar (EBNF)

```ebnf
program = { statement } ;

statement = assignment
          | operation
          | control_flow
          | function_definition
          | function_call
          | output ;

assignment = "Set", identifier, "to", expression ;
```



```
operation = arithmetic_operation
          | list_operation
          | dictionary_operation ;

arithmetic_operation = "Add", expression, "to", identifier
                     | "Subtract", expression, "from", identifier
                     | "Multiply", identifier, "by", expression
                     | "Divide", identifier, "by", expression ;

control_flow = if_statement
             | repeat_loop
             | while_loop
             | for_loop ;

if_statement = "If", condition, "then",
               { statement },
               [ "Else if", condition, "then", { statement } ],
               [ "Else", { statement } ],
               "End check" ;

repeat_loop = "Repeat", expression, "times",
              { statement },
              "End repeat" ;

while_loop = "Loop while", condition,
             { statement },
             "End loop" ;
```



```
for_loop = "Repeat for every", identifier, "in", expression,
           { statement },
           "End repeat" ;

condition = expression, comparison_operator, expression
          | condition, "and", condition
          | condition, "or", condition
          | "not", condition ;

comparison_operator = "is"
                    | "is not"
                    | "is greater than"
                    | "is less than"
                    | "is greater than or equal to"
                    | "is less than or equal to" ;

expression = literal
           | identifier
           | function_call ;

function_definition = "To do", identifier, [ "with inputs",
parameter_list ],
                      { statement },
                      "End function" ;

function_call = "Run", identifier, [ "with inputs", argument_list ],
                [ "save to", identifier ] ;

output = "Write", expression ;
```



```
identifier = letter, { letter | digit | "_" } ;

literal = integer | float | string | boolean | "Nothing" ;

integer = [ "-" ], digit, { digit } ;

float = [ "-" ], digit, { digit }, ".", digit, { digit } ;

string = '"', { character }, '"' | "'", { character }, "'" ;

boolean = "True" | "False" ;

letter = "a" | ... | "z" | "A" | ... | "Z" ;

digit = "0" | "1" | ... | "9" ;
```

A.3 Type System

Primitive Types

Type	Description	Example	Mutable
Integer	Arbitrary precision integers	`42`, `-17`.	No
Float	IEEE 754 double precision	`3.14`, `-0.5`	No
String	Unicode text	`"Hello"`	No
Boolean	Logical truth value	`True`, `False`	No
Nothing	Absence of value	`Nothing`	N/A

Composite Types

Type	Description	Example	Mutable
List	Ordered collection	`[1, 2, 3]`	Yes
Dictionary	Key-value mapping	`{"name": "Alice"}`	Yes
Set	Unique elements	`{1, 2, 3}`	Yes
Tuple	Immutable sequence	`(1, 2, 3)`	No
Stack	LIFO collection	-	Yes
Queue	FIFO collection	-	Yes

A.4 Scoping Rules

Scope Hierarchy:

```
Global Scope
  │
  ├─ Function Scope
  │   │
  │   └─ Block Scope (If, Loop)
  │       │
  │       └─ Nested Block Scope
  │
  └─ Another Function Scope
```

Resolution Order:

1. Current block scope
2. Enclosing block scope
3. Function scope
4. Global scope

Shadowing: Inner scope can shadow outer names temporarily.

A.5 Error Types

Error	**Description**	**Example**
SyntaxError	Invalid syntax	Missing `End check`
NameError	Undefined variable	Using `x` before set
TypeError	Incompatible type operation	`Add "text" to 5`
ValueError	Invalid value	`Divide by 0`
IndexError	Invalid index	`Get item 10 from list`
KeyError	Key not in dictionary	`Get "missing_key"`

Appendix B: Python-Astra Concept Mapping

B.1 Variable Operations

Python	Astra
`x = 10`	`Set x to 10`
`x += 5`	`Add 5 to x`
`x -= 3`	`Subtract 3 from x`
`x *= 2`	`Multiply x by 2`
`x /= 4`	`Divide x by 4`
`del x`	*(Variable goes out of scope)*

B.2 Control Flow

Python	**Astra**
`if x > 10:`	`If x is greater than 10 then`

`elif x > 5:`	`Else if x is greater than 5 then`
`else:`	`Else`
`for i in range(10):`	`Repeat 10 times`
`for item in items:`	`Repeat for every item in items`
`while condition:`	`Loop while condition`
`break`	`Stop loop`
`continue`	`Skip iteration`

B.3 Functions

Python	Astra
`def func(x, y):`	`To do func with inputs x, y`
`return result`	`Return result`
`value = func(a, b)`	`Run func with inputs a, b save to value`
`def func():`	`To do func`

B.4 Data Structures

Python	Astra
`lst = []`	`Create list named lst`
`lst.append(x)`	`Add x to lst`
`lst[0]`	`Get item 0 from lst save to x`
`dict = {}`	`Create dictionary named dict`
`dict['key'] = val`	`Set "key" in dict to val`
`dict['key']`	`Get "key" from dict save to val`
`set = set()`	`Create unique group named set`
`set.add(x)`	`Add x to group set`

B.5 Error Handling

Python	Astra
`try:`	`Attempt`
`except:`	`Recover`
`finally:`	`Always`
`raise Exception(msg)`	`Raise error msg`
`assert condition`	`Verify that condition`

B.6 Object-Oriented

Python	Astra
`class Dog:`	`Define Blueprint named Dog`
`def __init__(self):`	`Has properties...`
`def bark(self):`	`Action Bark`
`dog = Dog()`	`Create object dog from Dog`
`dog.bark()`	`Run Bark in dog`
`dog.name`	`Get name from dog` or `name in dog`

B.7 File Operations

Python	Astra
`with open('file.txt') as f:`	`Using file "file.txt" save to f`
`content = f.read()`	`Read from f save to content`
`f.write('text')`	`Write "text" into f`

B.8 Libraries

Python	Astra
`import math`	`Use python library math`
`math.sqrt(25)`	`Call python sqrt from math with i/p 25`
`import pandas as pd`	`Use python library pandas`
`pd.read_csv('file')`	`Call python read_csv from pandas "file"`

Appendix C: Common Patterns & Idioms

C.1 Input Validation Pattern

```astra
To do ValidateAndProcess with inputs data
    # Validate inputs
    If data is Nothing then
        Raise error "Data cannot be Nothing"
    End check

    If length of data is 0 then
        Raise error "Data cannot be empty"
    End check

    # Validate data type
    If type of data is not "list" then
        Raise error "Data must be a list"
    End check
```



```
    # Validate each element
    Repeat for every item in data
        If item is less than 0 then
            Raise error "All items must be non-negative"
        End check
    End repeat

    # Process validated data
    Run ProcessData with inputs data save to result
    Return result
End function
```

Use When:

- Accepting user input
- Receiving API parameters
- Processing external data
- Interfacing with untrusted sources

C.2 Retry with Exponential Backoff

```astra
To do async FetchWithRetry with inputs url, max_attempts
    Set attempts to 0
    Set delay to 1

    Loop while attempts is less than max_attempts
        Add 1 to attempts

 
```

```
        Attempt
            Await FetchURL with inputs url save to data
            Return data
        Recover
            Write "Attempt "
            Write attempts
            Write " failed: "
            Write error_message

            If attempts is less than max_attempts then
                Write "Retrying in "
                Write delay
                Write " seconds..."
                Await Sleep for delay seconds
                Multiply delay by 2 save to delay
            Else
                Raise error "Max retries exceeded"
            End check
        End attempt
    End loop
End function
```

Timeline:

```
Attempt 1: Fail → Wait 1 second
Attempt 2: Fail → Wait 2 seconds
Attempt 3: Fail → Wait 4 seconds
Attempt 4: Fail → Wait 8 seconds
```



```
...
```

Use When:

- Network requests
- API calls
- Database connections
- External service dependencies

C.3 Resource Management Pattern

```astra
To do SafeResourceOperation with inputs resource_path
    Set resource to Nothing

    Attempt
        # Acquire resource
        Open resource resource_path save to resource
        Write "Resource acquired"

        # Use resource
        Run ProcessResource with inputs resource save to result

        # Success
        Return result

    Recover
        Write "Error during operation: "
        Write error_message
```



```
        Return Nothing

    Always
        # Guaranteed cleanup
        If resource is not Nothing then
            Close resource
            Write "Resource released"
        End check
    End attempt
End function
```

Alternative Using Block:

```astra
To do SafeResourceOperation with inputs resource_path
    Using resource resource_path save to resource
        Run ProcessResource with inputs resource save to result
        Return result
    End usage
    # Resource automatically closed
End function
```

Use When:

- File operations
- Database connections
- Network sockets
- Hardware devices

C.4 Builder Pattern

```astra
Define Blueprint named QueryBuilder
    Has table set to Nothing
    Has columns set to Nothing
    Has conditions set to Nothing
    Has order_by set to Nothing
    Has limit set to Nothing

    Action SetTable with inputs table_name
        Set table to table_name
        Return this
    End Action

    Action SelectColumns with inputs column_list
        Set columns to column_list
        Return this
    End Action

    Action Where with inputs condition
        If conditions is Nothing then
            Create list named new_conditions
            Set conditions to new_conditions
        End check
        Add condition to conditions
        Return this
    End Action
```



```
    Action OrderBy with inputs column
        Set order_by to column
        Return this
    End Action

    Action Limit with inputs max_rows
        Set limit to max_rows
        Return this
    End Action

    Action Build
        # Construct SQL query
        Set query to "SELECT "

        If columns is Nothing then
            Add "*" to query
        Else
            Join columns with ", " save to cols
            Add cols to query
        End check

        Add " FROM " to query
        Add table to query

        If conditions is not Nothing then
            Add " WHERE " to query
            Join conditions with " AND " save to where_clause
            Add where_clause to query
        End check
```



```
        If order_by is not Nothing then
            Add " ORDER BY " to query
            Add order_by to query
        End check

        If limit is not Nothing then
            Add " LIMIT " to query
            Add limit to query
        End check

        Return query
    End Action
End Blueprint

# Usage
Create object builder from QueryBuilder
Run SetTable in builder with inputs "users"
Run SelectColumns in builder with inputs ["name", "email"]
Run Where in builder with inputs "age > 18"
Run OrderBy in builder with inputs "name"
Run Limit in builder with inputs 10
Run Build in builder save to query

Write query
# Output: SELECT name, email FROM users WHERE age > 18 ORDER BY name
LIMIT 10
```


Use When:

- Complex object construction
- Many optional parameters
- Fluent API design
- Query building

C.5 Factory Pattern

```astra
To do CreateShape with inputs shape_type, dimensions
    If shape_type is "circle" then
        Create object shape from Circle
        Get "radius" from dimensions save to radius
        Set radius in shape to radius
        Return shape

    Else if shape_type is "rectangle" then
        Create object shape from Rectangle
        Get "width" from dimensions save to width
        Get "height" from dimensions save to height
        Set width in shape to width
        Set height in shape to height
        Return shape

    Else if shape_type is "triangle" then
        Create object shape from Triangle
        Get "base" from dimensions save to base
        Get "height" from dimensions save to height
        Set base in shape to base
```



```
        Set height in shape to height
        Return shape

    Else
        Raise error "Unknown shape type: {shape_type}"
    End check
End function

# Usage
Create dictionary named circle_dims
Set "radius" in circle_dims to 5

Run CreateShape with inputs "circle", circle_dims save to circle
Run CalculateArea in circle save to area
```

Use When:

- Multiple related classes
- Creation logic depends on parameters
- Decoupling object creation from usage
- Plugin architectures

C.6 Strategy Pattern

```astra
Define Blueprint named PaymentProcessor
    Has strategy set to Nothing

    Action SetStrategy with inputs payment_strategy
```



```
        Set strategy to payment_strategy
    End Action

    Action ProcessPayment with inputs amount
        If strategy is Nothing then
            Raise error "No payment strategy set"
        End check

        Run Process in strategy with inputs amount
    End Action
End Blueprint

Define Blueprint named CreditCardStrategy
    Action Process with inputs amount
        Write "Processing $"
        Write amount
        Write " via Credit Card"
        # Credit card processing logic
    End Action
End Blueprint

Define Blueprint named PayPalStrategy
    Action Process with inputs amount
        Write "Processing $"
        Write amount
        Write " via PayPal"
        # PayPal processing logic
    End Action
End Blueprint
```



```
# Usage
Create object processor from PaymentProcessor
Create object credit_card from CreditCardStrategy

Run SetStrategy in processor with inputs credit_card
Run ProcessPayment in processor with inputs 100

# Switch strategy at runtime
Create object paypal from PayPalStrategy
Run SetStrategy in processor with inputs paypal
Run ProcessPayment in processor with inputs 50
```

Use When:

- Multiple algorithms for same task
- Runtime algorithm selection
- Avoiding complex conditionals
- Open/closed principle compliance

C.7 Observer Pattern

```astra
Define Blueprint named Observable
    Has observers set to Nothing

    Action Initialize
        Create list named new_observers
        Set observers to new_observers
    End Action
```



```
    Action Subscribe with inputs observer
        Add observer to observers
    End Action

    Action Unsubscribe with inputs observer
        Remove observer from observers
    End Action

    Action Notify with inputs event_data
        Repeat for every observer in observers
            Run Update in observer with inputs event_data
        End repeat
    End Action
End Blueprint

Define Blueprint named EmailObserver
    Action Update with inputs data
        Write "Sending email notification: "
        Write data
    End Action
End Blueprint

Define Blueprint named SMSObserver
    Action Update with inputs data
        Write "Sending SMS notification: "
        Write data
    End Action
End Blueprint
```



```
# Usage
Create object stock from Observable
Run Initialize in stock

Create object email_notifier from EmailObserver
Create object sms_notifier from SMSObserver

Run Subscribe in stock with inputs email_notifier
Run Subscribe in stock with inputs sms_notifier

# When stock price changes
Run Notify in stock with inputs "Stock price: $150"
# Both observers notified
```

Use When:

- Event-driven systems
- Pub/sub architectures
- GUI updates
- Monitoring systems

C.8 Singleton Pattern

```astra
Create dictionary named Singletons

To do GetSingleton with inputs class_name
    If class_name in Singletons then
        Get class_name from Singletons save to instance
```



```
        Return instance
    End check

    # Create new instance
    Create object instance from class_name
    Set class_name in Singletons to instance

    Return instance
End function

# Usage
Run GetSingleton with inputs "DatabaseConnection" save to db1
Run GetSingleton with inputs "DatabaseConnection" save to db2

# db1 and db2 are the same instance
```

Use When:
- Global configuration
- Database connections
- Logging systems
- Cache managers

C.9 Caching/Memoization Pattern

```astra
Create dictionary named FibonacciCache

To do FibonacciMemoized with inputs n
```



```
    # Check cache
    If n in FibonacciCache then
        Get n from FibonacciCache save to result
        Return result
    End check

    # Calculate
    If n is less than 2 then
        Set result to n
    Else
        Run FibonacciMemoized with inputs n - 1 save to a
        Run FibonacciMemoized with inputs n - 2 save to b
        Add a to b save to result
    End check

    # Store in cache
    Set n in FibonacciCache to result

    Return result
End function

# Without cache: Fibonacci(40) takes seconds
# With cache: Fibonacci(40) is instant
```

Generic Memoization:

```astra
To do Memoize with inputs function_name
    Create dictionary named cache
```



```
    To do MemoizedFunction with inputs args
        Convert args to string save to key

        If key in cache then
            Get key from cache save to result
            Return result
        End check

        Run function_name with inputs args save to result
        Set key in cache to result

        Return result
    End function

    Return MemoizedFunction
End function
```

Use When:

- Expensive calculations
- Recursive algorithms
- Repeated database queries
- API responses

C.10 Pagination Pattern

```astra
To do GetPaginatedResults with inputs dataset, page_number,
page_size
    # Calculate start and end indices
```



```
    Multiply page_number by page_size save to start_index
    Add page_size to start_index save to end_index

    # Get slice of data
    Get items start_index to end_index from dataset save to
page_data

    # Calculate metadata
    Get length of dataset save to total_items
    Divide total_items by page_size save to total_pages
    Round total_pages up save to total_pages

    # Build response
    Create dictionary named result
    Set "data" in result to page_data
    Set "page" in result to page_number
    Set "page_size" in result to page_size
    Set "total_items" in result to total_items
    Set "total_pages" in result to total_pages

    # Add navigation flags
    If page_number is greater than 0 then
        Set "has_previous" in result to True
    Else
        Set "has_previous" in result to False
    End check

    If page_number is less than total_pages minus 1 then
        Set "has_next" in result to True
```



```
    Else
        Set "has_next" in result to False
    End check

    Return result
End function

# Usage
Create list named all_users  # 1000 users
Run GetPaginatedResults with inputs all_users, 0, 20 save to page1
Run GetPaginatedResults with inputs all_users, 1, 20 save to page2
```

Use When:

- Large datasets
- API responses
- Database queries
- UI lists

C.11 Data Processing Pipeline

```astra
To do ProcessPipeline with inputs data
    # Stage 1: Validate
    Run ValidateData with inputs data save to validated_data
    Write "✓ Stage 1: Validation complete"

    # Stage 2: Clean
    Run CleanData with inputs validated_data save to cleaned_data
```



```
    Write "✓ Stage 2: Cleaning complete"

    # Stage 3: Transform
    Run TransformData with inputs cleaned_data save to
transformed_data
    Write "✓ Stage 3: Transformation complete"

    # Stage 4: Enrich
    Run EnrichData with inputs transformed_data save to
enriched_data
    Write "✓ Stage 4: Enrichment complete"

    # Stage 5: Aggregate
    Run AggregateData with inputs enriched_data save to final_data
    Write "✓ Stage 5: Aggregation complete"

    Return final_data
End function
# With error handling
To do RobustPipeline with inputs data
    Create list named stages
    Add "ValidateData" to stages
    Add "CleanData" to stages
    Add "TransformData" to stages
    Add "EnrichData" to stages
    Add "AggregateData" to stages

    Set current_data to data
```



```
    Repeat for every stage in stages using index
        Attempt
            Run stage with inputs current_data save to current_data
            Write "✓ Stage "
            Write index + 1
            Write ": "
            Write stage
            Write " complete"
        Recover
            Write "✗ Stage "
            Write index + 1
            Write " failed: "
            Write error_message
            Raise error "Pipeline failed at stage {stage}"
        End attempt
    End repeat

    Return current_data
End function
```

Use When:

- ETL processes
- Data transformations
- Multi-step workflows
- Processing batches

C.12 State Machine Pattern

```astra
Define Blueprint named OrderStateMachine
    Has state set to "pending"
    Has transitions set to Nothing

    Action Initialize
        Create dictionary named trans

        # Define valid transitions
        Set "pending" in trans to ["confirmed", "cancelled"]
        Set "confirmed" in trans to ["shipped", "cancelled"]
        Set "shipped" in trans to ["delivered", "returned"]
        Set "delivered" in trans to ["completed", "returned"]
        Set "returned" in trans to ["refunded"]
        Set "cancelled" in trans to []
        Set "completed" in trans to []
        Set "refunded" in trans to []

        Set transitions to trans
    End Action

    Action GetState
        Return state
    End Action

    Action CanTransitionTo with inputs new_state
        Get state from transitions save to valid_states
```



```
        If new_state in valid_states then
            Return True
        Else
            Return False
        End check
    End Action

    Action TransitionTo with inputs new_state
        Run CanTransitionTo with inputs new_state save to is_valid

        If not is_valid then
            Raise error "Invalid transition from {state} to
{new_state}"
        End check

        Write "Transitioning from "
        Write state
        Write " to "
        Write new_state

        Set state to new_state
    End Action
End Blueprint

# Usage
Create object order from OrderStateMachine
Run Initialize in order

Run TransitionTo in order with inputs "confirmed"  # OK
```



```
Run TransitionTo in order with inputs "shipped"    # OK
Run TransitionTo in order with inputs "pending"    # ERROR: Invalid transition
```

Use When:

- Order processing
- Workflow management
- Game states
- Protocol implementations

C.13 Command Pattern

```astra
Define Blueprint named Command
    Action Execute
        # Override in subclass
    End Action

    Action Undo
        # Override in subclass
    End Action
End Blueprint

Define Blueprint named AddCommand
    Has receiver set to Nothing
    Has value set to Nothing
    Has previous_value set to Nothing
```



```
    Action Initialize with inputs target, amount
        Set receiver to target
        Set value to amount
    End Action

    Action Execute
        Get current_value from receiver save to previous_value
        Add value to current_value save to new_value
        Set current_value in receiver to new_value
    End Action

    Action Undo
        Set current_value in receiver to previous_value
    End Action
End Blueprint

# Command invoker with history
Define Blueprint named CommandManager
    Has history set to Nothing
    Has current_index set to -1

    Action Initialize
        Create list named new_history
        Set history to new_history
    End Action

    Action ExecuteCommand with inputs command
        # Remove any commands after current position
        Loop while length of history is greater than current_index+1
```



```
            Remove last item from history
        End loop

        # Execute and add to history
        Run Execute in command
        Add command to history
        Add 1 to current_index
    End Action

    Action Undo
        If current_index is less than 0 then
            Write "Nothing to undo"
            Return
        End check

        Get item current_index from history save to command
        Run Undo in command
        Subtract 1 from current_index
    End Action

    Action Redo
        Add 1 to current_index save to next_index

        If next_index is greater than or equal to length of history
then
            Set current_index to next_index - 1
            Write "Nothing to redo"
            Return
        End check
```



```
        Set current_index to next_index
        Get item current_index from history save to command
        Run Execute in command
    End Action
End Blueprint
```

Use When:

- Undo/redo functionality
- Transaction management
- Job queues
- Macro recording

C.14 Circuit Breaker Pattern

```astra
Define Blueprint named CircuitBreaker
    Has state set to "closed"
    Has failure_count set to 0
    Has failure_threshold set to 5
    Has timeout set to 60
    Has last_failure_time set to Nothing

    Action Call with inputs function, args
        # Check if circuit is open
        If state is "open" then
            Get current time save to now
            Subtract last_failure_time from now save to elapsed
```



```
            If elapsed is greater than timeout then
                Write "Circuit breaker: attempting reset"
                Set state to "half-open"
            Else
                Raise error "Circuit breaker is OPEN"
            End check
        End check

        # Attempt call
        Attempt
            Run function with inputs args save to result

            # Success - reset if half-open
            If state is "half-open" then
                Write "Circuit breaker: reset to CLOSED"
                Set state to "closed"
                Set failure_count to 0
            End check

            Return result

        Recover
            # Failure - increment counter
            Add 1 to failure_count
            Get current time save to last_failure_time

            If failure_count is greater than or equal to
failure_threshold then
```



```
                Write "Circuit breaker: opening circuit"
                Set state to "open"
            End check

            Raise error error_message
        End attempt
    End Action
End Blueprint
```

Use When:

- Microservices communication
- External API calls
- Database connections
- Preventing cascading failures

Appendix D: Glossary of Terms

A

Abstract Syntax Tree (AST)

Internal representation of source code as a tree structure, where each node represents a construct in the code.

Activation Function

Non-linear function in neural networks that determines neuron output (e.g., ReLU, sigmoid, tanh).

Algorithm

Step-by-step procedure for solving a problem or performing a computation.

API (Application Programming Interface)

Set of rules and protocols for building and interacting with software applications.

Argument

Actual value passed to a function when called. See also: Parameter.

Array

Collection of elements stored at contiguous memory locations. In Astra, see: List.

Assertion

Statement that a condition must be true; program fails if false.

Assignment

Operation that binds a value to a variable name.

Asynchronous

Execution model where operations can proceed without waiting for other operations to complete.

Attribute

Property or characteristic of an object.

B

Backpropagation

Algorithm for training neural networks by propagating errors backward through layers.

Batch

Subset of training data processed together before updating model weights.

Binary Search

Efficient algorithm for finding element in sorted array (O(log n) time).

Binary Tree

Tree where each node has at most two children (left and right).

Binding

Association between a name and a value in memory.

Block

Group of statements enclosed by control structure (if, loop, function).

Blueprint

Astra's term for class—template for creating objects.

Boolean

Data type with two values: True or False.

Breadth-First Search (BFS)

Graph traversal visiting all neighbors before going deeper.

Breakpoint

Marker that pauses program execution for debugging.

Bug

Error or flaw in code causing incorrect behavior.

C

Cache

Temporary storage for frequently accessed data to improve performance.

Callback

Function passed as argument to be executed later.

Call Stack

Data structure tracking active function calls during execution.

Class

Template for creating objects with shared properties and methods. See: Blueprint.

Classifier

ML model that assigns input to predefined categories.

Closure

Function that captures variables from enclosing scope.

Clustering

Unsupervised ML technique grouping similar data points.

CNN (Convolutional Neural Network)

Neural network architecture specialized for image processing.

Code Coverage

Percentage of code executed by tests.

Collection

Data structure holding multiple elements (list, set, dictionary).

Compiler

Program that translates source code to machine code.

Complexity

Measure of resources (time/space) required by algorithm.

Composite Type

Data type containing multiple values (list, dictionary). Opposite: Primitive type.

Concatenation

Joining strings or sequences together.

Concurrency

Multiple tasks making progress simultaneously.

Condition

Expression evaluating to True or False.

Constructor

Special method for initializing new objects.

Context Manager

Object managing resource allocation and cleanup (Using blocks in Astra).

Correlation

Statistical relationship between variables.

Cross-Validation

Technique for assessing ML model performance on different data subsets.

CSV (Comma-Separated Values)

Text format for tabular data.

Cyclomatic Complexity

Metric measuring number of independent paths through code.

D

Data Structure

Organized format for storing and managing data.

DataFrame

Two-dimensional labeled data structure (pandas).

Dataset

Collection of data for analysis or training.

Debugging

Process of finding and fixing bugs.

Declaration

Statement introducing new variable, function, or type.

Deep Learning

Machine learning using neural networks with multiple layers.

Dependency

External code or library required by program.

Deployment

Making software available for use in production.

Depth-First Search (DFS)

Graph traversal going as deep as possible before backtracking.

Dictionary

Data structure mapping keys to values.

Directed Graph

Graph where edges have direction.

Dropout

Regularization technique randomly disabling neurons during training.

E

Edge

Connection between two vertices in a graph.

Element

Individual item in collection.

Encapsulation

Bundling data and methods that operate on that data.

Encoding

Converting data from one format to another.

Ensemble

ML technique combining multiple models.

Entropy

Measure of randomness or impurity.

Epoch

One complete pass through entire training dataset.

Error

Deviation from expected behavior or result.

Event

Occurrence triggering specific action.

Exception

Error disrupting normal program flow.

Expression

Combination of values, variables, and operators producing a result.

F

Feature

Individual measurable property used in ML.

Feature Engineering

Creating new features from existing data to improve model performance.

FIFO (First In, First Out)

Queue processing order.

Filter

Function selecting subset of data based on condition.

Float

Floating-point number (decimal).

Function

Named block of reusable code.

Functional Programming

Programming paradigm treating computation as evaluation of functions.

G

Garbage Collection

Automatic memory management reclaiming unused objects.

GAN (Generative Adversarial Network)

Neural network architecture for generating new data.

Generator

Function producing sequence of values lazily.

Git

Version control system for tracking code changes.

Global Scope

Variables accessible throughout entire program.

GPU (Graphics Processing Unit)

Specialized processor for parallel computations (ML acceleration).

Gradient

Direction and rate of steepest increase of function.

Graph

Data structure of vertices connected by edges.

H

Hash

Fixed-size value produced by hash function.

Hash Table

Data structure implementing associative array.

Heatmap

Data visualization showing magnitude with colors.

Heuristic

Practical approach or rule of thumb for problem-solving.

Hidden Layer

Intermediate layer in neural network between input and output.

Hyperparameter

Parameter controlling learning process (not learned from data).

I

Identifier

Name given to variable, function, or other entity.

Immutable

Cannot be changed after creation.

Import

Statement making external code available.

Index

Position of element in sequence (zero-based in Astra).

Inference

Using trained model to make predictions on new data.

Inheritance

Mechanism where class derives properties from parent class.

Integer

Whole number (positive, negative, or zero).

Integration Test

Testing multiple components together.

Interpreter

Program executing code directly without compilation.

Iteration

Single pass through loop body.

Iterator

Object enabling traversal through collection.

J

JSON (JavaScript Object Notation)

Text format for data interchange.

K

Key

Unique identifier in dictionary.

Keyword

Reserved word with special meaning in language.

L

Label

Target value in supervised learning.

Lambda

Anonymous function (not directly supported in Astra).

Latency

Time delay between cause and effect.

Layer

Component in neural network processing inputs to outputs.

Learning Rate

Hyperparameter controlling weight update size.

Lexical Scope

Scope determined by code structure, not execution.

Library

Collection of reusable code.

LIFO (Last In, First Out)

Stack processing order.

Linear Regression

ML algorithm modeling linear relationship between variables.

List

Ordered, mutable collection of elements.

Literal

Fixed value in code (e.g., 42, "hello", True).

Local Scope

Variables only accessible within specific block or function.

Logging

Recording program events for monitoring and debugging.

Loop

Control structure executing code repeatedly.

Loss Function

Measures prediction error in ML model.

M

Machine Learning

AI technique where systems learn patterns from data.

Map

Function applying operation to each element in collection.

Matrix

Two-dimensional array of numbers.

Memoization

Optimization caching function results.

Method

Function belonging to object or class.

ML (Machine Learning)

See: Machine Learning.

Model

Mathematical representation learned from data.

Module

File containing related code.

MST (Minimum Spanning Tree)

Subset of graph edges connecting all vertices with minimum total weight.

Mutable

Can be changed after creation.

N

Neural Network

ML model inspired by biological neurons.

Node

Element in tree or graph; also neuron in neural network.

Nothing

Astra value representing absence of value (similar to null/None).

NLP (Natural Language Processing)

AI field focused on language understanding and generation.

O

Object

Instance of class/blueprint with properties and methods.

OOP (Object-Oriented Programming)

Programming paradigm based on objects.

Optimizer

Algorithm updating model weights during training.

Overfitting

Model performs well on training data but poorly on new data.

P

Package

Collection of modules.

Parameter

Variable in function definition receiving argument values.

Parse

Analyzing code structure according to grammar rules.

Path

Sequence of edges connecting vertices in graph.

Pipeline

Series of processing stages.

Pointer

Reference to memory location.

Polymorphism

Ability to process objects differently based on type.

Precision

ML metric: proportion of positive predictions that are correct.

Prediction

Output produced by trained model.

Primitive Type

Basic, immutable data type (integer, float, string, boolean).

Profiling

Measuring program performance.

Property

Attribute of object.

Q

Queue

FIFO data structure.

Quantization

Reducing model precision to decrease size.

R

Recursion

Function calling itself.

Refactoring

Restructuring code without changing behavior.

Reference

Pointer to object in memory.

Regression

ML task predicting continuous values.

Regularization

Technique preventing overfitting.

Reinforcement Learning

ML where agent learns through rewards.

Return

Statement sending value back to function caller.

RNN (Recurrent Neural Network)

Neural network for sequential data.

ROC Curve

Plot of true positive rate vs false positive rate.

S

Scope

Region where name is valid and visible.

Semantics

Meaning of code (what it does).

Set

Unordered collection of unique elements.

Shadowing

Inner scope variable hiding outer scope variable with same name.

Singleton

Design pattern ensuring only one instance of class.

Softmax

Activation function converting scores to probabilities.

Stack

LIFO data structure; also call stack tracking function calls.

Statement

Complete unit of execution.

String

Sequence of characters.

Supervised Learning

ML learning from labeled examples.

Syntax

Rules governing code structure.

T

TDD (Test-Driven Development)

Writing tests before implementation code.

Tensor

Multi-dimensional array (used in deep learning).

Test Coverage

See: Code Coverage.

Throughput

Amount of work completed per unit time.

Token

Smallest unit of meaning in code.

Training

Process of teaching ML model from data.

Transfer Learning

Using pre-trained model for new task.

Traversal

Visiting all nodes in tree or graph.

Tree

Hierarchical data structure with root and children.

Tuple

Immutable ordered collection.

Type

Category of values with defined operations.

U

Underfitting

Model too simple to capture data patterns.

Unit Test

Test of individual function in isolation.

Unsupervised Learning

ML finding patterns in unlabeled data without predefined outputs.

URL (Uniform Resource Locator)

Address for accessing web resources.

Use Case

Specific situation where system or feature is used.

UTF-8

Character encoding supporting all Unicode characters.

V

Validation

Process of checking correctness, completeness, or compliance.

Validation Set

Data used to tune model hyperparameters during training.

Value

Data stored in variable or returned by expression.

Variable

Named storage location for value.

Vector

One-dimensional array of numbers.

Vertex

Node in graph.

Visualization

Graphical representation of data.

W

Weight

Learned parameter in neural network controlling connection strength.

Weighted Graph

Graph where edges have associated numerical values.

While Loop

Loop executing while condition is true.

X

XOR (Exclusive OR)

Logical operation true when inputs differ.

XML (eXtensible Markup Language)

Markup language for encoding documents.

Y

YAML (YAML Ain't Markup Language)

Human-readable data serialization format.

Z

Zero-based Indexing

Numbering system where first element has index 0.

Appendix E: Exercise Solutions

Chapter 1: Philosophy of Astra

Beginner Exercise

Rewrite: `total = total * 2`

Solution:

```astra
Multiply total by 2
```

Explanation: The Astra version explicitly states the action (multiply) and what is being modified (total), making the intent clear without requiring understanding of variable assignment semantics.

Intermediate Exercise

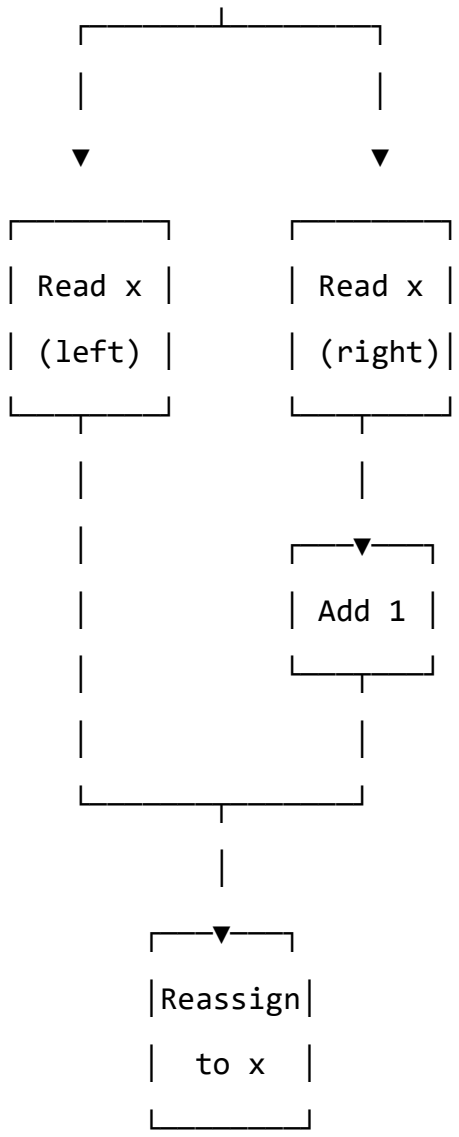
Draw diagram showing hidden operations in expression

Solution:

Traditional expression: `x = x + 1`

Diagram:

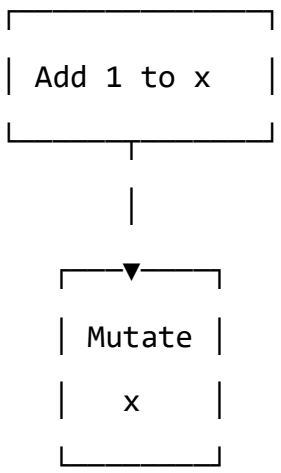
```
┌─────────────────────────────────────┐
│  Expression: x = x + 1              │
└─────────────────────────────────────┘
            │
```

Hidden operations:

1. Dereference x (read current value)
2. Add 1 to that value
3. Store result back to x
4. Deallocate old value

```

**Astra version: `Add 1 to x`**

```

Single, explicit operation

```

**Advanced Exercise**

**Explain benefits for AI-generated code**

**Solution:**

**Why Explicit Intent Benefits AI:**

1. **Reduced Ambiguity**

   - Traditional: `x = x + 1` could mean increment, reassignment, or initialization

   - Astra: `Add 1 to x` unambiguously means mutation

   - AI models trained on natural language understand "Add" better than `=`

2. **Semantic Clarity**

   - AI doesn't need to understand operator precedence

   - Each line maps to single semantic action

   - Natural language commands match AI training data

 
```

3. **Verification**

   - AI can verify correctness by reading code aloud

   - Explicit blocks (`End check`, `End loop`) prevent structure errors

   - No invisible scope rules to get wrong

4. **Fewer Edge Cases**

   - No confusion between `=` (assignment) and `==` (comparison)

   - No implicit type conversions

   - No silent failures

5. **Code Generation Accuracy**

   - Language models generate Astra more reliably

   - Less "hallucination" of incorrect syntax

   - Easier to map natural language requests to Astra code

Example:

User: "Add 5 to the score"

Traditional (Python):

```python
score = score + 5  # Requires understanding assignment
score += 5         # Requires knowing shorthand
```

Astra:

```astra
Add 5 to score     # Direct mapping from user intent
```


Chapter 3: Variables & Scope

Beginner Exercise

Draw diagram: name vs value

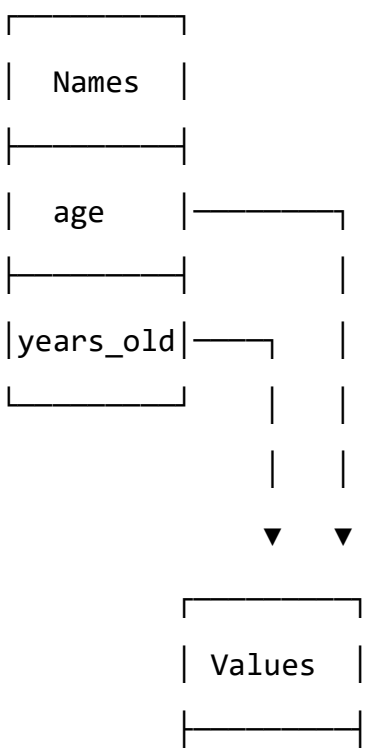
Solution:

```astra
Set age to 25
Set years_old to age
```

Diagram:

```
```

Memory Layout:




```
          │   25    │
          └─────────┘
```

Key Points:

- "age" is a NAME (label/identifier)
- 25 is a VALUE (data in memory)
- "years_old" is another NAME
- Both names point to the SAME value
- Names are NOT containers

```

**After mutation:**

```astra
Add 1 to age
```

```

Memory Layout:

```
┌─────────┐
│ Names │
├─────────┤
│ age   │─────────┐
├─────────┤   │
│years_old│──┐   │
└─────────┘ │   │
             │   │
             ▼   ▼
          ┌──────────┐
          │ Values │
```


```
          ├──────────────┤
          │   25         │ (still referenced by years_old)
          ├──────────────┤
          │   26         │ (new value for age)
          └──────────────┘
```

Intermediate Exercise

Predict output with nested scopes

Given:

```astra
Set x to 10
If x is greater than 5 then
    Set x to 20
    Set y to 30
    Write x
    Write y
End check
Write x
Write y
```

Solution:

Output:

```
20
30
```



```
10
ERROR: y is not defined
```

Explanation:

Scope Analysis:

```
Global Scope:
├─ x = 10 (created)
│
└─ If Block Scope:
    ├─ x = 20 (shadows global x)
    └─ y = 30 (local to If block)

After If block:
├─ x = 10 (global x restored)
└─ y = undefined (destroyed when block ended)
```

Step-by-step:

1. Line 1: Set global `x` to 10
2. Line 2: Condition true (10 > 5), enter If block
3. Line 3: Create local `x` = 20 (shadows global)
4. Line 4: Create local `y` = 30
5. Line 5: Write local `x` → Output: **20**
6. Line 6: Write local `y` → Output: **30**
7. Line 7: Exit If block (local `x` and `y` destroyed)
8. Line 8: Write global `x` → Output: **10**
9. Line 9: Write `y` → **ERROR** (y doesn't exist in global scope)

Advanced Exercise

Design unsafe scope rule and explain

Solution:

Unsafe Rule: "Variables leak into outer scope"

How it would work:

Variables created inside blocks remain accessible after block ends.

```astra
# Hypothetical unsafe Astra
If True then
    Set leaked_variable to 100
End check

Write leaked_variable  # Would work under unsafe rule
```

Problems this causes:

1. **Namespace Pollution**

```astra
If some_condition then
    Set temp to calculate_something
End check

If other_condition then
    Set temp to other_calculation  # Accidentally overwrites previous temp
```



```
End check

Write temp  # Which temp? Unpredictable!
```

2. **Hidden Dependencies**

```astra
To do ProcessData
    If data is valid then
        Set result to "processed"
    End check
    # ... 100 lines later ...
    Write result  # Depends on distant condition
End function
```

3. **Memory Leaks**

```astra
Repeat 1000 times
    Set large_array to create_big_data
    # Process array
End repeat
# All 1000 arrays still in memory!
```

4. **Race Conditions**

```astra
# Thread 1
If condition_a then
```



```
    Set shared_var to value_a
End check

# Thread 2
If condition_b then
    Set shared_var to value_b
End check

# Who won? Unpredictable!
```

5. **Maintenance Nightmares**

```astra
To do ComplexFunction
    If x is greater than 0 then
        If y is greater than 0 then
            Set result to calculate_a
        Else
            Set result to calculate_b
        End check
    Else
        Set result to calculate_c
    End check

    Return result  # Did we always set result?
End function
```


Why Astra's Rules Prevent This:

Astra's explicit scope rules:
- Variables are destroyed when block ends
- Forces explicit passing of data
- Makes dependencies visible
- Prevents accidental sharing
- Enables safe parallelism

Safe Astra Version:

```astra
To do ProcessData with inputs data
    Set result to Nothing

    If data is valid then
        Set result to "processed"
    End check

    If result is Nothing then
        Raise error "Data was not processed"
    End check

    Return result
End function
```

Chapter 8: Functions

Beginner Exercise

Celsius to Fahrenheit converter

Solution:

```astra
To do CelsiusToFahrenheit with inputs celsius
    # Formula: F = C × 9/5 + 32
    Multiply celsius by 9 save to temp
    Divide temp by 5 save to temp
    Add 32 to temp save to fahrenheit
    Return fahrenheit
End function

# Test cases
Run CelsiusToFahrenheit with inputs 0 save to result
Write "0°C = "
Write result
Write "°F"  # Output: 0°C = 32°F

Run CelsiusToFahrenheit with inputs 100 save to result
Write "100°C = "
Write result
Write "°F"  # Output: 100°C = 212°F

Run CelsiusToFahrenheit with inputs 37 save to result
Write "37°C = "
Write result
Write "°F"  # Output: 37°C = 98.6°F
```


Intermediate Exercise

Find largest number in list

Solution:

```astra
To do FindLargest with inputs numbers
    # Handle empty list
    If length of numbers is 0 then
        Raise error "Cannot find largest in empty list"
    End check

    # Initialize with first element
    Get item 0 from numbers save to largest

    # Check rest of elements
    Repeat for every num in numbers
        If num is greater than largest then
            Set largest to num
        End check
    End repeat

    Return largest
End function

# Test cases
Create list named test1
Add 5 to test1
Add 2 to test1
```



```
Add 9 to test1
Add 1 to test1
Add 7 to test1

Run FindLargest with inputs test1 save to result
Write "Largest: "
Write result  # Output: 9

# Test with negative numbers
Create list named test2
Add -5 to test2
Add -2 to test2
Add -9 to test2

Run FindLargest with inputs test2 save to result
Write "Largest: "
Write result  # Output: -2

# Test with single element
Create list named test3
Add 42 to test3

Run FindLargest with inputs test3 save to result
Write "Largest: "
Write result  # Output: 42

# Test empty list
Create list named test4
Attempt
```



```
    Run FindLargest with inputs test4
Recover
    Write "Correctly handled empty list: "
    Write error_message
End attempt
```

Advanced Exercise

Fibonacci with recursion, call stack, and optimization

Solution:

1. Recursive Version:

```astra
To do FibonacciRecursive with inputs n
    If n is less than or equal to 1 then
        Return n
    End check

    Run FibonacciRecursive with inputs n - 1 save to fib_n_minus_1
    Run FibonacciRecursive with inputs n - 2 save to fib_n_minus_2
    Add fib_n_minus_1 to fib_n_minus_2 save to result

    Return result
End function
```

Call Stack for Fibonacci(5):

```
Call Stack Visualization:
```



```
Main()
  │
  └─► FibonacciRecursive(5)
       ├─► FibonacciRecursive(4)
       │    ├─► FibonacciRecursive(3)
       │    │    ├─► FibonacciRecursive(2)
       │    │    │    ├─► FibonacciRecursive(1) → returns 1
       │    │    │    └─► FibonacciRecursive(0) → returns 0
       │    │    │    Result: 1
       │    │    └─► FibonacciRecursive(1) → returns 1
       │    │    Result: 2
       │    └─► FibonacciRecursive(2)
       │         ├─► FibonacciRecursive(1) → returns 1
       │         └─► FibonacciRecursive(0) → returns 0
       │         Result: 1
       │    Result: 3
       └─► FibonacciRecursive(3)
            ├─► FibonacciRecursive(2)
            │    ├─► FibonacciRecursive(1) → returns 1
            │    └─► FibonacciRecursive(0) → returns 0
            │    Result: 1
            └─► FibonacciRecursive(1) → returns 1
            Result: 2
       Result: 5

Total function calls: 15
Notice: Fibonacci(2) calculated 3 times!
```


Time Complexity: O(2^n) - exponential, very slow

2. Optimized with Memoization:

```astra
Create dictionary named FibCache

To do FibonacciMemoized with inputs n
    # Check cache first
    If n in FibCache then
        Get n from FibCache save to cached_result
        Return cached_result
    End check

    # Base case
    If n is less than or equal to 1 then
        Set n in FibCache to n
        Return n
    End check

    # Recursive case with caching
    Run FibonacciMemoized with inputs n - 1 save to fib_n_minus_1
    Run FibonacciMemoized with inputs n - 2 save to fib_n_minus_2
    Add fib_n_minus_1 to fib_n_minus_2 save to result

    Set n in FibCache to result
    Return result
End function
```


Time Complexity: O(n) - linear, much faster

3. Iterative Version (Most Efficient):

```astra
To do FibonacciIterative with inputs n
    If n is less than or equal to 1 then
        Return n
    End check

    Set prev to 0
    Set curr to 1

    Repeat n - 1 times
        Add prev to curr save to next
        Set prev to curr
        Set curr to next
    End repeat

    Return curr
End function
```

Time Complexity: O(n) - linear, no recursion overhead

Space Complexity: O(1) - constant space

Performance Comparison:

```astra
# Test Fibonacci(30)

Start timer
```



```
Run FibonacciRecursive with inputs 30 save to result
Stop timer save to recursive_time
Write "Recursive: "
Write recursive_time
Write " seconds"
# Output: ~5 seconds

Start timer
Run FibonacciMemoized with inputs 30 save to result
Stop timer save to memoized_time
Write "Memoized: "
Write memoized_time
Write " seconds"
# Output: ~0.001 seconds

Start timer
Run FibonacciIterative with inputs 30 save to result
Stop timer save to iterative_time
Write "Iterative: "
Write iterative_time
Write " seconds"
# Output: ~0.0001 seconds
```

Chapter 16: Machine Learning

Beginner Exercise

Train first ML model on iris dataset

Solution:

```astra
Use python library sklearn.datasets as datasets
Use python library sklearn.model_selection as model_selection
Use python library sklearn.tree as tree
Use python library sklearn.neighbors as neighbors
Use python library sklearn.svm as svm

# Load iris dataset
Load iris dataset save to iris_data
Get data from iris_data save to X
Get target from iris_data save to y

# Split data
Split X, y into train and test with ratio 0.8 save to X_train, X_test, y_train, y_test

# Algorithm 1: Decision Tree
Create predictor dt using DecisionTreeClassifier
Train dt using X_train to predict y_train
Predict using dt with inputs X_test save to pred_dt
Calculate accuracy between y_test and pred_dt save to acc_dt

# Algorithm 2: K-Nearest Neighbors
Create predictor knn using KNeighborsClassifier with k 3
Train knn using X_train to predict y_train
Predict using knn with inputs X_test save to pred_knn
Calculate accuracy between y_test and pred_knn save to acc_knn
```



```
# Algorithm 3: Support Vector Machine
Create predictor svm_model using SVC
Train svm_model using X_train to predict y_train
Predict using svm_model with inputs X_test save to pred_svm
Calculate accuracy between y_test and pred_svm save to acc_svm

# Results
Write "===== Model Comparison ====="
Write "Decision Tree: "
Write acc_dt

Write "K-Nearest Neighbors: "
Write acc_knn

Write "SVM: "
Write acc_svm

# Identify best
Create list named accuracies
Add acc_dt to accuracies
Add acc_knn to accuracies
Add acc_svm to accuracies

Get maximum from accuracies save to best_acc
Write "
Best accuracy: "
Write best_acc
```


Chapter 20: Testing & Debugging

Beginner Exercise

Palindrome checker with tests

Solution:

```astra
To do IsPalindrome with inputs text
    # Remove spaces and convert to lowercase
    Replace " " with "" in text save to cleaned
    Convert cleaned to lowercase save to cleaned

    # Compare with reverse
    Reverse cleaned save to reversed

    If cleaned is equal to reversed then
        Return True
    Else
        Return False
    End check
End function

To do Test_IsPalindrome
    Write "Running palindrome tests..."

    # Test 1: Simple palindrome
    Run IsPalindrome with inputs "racecar" save to result
```



```
    Verify that result is True with message "Test 1 failed: racecar"
    Write "✓ Test 1: Simple palindrome"

    # Test 2: Single character
    Run IsPalindrome with inputs "a" save to result
    Verify that result is True with message "Test 2 failed: single
char"
    Write "✓ Test 2: Single character"

    # Test 3: Empty string
    Run IsPalindrome with inputs "" save to result
    Verify that result is True with message "Test 3 failed: empty"
    Write "✓ Test 3: Empty string"

    # Test 4: Non-palindrome
    Run IsPalindrome with inputs "hello" save to result
    Verify that result is False with message "Test 4 failed: non-
palindrome"
    Write "✓ Test 4: Non-palindrome"

    # Test 5: Palindrome with spaces
    Run IsPalindrome with inputs "A man a plan a canal Panama" save
to result
    Verify that result is True with message "Test 5 failed: with
spaces"
    Write "✓ Test 5: Palindrome with spaces"

    # Test 6: Mixed case
    Run IsPalindrome with inputs "RaceCar" save to result
```



```
    Verify that result is True with message "Test 6 failed: mixed
case"

    Write "✓ Test 6: Mixed case"


    # Test 7: Palindrome with punctuation (should fail with current
implementation)

    Run IsPalindrome with inputs "race-car" save to result

    Verify that result is False with message "Test 7: punctuation
not handled"

    Write "✓ Test 7: Punctuation (not handled)"


    Write "===== All tests passed! ====="

End function


Run Test_IsPalindrome
```

Enhanced Version (handles punctuation):

```astra
To do IsPalindromeAdvanced with inputs text

    # Remove all non-alphanumeric characters

    Set cleaned to ""


    Repeat for every char in text

        If char is alphanumeric then

            Convert char to lowercase save to lower_char

            Add lower_char to cleaned

        End check

    End repeat
```



```
    # Compare with reverse
    Reverse cleaned save to reversed

    Return cleaned is equal to reversed
End function
```

Summary of Key Learnings

Programming Fundamentals

- **Variables are names, not containers** - they bind to values
- **Scope determines visibility** - lexical scope prevents surprises
- **Functions enable reuse** - write once, use many times
- **Objects bundle data and behavior** - encapsulation

Data Structures

- **Choose the right structure** - list, dict, set, stack, queue, tree, graph
- **Understand time complexity** - O(1), O(log n), O(n), O(n²)
- **Reference vs value semantics** - prevents subtle bugs

Machine Learning

- **Quality data > fancy algorithms** - garbage in, garbage out
- **Always split data** - train/test prevents overfitting
- **Feature engineering matters** - often provides biggest gains
- **Start simple** - baseline before complexity

Software Engineering

- **Testing is not optional** - untested code doesn't work

- **Explicit is better than implicit** - reduces cognitive load
- **Fail fast and loudly** - don't hide errors
- **Document why, not what** - code shows what, comments explain why

---

Conclusion

Congratulations! You've completed the Astra Programming guide. You now understand:

1. **Core Programming Concepts** - variables, functions, control flow
2. **Data Structures** - organizing and accessing data efficiently
3. **Object-Oriented Programming** - modeling real-world entities
4. **Algorithms** - solving problems systematically
5. **Machine Learning** - teaching computers to learn from data
6. **Software Engineering** - writing maintainable, tested code

Next Steps:

- Build real projects using Astra
- Contribute to the Astra community
- Teach others what you've learned
- Keep practicing and experimenting

Remember: Programming is about solving problems, not memorizing syntax. Astra removes syntax barriers so you can focus on what matters: **clear thinking and effective solutions.**

Index

END OF ASTRA PROGRAMMING: THE DEFINITIVE GUIDE

Version 1.0

For updates, community, and resources:

- **Website**: https://astra-lang.org
- **Documentation**: https://docs.astra-lang.org
- **GitHub**: https://github.com/astra-lang
- **Community**: https://community.astra-lang.org

Happy Coding!

By Syed Ismail Nasser ☺

-B.E Robotics and Automation

