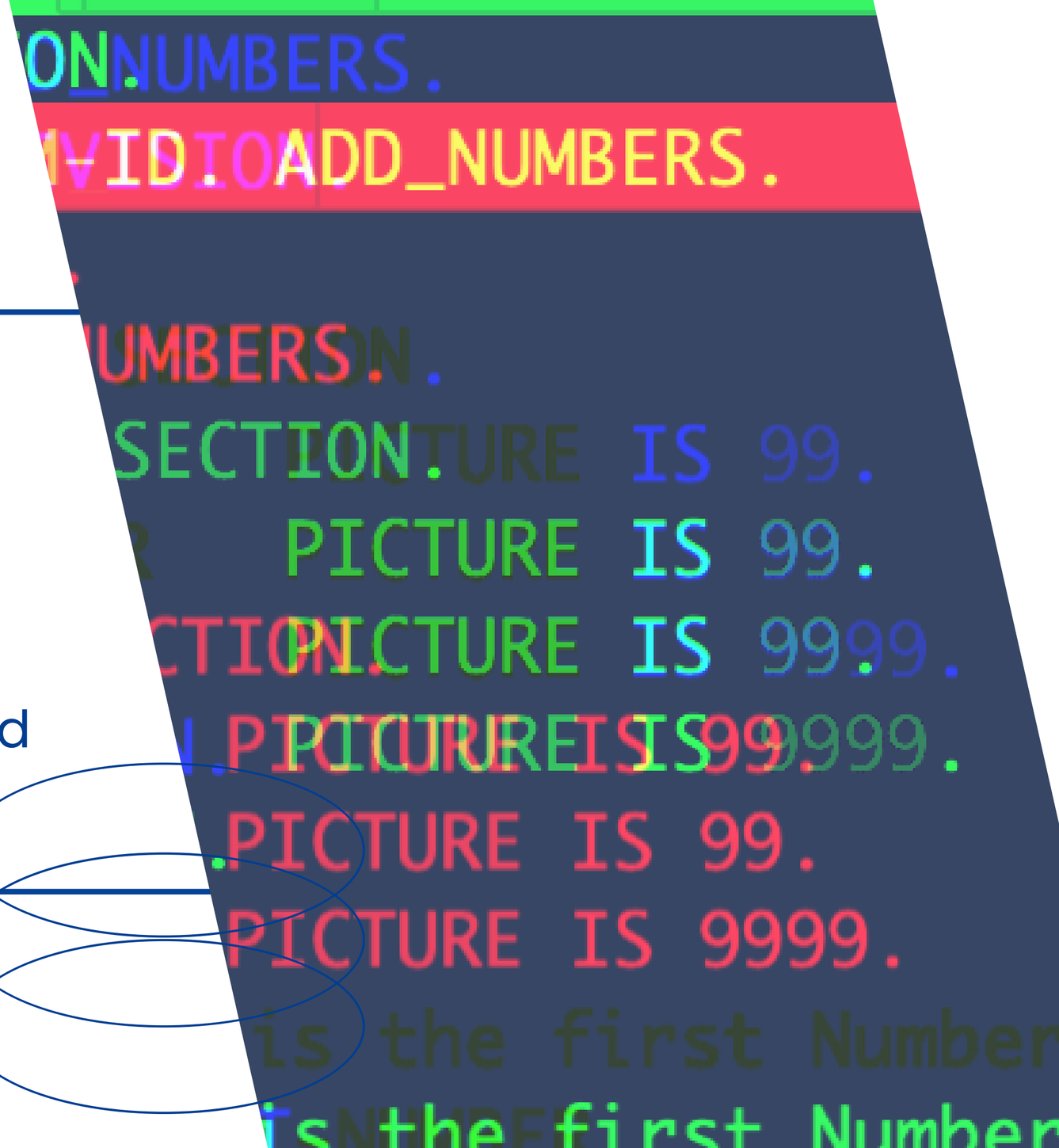
COBOL

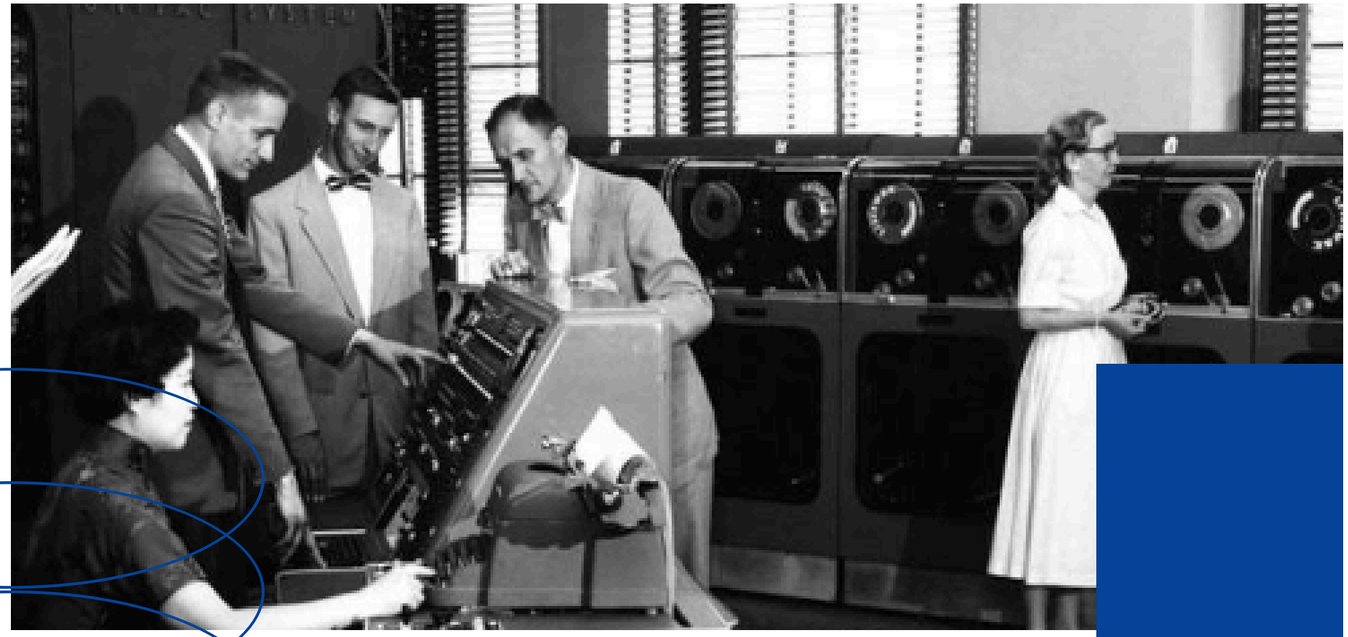
Business Logic That Runs the World

Karen Arteaga Mendoza



WHY COBOL?

Fun Fact: COBOL is older than the Moon Landing!



CODASYL

CREATED IN 1959 TO HELP BUSINESSES PROCESS DATA

MADE FOR READABILITY, PORTABILITY, AND LONG-TERM USE

STILL POWERS BANKING, INSURANCE, AND GOVERNMENT SYSTEMS

Y2K + COBOL
WHAT'S THE BIG DEAL?

What was the Y2K problem?

Why was COBOL at the center of Y2K?

Why do you think companies didn't just rewrite their COBOL programs before the year 2000?

Y2K + COBOL WHAT HAPPENED?

COBOL programs stored years as two digits using PIC 99, which caused confusion in 2000

Teams scanned millions of lines of COBOL code to locate all date-related fields.

This required updating not just the variable but every place that used or displayed it — including screens, reports, and databases.

COBOL’S PROGRAMMING PARADIGM

Data	Declares all data items, files and structures
Procedure	Contains the executable instructions

PROCEDURAL LANGUAGE

- Focuses on how tasks are performed.
- Code is written as a sequence of instructions.
- Uses constructs like PERFORM, IF, MOVE, and STOP RUN.

DATA AND LOGIC ARE SEPARATE

- Data Division: describes what data the program uses.
- Procedure Division: describes how the program processes it.

NO OBJECTS (UNTIL 2002)

- Traditional COBOL doesn't use objects or methods.
- Everything is procedural: no encapsulation, no classes.

MODULAR THINKING IN A PROCEDURAL WORLD

```
PERFORM CALCULATE-TOTAL
...
CALCULATE-TOTAL.
    COMPUTE TOTAL = PRICE * QUANTITY.
```

COBOL SUPPORTS MODULARITY

- **Sections:** Group related paragraphs together.
- **Paragraphs:** Named blocks of logic, reusable via PERFORM.
- **PERFORM:** Like a function call — jumps to a paragraph and returns.

BENEFITS OF MODULARITY

- Easier to read, test, and maintain code.
- Encourages reuse (validation, calculations, formatting).

COBOL STRUCTURE

IDENTIFICATION DIVISION

ENVIRONMENT DIVSION

DATA DIVISION

PROCEDURE DIVISION

```
IDENTIFICATION DIVISION.
  PARAGRAPHS
    ENTRIES
      CLAUSES
ENVIRONMENT DIVISION.
  SECTIONS
    PARAGRAPHS
      ENTRIES
        CLAUSES
          PHRASES
DATA DIVISION.
  SECTIONS
    ENTRIES
      CLAUSES
        PHRASES
PROCEDURE DIVISION.
  SECTIONS
    PARAGRAPHS
      SENTENCES
        STATEMENTS
          PHRASES
```

COBOL IS BUILT FROM ENGLISH-LIKE UNITS

COBOL's syntax is highly structured and readable because it's made of logical building blocks, from smallest to largest

1

Words

- The smallest unit (MOVE, WS-VAR, 10)
- **Types:** Reserved Words, User-defined Names, Literals

2

Literals
Fixed values used in the code

3

Phrases
Groups of words that modify or support a clause

4

Clauses
Specify attributes or actions, often used in Data or Environment Divisions

5

Statements
The core instructions the program executes

6

SENTENCES: WHEN DO I USE A PERIOD IN COBOL?

A sentence is one or more statements that end with a period (.)

The Period (.) is a Full Stop

- In COBOL, a period ends a complete unit — like a sentence in English.
- **It tells the compiler**: "This instruction block is complete."

At the end of a sentence in the Procedure Division

```
DISPLAY "Hello, world".
STOP RUN.
```

Avoid periods too early in control structures

```
IF A > B.
    DISPLAY "Too early!".
```

To end a paragraph or section

```
MAIN-PARA.
    DISPLAY "Done".
    STOP RUN.
```

WHERE DO THE PERIODS GO?

```
IDENTIFICATION DIVISION
PROGRAM-ID TuitionCalc

DATA DIVISION
WORKING-STORAGE SECTION
01 STUDENT-NAME     PIC X(30)
01 COURSES          PIC 9(2)
01 FEE              PIC 9(4)V99

PROCEDURE DIVISION
MAIN-LOGIC
    DISPLAY "Enter student name:"
    ACCEPT STUDENT-NAME
    DISPLAY "Enter number of courses:"
    ACCEPT COURSES
    IF COURSES > 4
        COMPUTE FEE = COURSES * 100.00
    DISPLAY "Total tuition: $" FEE
    STOP RUN
```

The code is missing its periods. Add them only where needed — be careful, not every line requires one!

Hint: Don’t put a period after the IF line — unless you're sure the block is complete!

WHERE DO THE PERIODS GO?

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TuitionCalc.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 STUDENT-NAME     PIC X(30).
01 COURSES          PIC 9(2).
01 FEE              PIC 9(4)V99.

PROCEDURE DIVISION.
MAIN-LOGIC.
    DISPLAY "Enter student name:".
    ACCEPT STUDENT-NAME.
    DISPLAY "Enter number of courses:".
    ACCEPT COURSES.
    IF COURSES > 4
        COMPUTE FEE = COURSES * 100.00
    DISPLAY "Total tuition: $" FEE.
    STOP RUN.
```

The code is missing its periods. Add them only where needed — be careful, not every line requires one!

Why No Period After IF?

- The IF statement continues to the next line.
- A period would prematurely end the IF block, causing the next statement to always run.
- The correct approach: no period until the entire block (or sentence) ends.

IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
PROGRAM-ID. YourProgramName.
AUTHOR. Your name.
DATE-WRITTEN. YYYY-MM-DD.
COMMENT. "Description of the program".
```

Provides metadata: name, author, date, description

Required paragraph: PROGRAM-ID.

ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. IBM-Z-SERIES.
  OBJECT-COMPUTER. IBM-Z-SERIES.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
     SELECT EMP-FILE ASSIGN TO 'EMPLOYEE.DAT'.
```

System configuration and file assignments

Key sections:

- CONFIGURATION
- INPUT-OUTPUT

This division is optional, it is required if your program uses external files for reading or writing

DATA DIVISION

```
DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 EMPLOYEE-DETAILS.
    05 EMP-ID        PIC 9(5) .
    05 EMP-NAME      PIC X(30).
    05 EMP-SALARY    PIC 9(7)V99.
```

All variables and file records are declared here

Sections:

- FILE SECTION
- WORKING-STORAGE SECTION
- LOCAL-STORAGE SECTION
- LINKAGE SECTION

PIC CLAUSE

```
PIC <Type-Character>(<Size>)
```

X	Alphanumeric (letters, numbers, symbols)
9	Numeric digits only (0-9)
A	Alphabetic characters (A-Z)
S	Signed numeric (positive/negative)
V	Implied decimal point (only in numbers-9

Type Character(s): A letter that defines what kind of data the field holds.

Size (inside parentheses): A number that tells how many characters or digits the field can store.

PROCEDURE DIVISION

```
PROCEDURE DIVISION.
  A000-FIRST-PARA.
  DISPLAY ‘Hello World!’.
  MOVE ‘Tutorials’ TO WS-NAME.
  DISPLAY “My name is: “WS-NAME
  DISPLAY “My ID is: “WS-ID.
STOP RUN.
```

Where the executable logic goes

Built with:
Paragraphs → Sentences → Statements

LOOPS

```
PERFORM <Paragraph-Name> UNTIL <Condition>.
```

```
WORKING-STORAGE SECTION.
01 COUNTER     PIC 9(2) VALUE 1.

PROCEDURE DIVISION.
    PERFORM DISPLAY-NUMBER UNTIL COUNTER > 5.
    STOP RUN.

DISPLAY-NUMBER.
    DISPLAY "Counter = " COUNTER.
    ADD 1 TO COUNTER.
```

Repetition using keywords like PERFORM, UNTIL, and VARYING.

COBOL doesn’t have traditional for or while loops like some other languages

ERROR HANDLING

```
IF WS-EMP-STATUS NOT = "00"
    DISPLAY "Error: " WS-EMP-STATUS.
```

No try-catch: COBOL uses status codes

Keywords: FILE STATUS, AT END, INVALID KEY

“00”	Successful operation
“10”	End of file
“23”	Record not found (indexed)
“35”	File not found
“39”	File mismatch (access mode)
“91”	File already open or locked

GNUCOBOL

- A free, open-source COBOL compiler
- Translates COBOL code into C, then compiles with GCC
- Great for learning, prototyping, and even real-world apps

```
IF I = 1
 DISPLAY “GnuCOBOL”
END-IF
```

GnuCOBOL

GnuCOBOL supports free-format syntax, so it’s more flexible with periods, indentation, and case sensitivity.

Still used in:

- 70% of financial transactions
- Government systems
- Legacy maintenance projects

High demand, low supply of COBOL developers

WHY LEARN COBOL TODAY?

Fun Fact: COBOL systems can outlive developers

- Think like a business analyst
- Practice PERFORM, MOVE, and IF
- Use meaningful names

FINAL TIPS & MOTIVATION

COBOL isn't dead — it's behind the scenes of modern life