Object-Oriented Programming
A Brief History of Programming Paradigms

Functional Programming (1930s - present)
- all about evaluating mathematical formulas using function calls
- no state changes or side effects
- Examples: lambda calculus, logic programming, Lisp, Scheme, Haskell

Imperative Programming (50s - 70s)
- focuses on global state changes
- a variable contains a value, an assignment statement changes it
- a print statement sends a value to output
- a "go to" transfers control to another statement
- "verbs" are the most important thing
- Functions / subprograms not really emphasized - used for reuse
- Examples: FORTRAN, COBOL, Algol, Basic
A Brief History of Programming Paradigms

Procedural Programming

• Similar to Imperative, but state changes are localized within subprograms
• State changes are communicated to other subprograms by parameters (arguments, return values)

Structured Programming (70s-80s)

• Focus on making programs easier to write, debug, and understand by use of subprograms, block structures, and for/while loops
• "go to" statements are blasphemous
• Ex: Pascal, C

Object-Oriented Programming (80s - present)

• Imperative in style, but features added to support Objects
• Ex: Smalltalk, Simula, C++, Python, Visual Basic, Java, Ruby
A Brief History of Programming Paradigms

In Procedural programs, you:
• break down a task into variables, data structures, and subprograms
• you use subprograms to operate on data structures

In Object-Oriented programs, you:
• define objects that expose behavior (methods) and data (attributes) using well-defined interfaces
• bundle everything together, so that an object only operates on its own attributes using methods
Four Basic Programming Concepts in OOP:

• **Encapsulation**: hiding implementation details of a class from other objects.

• **Abstraction**: simplifying complex reality by modeling classes appropriate to the problem.

• **Inheritance**: a way to define new classes using parts of classes that have already been defined.

• **Polymorphism**: the process of interpreting an operator or function in different ways for different data types.