Object-Oriented Programming
Imperative vs. Object-Oriented Programming

Imperative Programming (50s - 70s)

- uses statements to change a program's state
- 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

Object-Oriented Programming (80s - present)

- Imperative in style, structured, but features added to support "objects"
- Ex: Smalltalk, Simula, C++, Python, Visual Basic, Java, Ruby
In Imperative Programming, you:

- break down a task into variables, data structures, and subprograms
- you use subprograms to operate on data structures

In Object-Oriented Programming, 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.