

OBJECT-ORIENTED PROGRAMMING

Advanced Programming

Author: Eng. Carlos Andrés Sierra, M.Sc.
carlos.andres.sierra.v@gmail.com

Computer Engineer
Lecturer
Universidad Distrital Francisco José de Caldas

2024-I



Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



Outline

1 Object-Oriented Design

2 Good Practices in OO Design

3 Classes, Modules, and Packages

4 Resources Management



Basics of Object-Oriented Design I

JS → no methods

- **Object-oriented** has become one of the **most traditional and popular paradigms** in software development.
- It is based on the concept of objects, which can contain **C++**, **Python**, **JAVA** data, in the form of **fields** (often known as attributes or properties), and code, in the form of **procedures** (often known as methods).

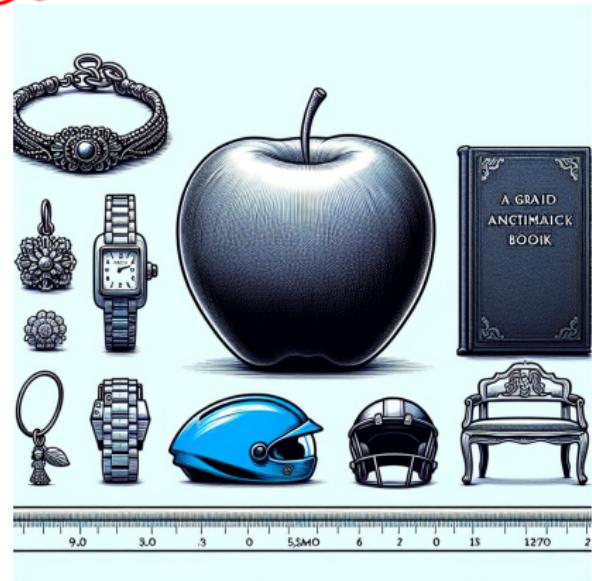


Figure: Prompt: Draw several objects sorted by size.



Basics of Object-Oriented Design I

- **Object-oriented** has become one of the **most traditional and popular paradigms** in software development.
- It is based on the concept of **objects**, which can contain data, in the form of **fields** (often known as **attributes** or **properties**), and code, in the form of **procedures** (often known as **methods**)

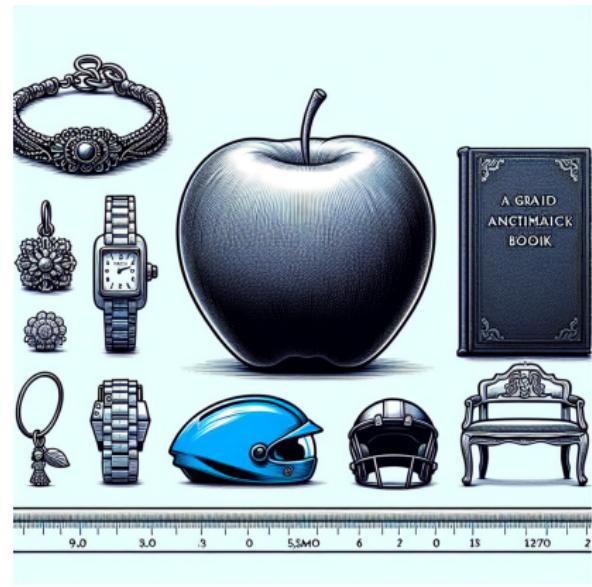


Figure: Prompt: Draw several objects sorted by size.



Basics of Object-Oriented Design II

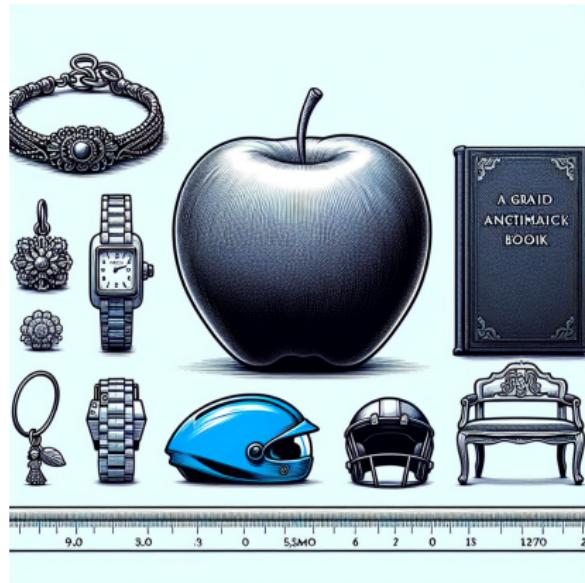


Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The main principles of OOD are:

Generalization



Basics of Object-Oriented Design II

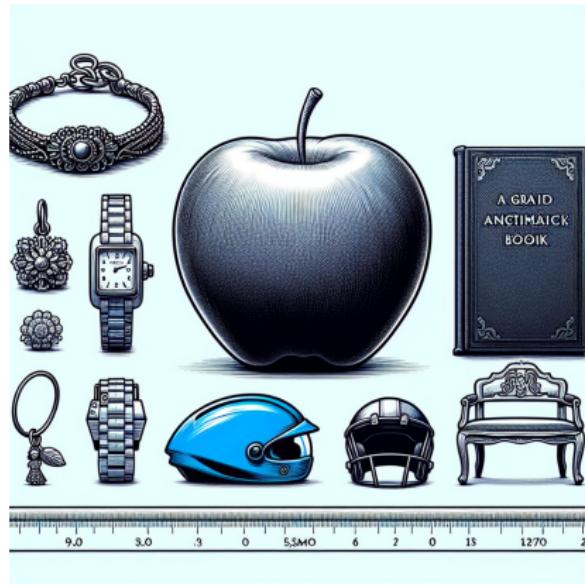


Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles of OOD** are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

Basics of Object-Oriented Design II

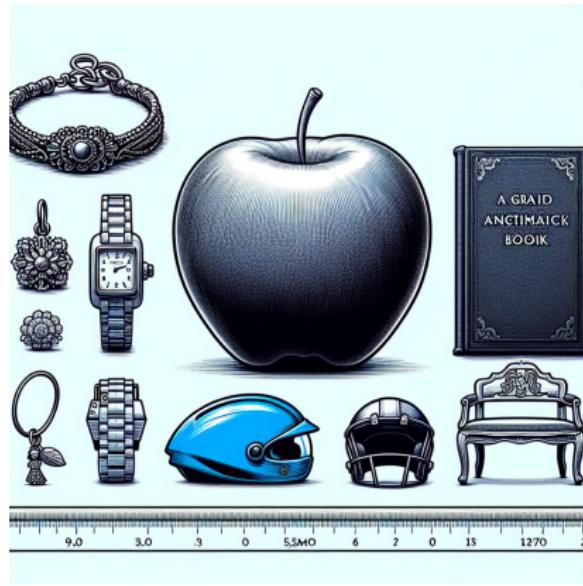


Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles** of OOD are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

Basics of Object-Oriented Design II

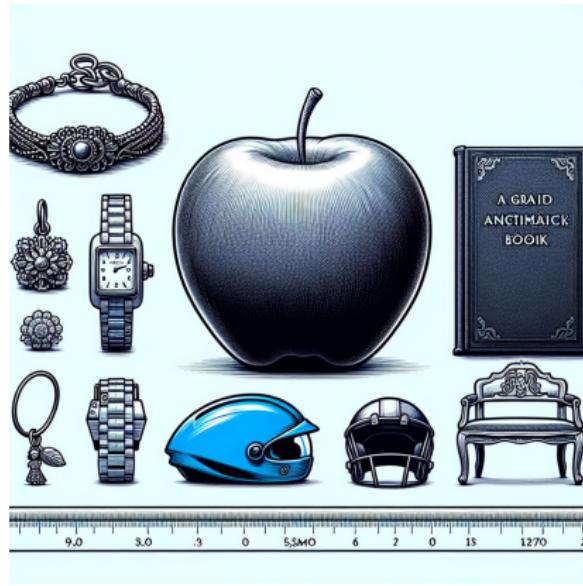


Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles** of OOD are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

Basics of Object-Oriented Design II

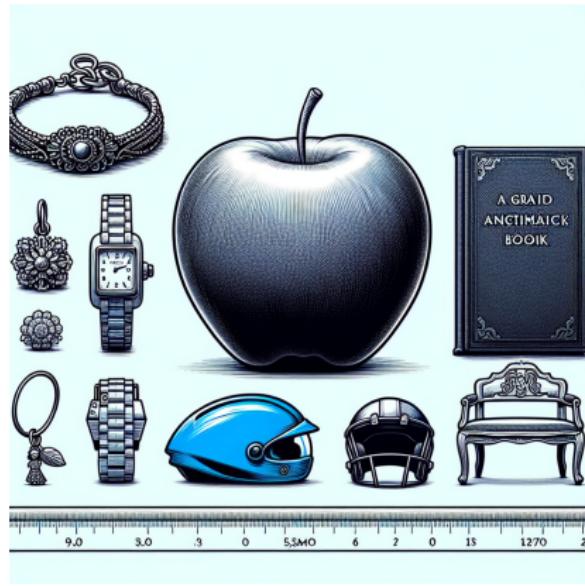


Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles** of OOD are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

Basics of Object-Oriented Design II

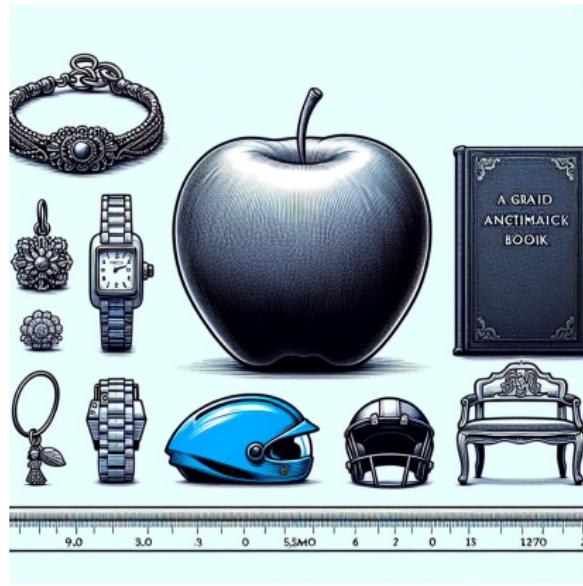


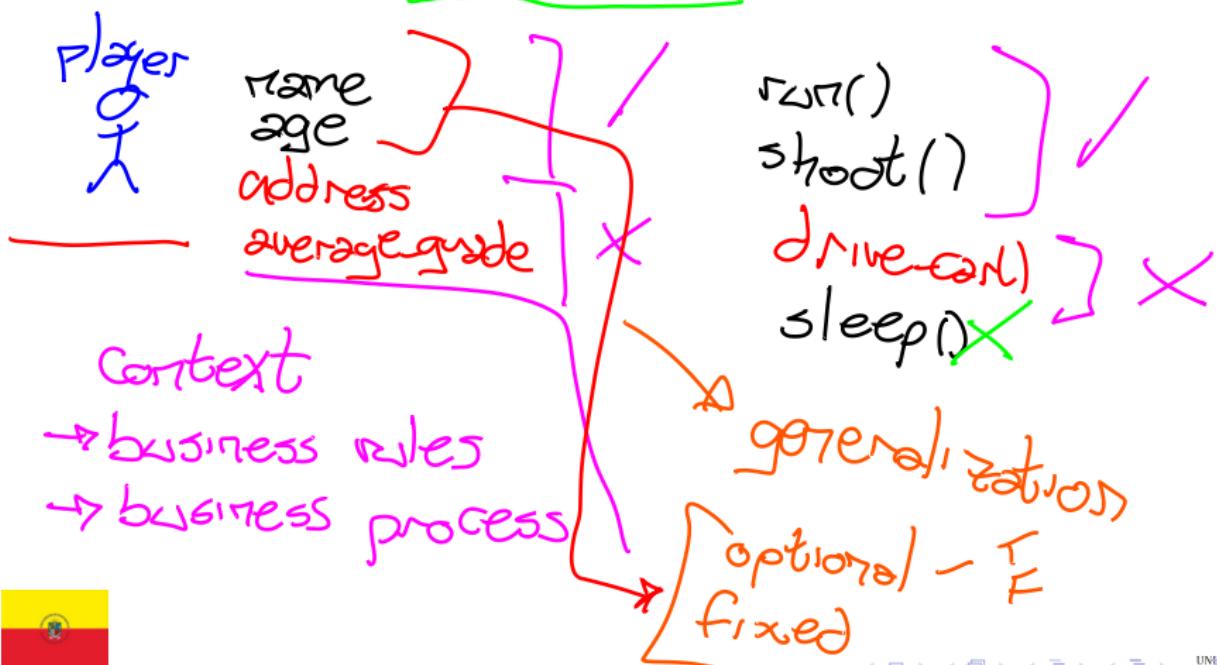
Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles** of OOD are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

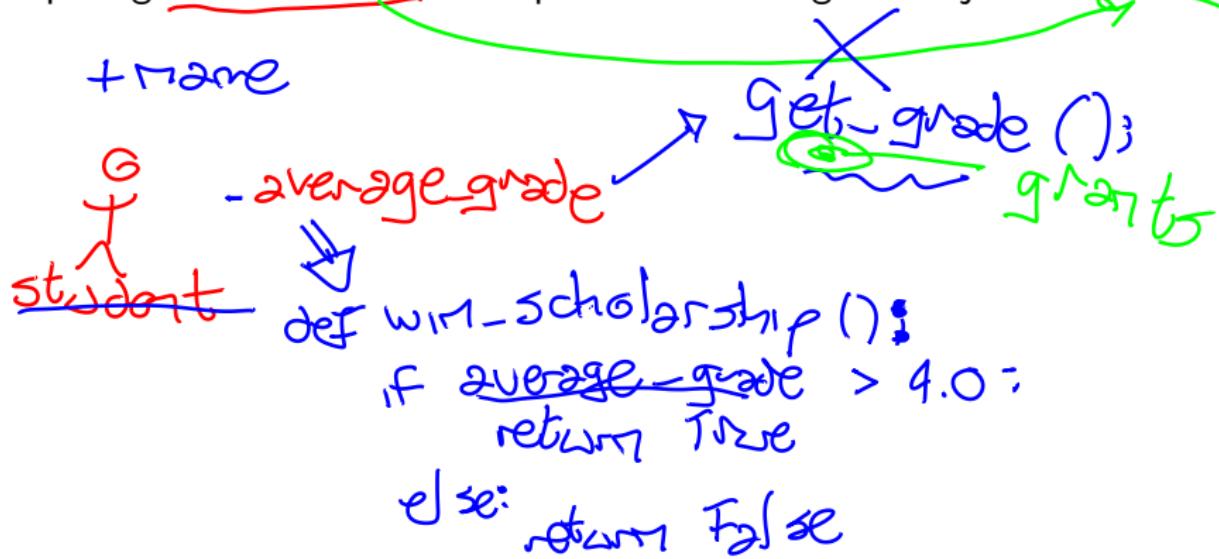
Abstraction in OOD

Abstraction is the process of **hiding** the **complex** implementation details and showing only the **necessary features** of an object.



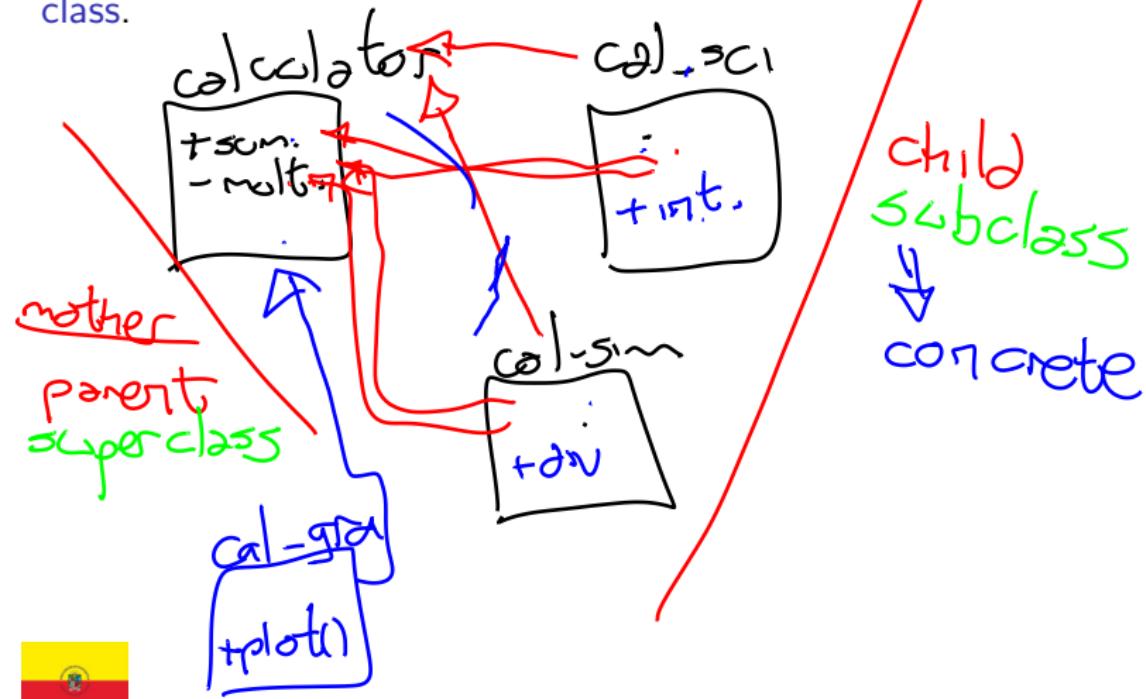
Encapsulation in OOD

Encapsulation is the process of **hiding** the internal state of an object and requiring all interactions to be performed through an object's **methods**.



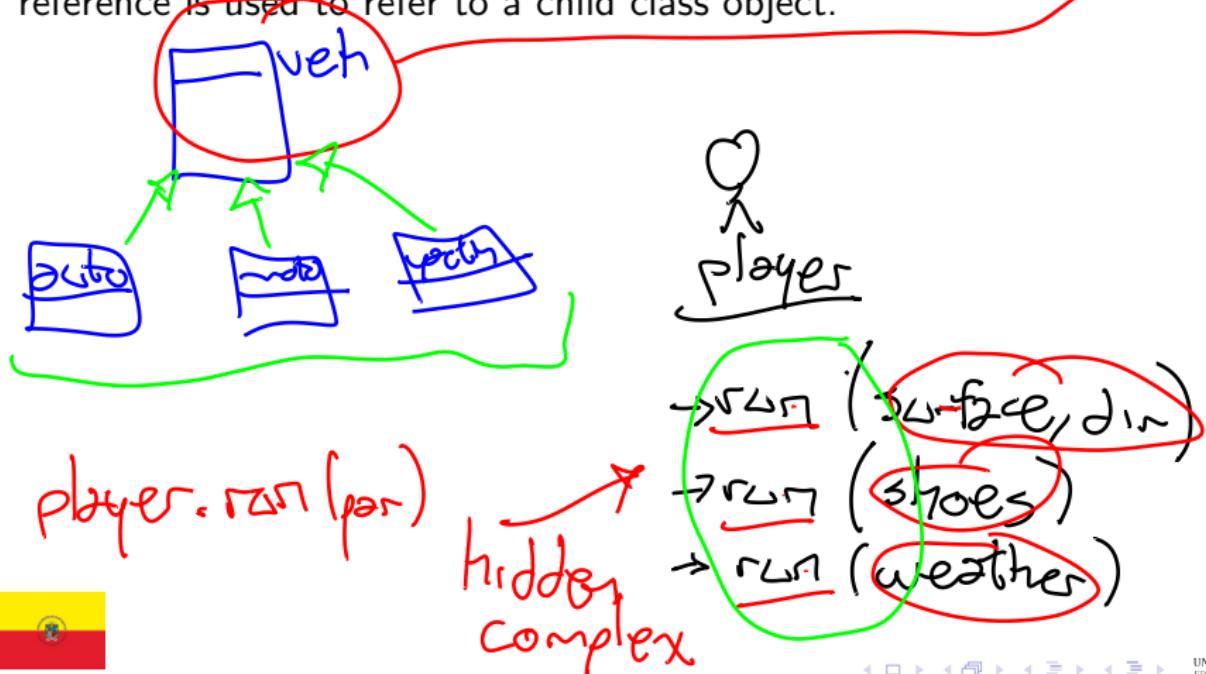
Inheritance in OOD

Inheritance is the process of creating a new class by **extending** an existing class.



Polymorphism in OOD

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.



Outline

1 Object-Oriented Design

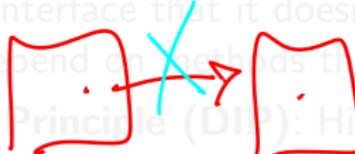
2 Good Practices in OO Design

3 Classes, Modules, and Packages

4 Resources Management



SOLID Principles

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
✓
- **Open/Closed Principle (OCP)**: A class should be open for extension, but closed for modification.
✓
- **Liskov Substitution Principle (LSP)**: Objects in a program should be replaceable with instances of their subtypes without affecting the correctness of that program.
✓
- **Interface Segregation Principle (ISP)**: A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.




SOLID Principles

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
- **Open/Closed Principle (OCP)**: A class should be open for extension; but closed for modification.
- **Liskov Substitution Principle (LSP)**: Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
+ inheritance
- **Interface Segregation Principle (ISP)**: A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.

- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.



SOLID Principles

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
 - **Open/Closed Principle (OCP)**: A class should be **open** for extension, but **closed** for modification.
 - **Liskov Substitution Principle (LSP)**: Objects in a program should be replaceable with instances of their **subtypes** without altering the correctness of that program.
 - **Interface Segregation Principle (ISP)**: A client should not be forced to implement an interface that it doesn't use or clients should not be forced to depend on methods they do not use.
 - **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstraction should not depend on details. Details should depend on abstractions.
-
- Diagram illustrating the Liskov Substitution Principle (LSP):
- listVehicle - O. Car**
 - 1. motor**
 - 2. car**
 - for V - listVehicle**
 - V. stop()**
- Annotations:
- Vehicle** (superclass)
 - Car**
 - Motor**
 - Bike**
 - stop()** (method)



SOLID Principles

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
- **Open/Closed Principle (OCP)**: A class should be open for extension, but closed for modification.
- **Liskov Substitution Principle (LSP)**: Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- **Interface Segregation Principle (ISP)**: A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.



SOLID Principles

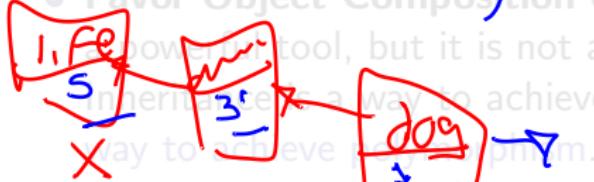
- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
- **Open/Closed Principle (OCP)**: A class should be open for extension, but closed for modification.
- **Liskov Substitution Principle (LSP)**: Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- **Interface Segregation Principle (ISP)**: A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.



Good Practices

- Composition over Inheritance: Inheritance should be used **only** when there is a clear relationship between the base class and the derived class. In other cases, **composition** should be used.

- Favor Object Composition over Class Inheritance: Inheritance is a powerful tool, but it is not always the best tool for the job.



- Code to interfaces, not Implementations: This principle is about designing your classes so that they depend on interfaces rather than concrete classes.

~~base - dog~~
+ yes: Tyro



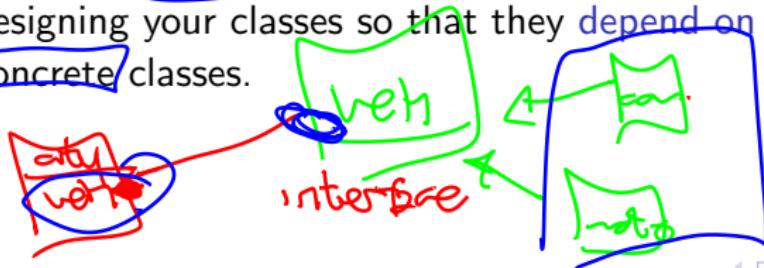
Good Practices

- **Composition over Inheritance:** Inheritance should be used **only when there is a clear relationship** between the base class and the derived class. In other cases, **composition** should be used.
- **Favor Object Composition over Class Inheritance.** Inheritance is a powerful tool, but it is **not always the best tool** for the job.
Inheritance is a way to achieve polymorphism, but it is **not the only way to achieve polymorphism.**
- **Code to Interfaces, not Implementations:** This principle is about designing your classes so that they depend on interfaces rather than concrete classes.



Good Practices

- **Composition over Inheritance:** Inheritance should be used **only when there is a clear relationship** between the base class and the derived class. In other cases, **composition** should be used.
- **Favor Object Composition over Class Inheritance:** Inheritance is a powerful tool, but it is not always the best tool for the job. Inheritance is a way to achieve polymorphism, but it is **not the only way to achieve polymorphism**.
- **Code to Interfaces, not Implementations:** This principle is about designing your classes so that they **depend on interfaces** rather than **concrete classes**.



Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



Modules in Python

Modules are [Python files](#) that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to [import](#) it using the `import` statement.
- **Creating Modules:** To create a module, you just have to save the code you want in a file with the file extension `.py`.
- **Built-in Modules:** Python has a set of built-in modules that you can use without installing them.



Modules in Python

Modules are [Python files](#) that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to [import](#) it using the `import` statement.
- **Creating Modules:** To create a module, you just have to save the code you want in a file with the [file extension](#) `.py`.
- **Built-in Modules:** Python has a set of built-in modules that you can use without installing them.



Modules in Python

Modules are [Python files](#) that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to [import](#) it using the `import` statement.
- **Creating Modules:** To create a module, you just have to save the code you want in a file with the [file extension](#) `.py`.
- **Built-in Modules:** Python has a set of [built-in modules](#) that you can use without installing them.



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a directory with an `__init__.py` file.
- **Importing Packages:** To import a package, you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating isolated environments for your **Python projects**.
- **Third-party Packages:** Python has a set of third-party packages that you can install using pip.



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a `directory` with an `__init__.py` file.
- **Importing Packages:** To `import a package`, you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating isolated environments for your `Python projects`.
- **Third-party Packages:** Python has a set of third-party packages that you can install using `pip`.



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a `directory` with an `__init__.py` file.
- **Importing Packages:** To `import a package`, you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating `isolated environments` for your **Python projects**.
- **Third-party Packages:** Python has a set of third-party packages that you can install using `pip`.



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a [directory](#) with an `__init__.py` file.
- **Importing Packages:** To [import a package](#), you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating [isolated environments](#) for your **Python projects**.
- **Third-party Packages:** Python has a set of [third-party packages](#) that you can install using pip.



Outline

1 Object-Oriented Design

2 Good Practices in OO Design

3 Classes, Modules, and Packages

4 Resources Management



Resources in Computation

binary

- **Memory Management**: It is the process of managing computer memory.

$$x = 5 \Rightarrow x \Rightarrow \textcircled{x} \infty$$

- **File Management:** It is the process of managing computer files.

Value design

- The diagram shows a 3x3 grid with the following values:

5	7	
0		
1		

To the right of the grid, there is a chemical equation: $O_2 + O \rightarrow O_2O$. A red arrow points from the '1' in the bottom-left cell of the grid to the oxygen atom in the reactant side of the equation.

$$\rightarrow \text{O}_2\text{O}_2$$

$$y = y + 2$$

referencia

$$\text{obj2} = \text{obj1}^{\text{assign}}$$

• clone
• copy



abstract
latent type

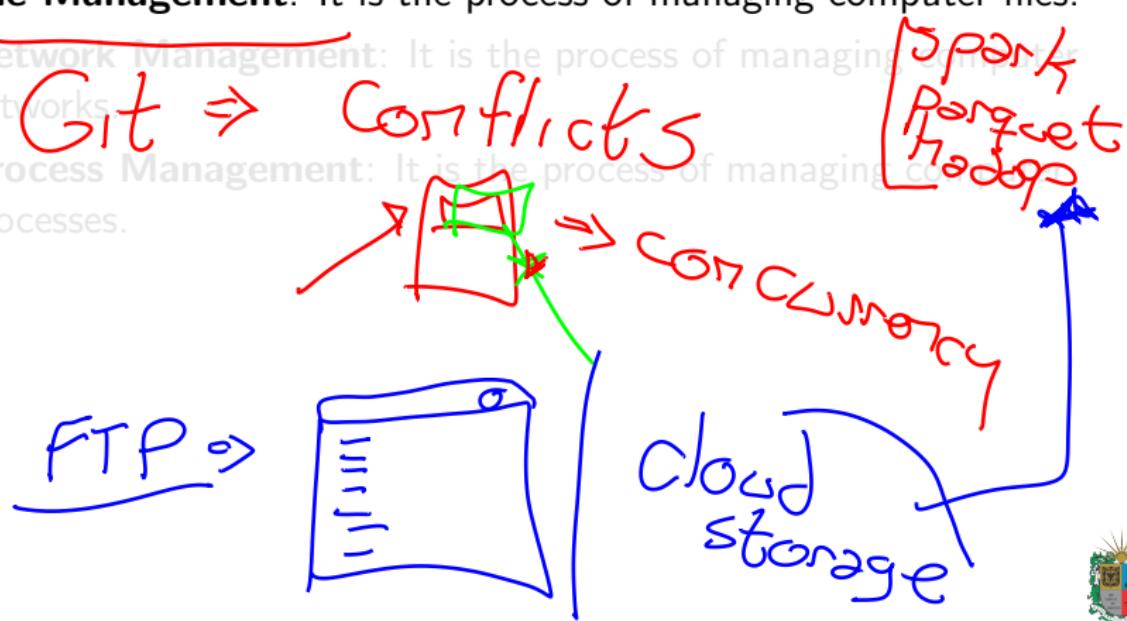
A hand-drawn diagram of a stack frame. A green bracket on the left is labeled '10' above it and '2' below it. Inside the frame, the word 'obj' is at the top, followed by 'size' and 'name' on the left, and 'l8' at the bottom right.

$$\text{O}_x \cap J \rightarrow \text{O}_x \cap Z \rightarrow \text{O}_x Z \rightsquigarrow \text{O}_x m$$



Resources in Computation

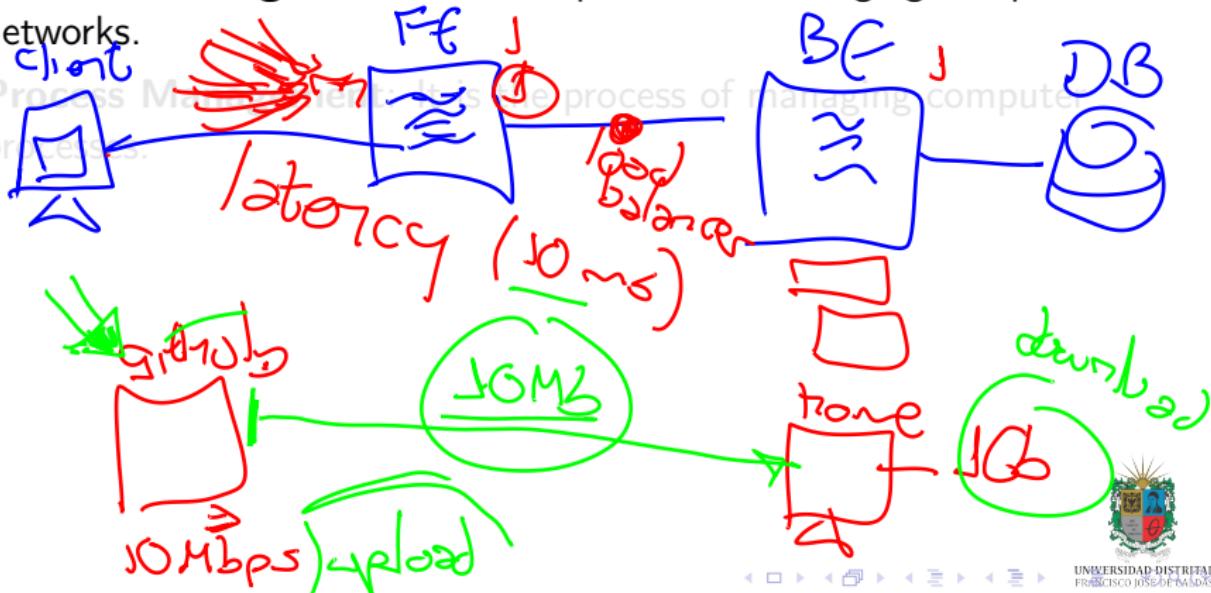
- **Memory Management:** It is the process of managing computer memory.
- **File Management:** It is the process of managing computer files.
- Network Management: It is the process of managing computer networks.
- **Process Management:** It is the process of managing computer processes.



Resources in Computation

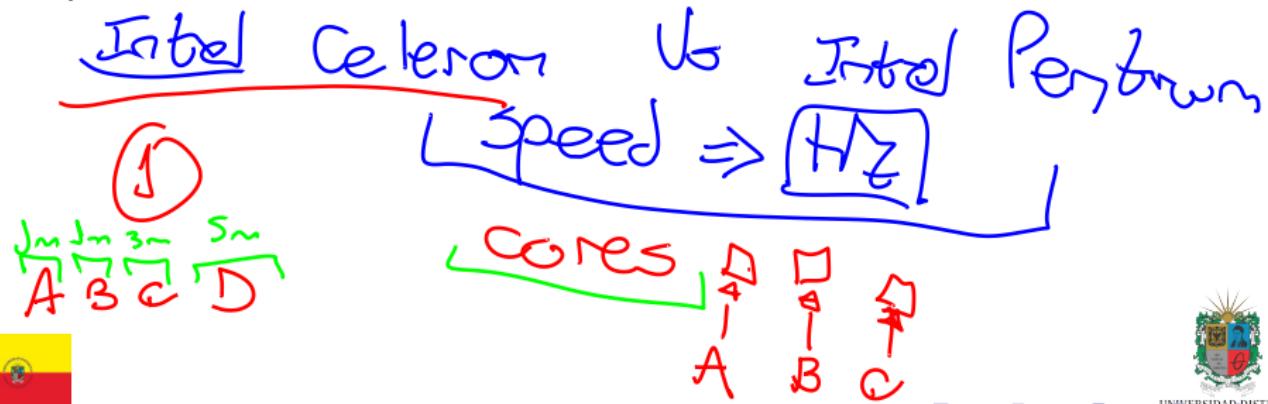
100000 \Rightarrow 100 Gbps

- **Memory Management:** It is the process of managing computer memory.
- **File Management:** It is the process of managing computer files.
- **Network Management:** It is the process of managing computer networks.
- **Process Management:** It is the process of managing computer processes.



Resources in Computation

- **Memory Management:** It is the process of managing computer memory.
- **File Management:** It is the process of managing computer files.
- **Network Management:** It is the process of managing computer networks.
- **Process Management:** It is the process of managing computer processes.



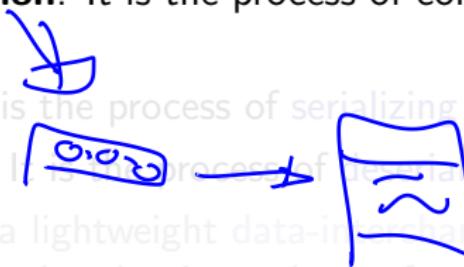
Objects Serialization

- **Serialization:** It is the process of converting an object into a stream of bytes.
 - Deserialization: It is the process of converting a stream of bytes into an object.
 - Pickling: It is the process of serializing an object in Python.
 - Unpickling: It is the process of deserializing an object in Python.
 - JSON: It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- Buffered Reader* *→ file*
Buffered Writer
- Pickle*



Objects Serialization

- **Serialization:** It is the process of converting an object into a **stream of bytes**.
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.
- **Pickling:** It is the process of serializing an object in Python.
- **Unpickling:** It is the process of deserializing an object in Python.
- **JSON:** It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.



Objects Serialization

- **Serialization:** It is the process of converting an object into a stream of bytes.
- **Deserialization:** It is the process of converting a stream of bytes into an object.
- **Pickling:** It is the process of serializing an object in Python.
- **Unpickling:** It is the process of deserializing an object in Python.
- **JSON:** It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

Java

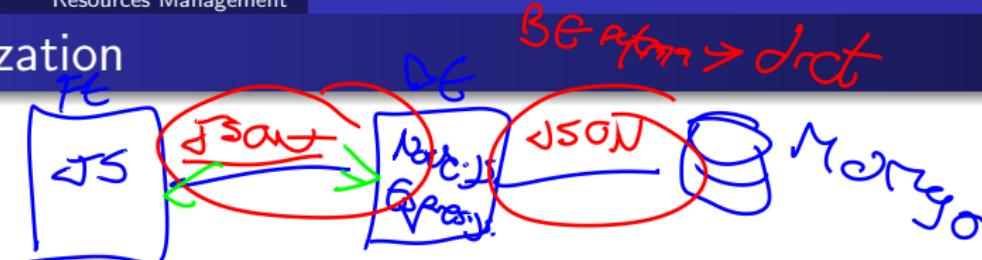


Objects Serialization

- **Serialization:** It is the process of converting an object into a **stream of bytes**.
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.
- **Pickling:** It is the process of **serializing** an object in Python.
- **Unpickling:** It is the process of **deserializing** an object in Python.
- **JSON:** It is a lightweight data-interchange format that is **easy for humans to read and write and easy for machines to parse and generate.**



Objects Serialization



- **Serialization:** It is the process of converting an object into a **stream of bytes**.
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.
- **Pickling:** It is the process of **serializing** an object in Python.
- **Unpickling:** It is the process of **deserializing** an object in Python.
- **JSON:** It is a **lightweight data-interchange format** that is easy for **humans** to read and write and easy for **machines** to parse and generate.

JavaScript Object Notation

Payload => Data

BSON - gRPC



Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is *no longer in use*.
 - Reference Counting: It is a technique used by Python to manage memory.
 - Memory Profiling: It is the process of analyzing the memory usage of a program.
 - Memory Leaks: They are a common problem in programming where memory is allocated but never deallocated.
 - Memory Management Tools: There are several tools available for memory management in Python.
- fast used 30 minutes*
- 



Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming memory* that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage memory**.
- **Memory Profiling:** It is the process of analyzing the memory usage of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but never deallocated.
- **Memory Management Tools:** There are several tools available for memory management in Python.



Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage memory**.
- **Memory Profiling:** It is the process of **analyzing** the **memory usage** of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but never deallocated.
- **Memory Management Tools:** There are several tools available for memory management in Python.



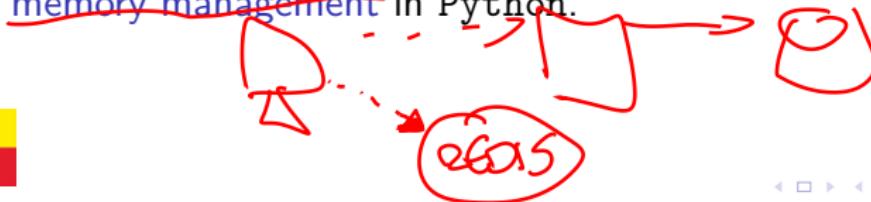
Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage memory**.
- **Memory Profiling:** It is the process of analyzing the **memory usage** of a program.
- **Memory Leaks:** They are a **common problem** in programming where **memory is allocated but never deallocated**.
- **Memory Management Tools:** There are several tools available for **memory management** in Python.



Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is *no longer in use*.
- **Reference Counting:** It is a technique used by Python to *manage memory*.
- **Memory Profiling:** It is the process of analyzing the *memory usage* of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but *never deallocated*.
- **Memory Management Tools:** There are several tools available for *memory management* in Python.



Threads, Processes, and Concurrency

- **Threads:** They are the smallest unit of execution that can be scheduled by an operating system.
- **Processes:** They are the largest unit of execution that can be scheduled by an operating system.
↳ Firefox → **Process - Application**
↳ Java
↳ kernel
↳ thread
- **Parallelism:** It is the ability of a program to execute multiple tasks simultaneously.
- **Synchronization:** Synchronization is the process of coordinating the execution of multiple threads or processes.



Threads, Processes, and Concurrency

- **Threads:** They are the **smallest unit of execution** that can be scheduled by an operating system.
 - **Processes:** They are the **largest unit of execution** that can be scheduled by an operating system.
 - **Parallelism:** It is the ability of a program to run multiple tasks simultaneously.
 - **Synchronization:** Synchronization is the process of coordinating the execution of multiple threads or processes.
- (Handwritten notes: A blue bracket groups "Threads" and "Processes". A red bracket groups "Parallelism" and "Synchronization". A red bracket groups "Threads" and "Synchronization".)*



Threads, Processes, and Concurrency

- **Threads:** They are the **smallest unit of execution** that can be scheduled by an operating system.
- **Processes:** They are the **largest unit of execution** that can be scheduled by an operating system.
- **Parallelism:** It is the ability of a program to execute multiple tasks simultaneously.
- **Synchronization:** Synchronization is the process of coordinating the execution of multiple threads or processes.

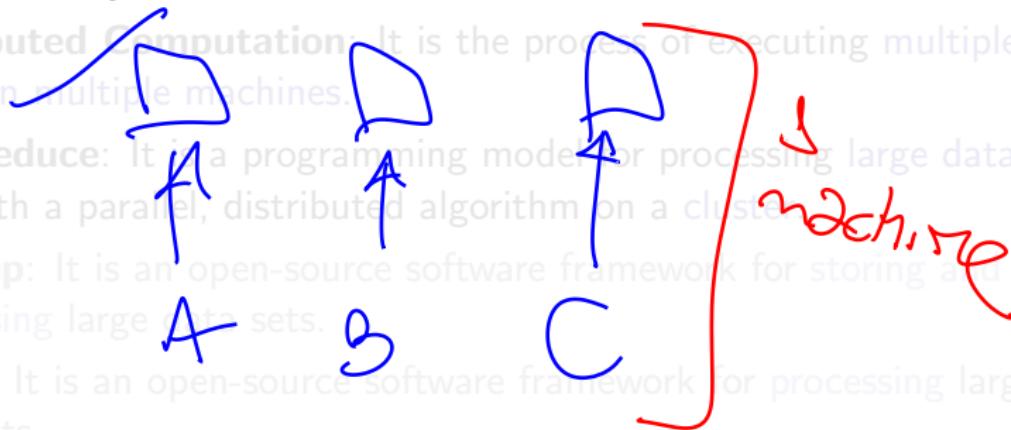


Threads, Processes, and Concurrency

- **Threads:** They are the **smallest unit of execution** that can be scheduled by an operating system.
- **Processes:** They are the **largest unit of execution** that can be scheduled by an operating system.
- **Parallelism:** It is the ability of a program to execute **multiple tasks simultaneously**.
- **Synchronization:** Synchronization is the process of coordinating the execution of **multiple threads** or processes.



Parallel Computation

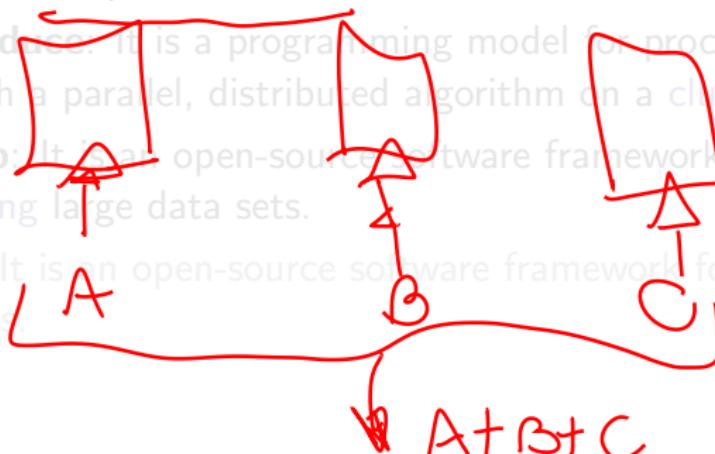
- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing multiple tasks on **multiple machines**.A hand-drawn diagram illustrates the concept of distributed computation. Three blue arrows labeled A, B, and C point upwards from the bottom towards three separate blue shapes at the top, representing individual machines. A red bracket on the right side groups all three machines together, with the word "machine" written in red next to it.
- **MapReduce:** It is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
- **Hadoop:** It is an open-source software framework for storing and processing large data sets.
- **Spark:** It is an open-source software framework for processing large data sets.



Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.

- MapReduce: It is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
- Hadoop: It is an open-source software framework for storing and processing large data sets.
- Spark: It is an open-source software framework for processing large data sets.



Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a **parallel, distributed algorithm** on a **cluster**.



Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a parallel, distributed algorithm on a **cluster**.
- **Hadoop:** It is an **open-source** software framework for **storing and processing** large data sets.
- **Spark:** It is an open-source software framework for processing large data sets.



Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a parallel, distributed algorithm on a **cluster**.
- **Hadoop:** It is an open-source software framework for **storing and processing large data sets**.
- **Spark:** It is an open-source software framework for **processing large data sets**.

Pyspark

↳ Apache

Apache Servers (XAMP)
NetBeans
Redis
Hadoop
Kafka

MGBabit
Parquet
Iceberg



Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



Thanks!

Questions?



Repo:

[github.com/engandres/ud-public/tree/main/courses/
advanced-programming](https://github.com/engandres/ud-public/tree/main/courses/advanced-programming)

