let courses =[

{

"id": 1,

"module": "Introduction to Python",

"title": "What is Python?",

"content": `Python is a high-level, interpreted programming language known for its simplicity and versatility.

It was created by Guido van Rossum and first released in 1991.

Python’s design philosophy emphasizes code readability.

It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python’s syntax is clean and easy to understand.

Its dynamic typing and automatic memory management make it a popular choice for developers.

The Python Package Index (PyPI) hosts thousands of third-party libraries.

Python is widely used in web development, data analysis, artificial intelligence, and automation.

Many major companies, including Google and Netflix, use Python in production environments.

Python’s active community and extensive documentation make it easy to learn and use.`

},

{

"id": 2,

"module": "Introduction to Python",

"title": "Installing Python",

"content": `Installing Python is simple on most systems.

Visit the official Python website (https://www.python.org/) to download the latest version.

On Windows, run the installer and ensure to check the box to add Python to the system PATH.

On macOS, Python can be installed via Homebrew with the command "brew install python3".

Linux users can install Python using their package manager, such as "sudo apt-get install python3".

After installation, you can verify the installation by running "python --version" in the command line.

The Python interpreter can be accessed through the command line by typing "python3".

It is also recommended to install a virtual environment to manage dependencies for different projects.

Virtual environments can be created using the "venv" module or by installing third-party tools like "virtualenv".

IDEs like VSCode or PyCharm can help in writing Python code efficiently.`

},

{

"id": 3,

"module": "Introduction to Python",

"title": "Your First Python Program",

"content": `To write your first Python program, open a text editor and save the file with a ".py" extension.

Start by writing the following code: "print('Hello, World!')".

Save the file as "hello.py" and run it using the Python interpreter by typing "python hello.py".

The output should be "Hello, World!" displayed in the terminal.

The "print" function sends text to the console.

This is the simplest Python program and is often used to introduce new learners to programming.

Understanding how to run Python programs is the first step in becoming proficient in the language.

As you continue learning, you will add more complex functionality to your Python programs.

From here, you can experiment with variables, input, and more complex functions.

Writing and running Python code can be an exciting process as you see your ideas come to life.`

},

{

"id": 4,

"module": "Core Python Concepts",

"title": "Variables and Data Types",

"content": `In Python, variables are used to store data that can be used later in the program.

Variables are dynamically typed, meaning you don’t have to declare their type explicitly.

Common data types in Python include integers, floats, strings, and booleans.

A string is a sequence of characters enclosed in single, double, or triple quotes.

Integers represent whole numbers, while floats represent decimal numbers.

Booleans can be either "True" or "False" and are used in logical operations.

You can assign values to variables using the "=" sign.

Python also supports compound data types like lists, tuples, dictionaries, and sets.

Lists are mutable, meaning you can change their values, while tuples are immutable.

Understanding data types is crucial for performing operations and writing efficient code in Python.`

},

{

"id": 5,

"module": "Core Python Concepts",

"title": "Control Flow",

"content": `Control flow statements allow you to control the execution order of your code.

The "if" statement executes a block of code if a condition is true.

The "elif" statement provides an alternative condition, and "else" handles the case where all conditions are false.

For example, "if x > 10" will execute the code if x is greater than 10, and "else" can handle the case if it is not.

Python also includes "while" loops for repeating a block of code as long as a condition is true.

"For" loops iterate over a sequence, such as a list or a range of numbers.

You can control the flow of a loop with "break" (to exit the loop) and "continue" (to skip to the next iteration).

Logical operators like "and", "or", and "not" can combine multiple conditions in control flow statements.

Control flow is essential for making decisions and managing the flow of execution in Python programs.

Learning how to use control flow allows you to create dynamic and flexible programs.`

},

{

"id": 6,

"module": "Core Python Concepts",

"title": "Loops",

"content": `Loops are used to repeat a block of code multiple times.

"For" loops are commonly used when you know the number of iterations in advance.

A "for" loop in Python iterates over a sequence, such as a list, tuple, or range of numbers.

Example: "for i in range(5)" will iterate through the numbers 0 to 4.

"While" loops are used when you want to repeat a block of code while a condition is true.

Example: "while x < 10" will continue looping as long as x is less than 10.

Loop control statements, like "break" and "continue", allow you to modify the flow of the loop.

"Break" exits the loop immediately, while "continue" skips to the next iteration of the loop.

Nested loops can also be used, where a loop is inside another loop, but be mindful of performance.

Mastering loops is important for performing repetitive tasks efficiently in Python.`

},

{

"id": 7,

"module": "Core Python Concepts",

"title": "Functions",

"content": `Functions allow you to group related code together, making it reusable and easier to maintain.

In Python, you define functions using the "def" keyword, followed by the function name and parentheses.

Example: "def greet(name):" defines a function that takes a parameter "name".

Functions can return values using the "return" keyword.

You can pass arguments to a function, allowing it to perform tasks with different data each time.

Python supports default arguments, so you can provide values if the caller doesn’t specify them.

Functions can also have variable-length arguments using the *args and **kwargs syntax.

Functions help to organize your code and avoid repetition.

A well-designed function can improve code readability and reduce the risk of errors.

Learning how to write functions is a key step toward becoming a proficient Python programmer.`

},

{

"id": 8,

"module": "Intermediate Python",

"title": "Object-Oriented Programming (OOP)",

"content": `Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects.

In OOP, classes define the structure and behavior of objects.

A class is like a blueprint, while an object is an instance of that class.

Classes can contain attributes (data) and methods (functions that define behavior).

Inheritance allows a class to inherit attributes and methods from another class.

Polymorphism allows objects of different classes to be treated as instances of the same class.

Encapsulation is the practice of hiding internal details and exposing only necessary parts of the object.

OOP makes it easier to organize and manage complex codebases.

It encourages code reuse, making programs easier to maintain and scale.

Mastering OOP principles is essential for building large, modular applications in Python.`

},

{

"id": 9,

"module": "Intermediate Python",

"title": "Modules and Packages",

"content": `Modules are files that contain Python code, which can be imported and reused in other programs.

A module can contain functions, classes, and variables, as well as runnable code.

To import a module, use the "import" statement, followed by the module name.

Python has many built-in modules, such as "math" and "os", that provide useful functionality.

You can also create your own modules by saving your code in a file with a ".py" extension.

A package is a collection of related modules organized in a directory hierarchy.

Packages allow you to structure your code and avoid naming conflicts.

Python’s standard library contains a wealth of modules and packages for a wide range of tasks.

You can install third-party packages using "pip", Python’s package manager.

Modules and packages are vital for organizing and reusing code efficiently.`

},

{

"id": 10,

"module": "Intermediate Python",

"title": "File Handling",

"content": `File handling is a fundamental skill in Python that enables reading from and writing to files.

To open a file, use the built-in "open()" function with the mode specified as 'r', 'w', or 'a'.

The 'r' mode opens a file for reading, 'w' for writing (overwriting the file), and 'a' for appending to the file.

You can use the "read()" or "readlines()" method to read content from a file.

Writing is done using the "write()" or "writelines()" method to add text to the file.

Always close the file after operations using "close()" or a "with" statement for automatic closure.

The "with" statement is preferred as it ensures the file is properly closed even if an error occurs.

You can work with binary files by specifying 'rb' or 'wb' modes.

Python also provides modules like "csv" and "json" for handling structured file formats.

File handling enables developers to create, process, and store data efficiently.

Mastering file operations is essential for working with data-driven applications.`

},

{

"id": 11,

"module": "Advanced Python",

"title": "Error and Exception Handling",

"content": `Error handling in Python ensures your program can handle unexpected situations gracefully.

Use "try" and "except" blocks to catch exceptions and prevent crashes.

Example: "try: x = 10 / 0 except ZeroDivisionError: print('Cannot divide by zero')".

The "finally" block is executed regardless of whether an exception occurred, often used for cleanup actions.

"Raise" is used to manually trigger an exception with a custom message.

Python provides a wide range of built-in exceptions like "ValueError", "TypeError", and "FileNotFoundError".

You can define custom exceptions by creating classes that inherit from "Exception".

Proper exception handling improves user experience and program reliability.

Logging errors with the "logging" module helps in debugging and maintaining applications.

Writing robust code involves anticipating potential issues and preparing appropriate responses.`

},

{

"id": 12,

"module": "Advanced Python",

"title": "Data Structures",

"content": "Python provides versatile data structures to store and organize data. Lists are mutable sequences used for storing collections of items. Tuples are immutable and are useful for fixed data. Sets store unique items and support mathematical operations like union and intersection. Dictionaries store key-value pairs, making data retrieval efficient. Python also includes collections like defaultdict, OrderedDict, and namedtuple for specialized needs. Understanding data structures helps optimize algorithms and memory usage. Lists and dictionaries are the most frequently used in Python programs. Data structures are essential for managing complexity in software development. Mastering them is key to writing efficient and effective Python code."

},

```
  {
    "id": 13,
    "module": "Advanced Python",
    "title": "Working with APIs",
    "content": "APIs (Application Programming Interfaces) allow communication between software applications. Python's 'requests' library is widely used for making HTTP requests to APIs. Use 'requests.get()' to fetch data from an API and 'requests.post()' to send data. JSON is the most common format for API data, and Python provides a built-in 'json' module for parsing. APIs often require authentication using keys or tokens for secure access. Understanding API documentation is crucial for using them effectively. Common use cases include fetching data from weather, finance, or social media APIs. You can also create your own APIs using frameworks like Flask or Django. APIs are vital for integrating different systems and automating tasks. Practicing with APIs enhances your ability to work on modern web and software projects."
  },
  {
    "id": 14,
    "module": "Advanced Python",
    "title": "Web Development with Flask",
    "content": "Flask is a lightweight Python framework for web development. It follows the WSGI standard and allows building scalable web applications. Start by installing Flask using 'pip install flask'. Create a basic app with a few lines of code using the 'Flask' class. Flask uses routes to map URLs to functions, defining the app's structure. Templates in Flask allow dynamic HTML generation with Jinja2. The framework supports session management and can connect to databases. Extensions like Flask-SQLAlchemy provide advanced functionality. Flask is highly customizable and suitable for small to medium-sized projects. Learning Flask helps in understanding backend development and building REST APIs."
  },
  {
    "id": 15,
    "module": "Projects and Practice",
    "title": "Mini Project: Calculator",
```

"content": "A calculator is a great project to practice Python basics. Start by defining functions for arithmetic operations like addition, subtraction, multiplication, and division. Use a menu-driven approach to let users choose an operation. Handle user input using the 'input()' function. Validate inputs to ensure correct data types and handle exceptions for invalid inputs. Implement a loop to allow users to perform multiple calculations. Enhance the program by adding advanced operations like square roots or exponentiation. You can also create a command-line interface for better user interaction. This project reinforces concepts like functions, loops, and exception handling. Completing the calculator builds confidence in your Python skills."

},

{

"id": 16,

"module": "Projects and Practice",

"title": "Mini Project: To-Do List App",

"content": "A To-Do List app helps practice Python with file handling and basic algorithms. Start by creating a menu-driven interface with options to add, view, edit, and delete tasks. Use a list or dictionary to store tasks in memory during runtime. Save tasks to a file so that the data persists between sessions. Use 'open()', 'readlines()', and 'writelines()' for file handling. Implement input validation to ensure meaningful task entries. Enhance functionality with task priorities or due dates. Add a search feature to find tasks quickly. Test the program for edge cases like empty task lists. This project teaches modular programming and enhances problem-solving skills. Building this app provides a solid foundation for creating practical tools."

},

{

"id": 17,

"module": "Projects and Practice",

"title": "Capstone Project: Build a REST API",

"content": "Building a REST API consolidates Python and Flask knowledge. Start by setting up a Flask project and installing required libraries. Define endpoints for common operations like GET, POST, PUT, and DELETE. Use Flask's routing and request handling features to implement API logic. Manage data using Python dictionaries or integrate a database like SQLite or MongoDB. Use JSON format for data exchange between the client and server. Add authentication to secure the API and manage user access. Write proper error handling to return meaningful HTTP status codes. Test the API using tools like Postman or curl. This project

provides hands-on experience in backend development and API design. Completing the capstone demonstrates your ability to create scalable and maintainable applications."

},

{

"id": 18,

"module": "Course Completed",

"title": "Course Completed",

"content": "Congradulation you have completed the course successfully."

}

];

let javaCourses = [

{

"id": 1,

"module": "Introduction to Java",

"title": "What is Java?",

"content": `Java is a high-level, object-oriented programming language developed by Sun Microsystems in 1995.

It is platform-independent, meaning "write once, run anywhere" (WORA).

Java's syntax is similar to C++ but simpler, making it easier to learn.

It is widely used in web, mobile, and enterprise application development.

Java programs run on the Java Virtual Machine (JVM), ensuring cross-platform compatibility.

Popular for its robustness, Java includes strong memory management features.

Java supports multithreading, allowing simultaneous execution of multiple tasks.

It is backed by a large community and extensive libraries.

Companies like Amazon, Google, and Netflix rely on Java for scalability.

Learning Java opens doors to diverse software development opportunities.`

},

{

"id": 2,

"module": "Introduction to Java",

"title": "Installing Java",

"content": `To install Java, download the latest JDK (Java Development Kit) from Oracle's official website.

Follow the installation wizard for your OS and set up the JAVA_HOME environment variable.

Verify the installation by running "java -version" in your terminal.

An IDE like IntelliJ IDEA, Eclipse, or NetBeans can streamline Java development.

Ensure you also install the Java Runtime Environment (JRE) for running Java applications.

The JDK includes essential tools like javac (compiler) and java (runtime).

Platform-specific installation steps are well-documented in Java's official resources.

Keeping your Java version up-to-date ensures access to the latest features.

Learning the setup process is a foundational step in mastering Java.

Once installed, you are ready to create and execute your first Java program.`

},

{

"id": 3,

"module": "Core Java Concepts",

"title": "Variables and Data Types",

"content": `Java variables store data and must be declared with a specific type.

Common data types include int, double, char, and boolean.

Strings are handled using the String class and are immutable.

Java supports type casting for converting between compatible data types.

Local, instance, and static variables are the main variable types in Java.

Constants are defined using the "final" keyword.

Arrays allow storing multiple values of the same type in a single variable.

Java follows strict type checking to ensure robust code execution.

Understanding data types is key to efficient Java programming.

Practice declaring and using variables in simple Java programs.`

},

{

"id": 4,

"module": "Core Java Concepts",

"title": "Control Flow",

"content": `Java uses control flow statements to direct program execution.

"if-else" statements evaluate conditions to execute specific code blocks.

"switch" statements handle multiple conditions more efficiently.

Loops like "for", "while", and "do-while" enable repetitive task execution.

The "break" and "continue" keywords control loop execution flow.

Java supports logical operators like &&, ||, and ! for complex conditions.

Nested loops and conditional statements add flexibility to code.

Control flow ensures logical sequencing and decision-making in programs.

Efficient use of control flow minimizes errors and improves program clarity.

Practice building programs that incorporate these control statements.`

},

{

"id": 5,

"module": "Core Java Concepts",

"title": "Object-Oriented Programming (OOP)",

"content": `Java is inherently object-oriented, focusing on classes and objects.

A class is a blueprint for creating objects, containing attributes and methods.

Objects are instances of classes, encapsulating state and behavior.

Key OOP principles in Java include inheritance, polymorphism, and encapsulation.

Abstraction allows hiding implementation details from the user.

Interfaces and abstract classes enable flexible designs.

Java promotes code reusability and modular development.

Mastering OOP concepts is essential for effective Java programming.

Real-world examples demonstrate how Java simplifies complex systems.

Building small OOP-based programs helps solidify these concepts.`

},

{

"id": 6,

"module": "Intermediate Java",

"title": "Exception Handling",

"content": `Java uses exceptions to handle runtime errors gracefully.

Use "try" and "catch" blocks to catch and handle exceptions.

The "finally" block ensures cleanup actions are executed.

Common exceptions include NullPointerException and ArithmeticException.

Use "throw" to explicitly raise an exception in your program.

Custom exceptions can be created by extending the Exception class.

The "throws" keyword declares exceptions a method might throw.

Proper exception handling improves program reliability.

Java’s robust error-handling features are essential for real-world applications.

Practice writing programs that handle various exception scenarios.`

},

{

"id": 7,

"module": "Intermediate Java",

"title": "Collections Framework",

"content": `Java's Collections Framework provides powerful data structures.

Common interfaces include List, Set, and Map.

ArrayList and LinkedList are dynamic alternatives to arrays.

HashSet and TreeSet store unique elements with different orderings.

HashMap and TreeMap manage key-value pairs efficiently.

Collections offer methods for searching, sorting, and filtering data.

Generics enhance type safety within collections.

Mastering collections is vital for managing data in Java applications.

Real-world projects rely heavily on Java’s robust collections.

Practice using collections to solve various data-handling problems.`

},

{

"id": 8,

"module": "Intermediate Java",

"title": "Threads and Multithreading",

"content": `Multithreading allows concurrent execution of code in Java.

Threads are created using the Thread class or Runnable interface.

The "synchronized" keyword prevents thread interference and data inconsistency.

Java provides the Executor framework for managing thread pools.

Use "wait" and "notify" methods for inter-thread communication.

Multithreading improves application performance and responsiveness.

Proper synchronization ensures thread safety in shared resources.

Debugging multithreaded programs requires careful analysis.

Real-time applications like games and servers use multithreading extensively.

Start with simple multithreading examples to grasp its concepts.`

},

{

"id": 9,

"module": "Advanced Java",

"title": "Working with Files",

"content": `File handling in Java uses classes from the java.io and java.nio packages.

Use FileReader and FileWriter for reading and writing text files.

BufferedReader and BufferedWriter enhance efficiency with larger files.

ObjectOutputStream and ObjectInputStream handle object serialization.

Java NIO offers advanced features like non-blocking I/O and file channels.

Exception handling is crucial to manage file operation errors.

Practice creating, reading, and writing files in different formats.

Real-world applications often involve processing file data.

Java’s extensive file handling APIs make it suitable for large-scale data management.

Understanding file I/O is essential for building robust applications.`

},

{

"id": 10,

```
    "module": "Projects and Practice",
    "title": "Mini Project: Student Management System",
    "content": `Build a simple console-based student management system in Java.
    Implement features like adding, updating, and deleting student records.
    Use ArrayList or HashMap to store student data dynamically.
    Practice reading and writing student data to a file for persistence.
    Apply exception handling to manage invalid inputs gracefully.
    Design the program using OOP principles for modularity.
    Add a search feature to retrieve student information by ID or name.
    Enhance the program with sorting options for better usability.
    Testing and debugging ensure the program works flawlessly.
    Completing this project boosts confidence in Java programming.`
  },
  {
    "id": 11,
    "module": "Course Completed",
    "title": "Congradulations ",
    "content": `Congradulations You have completed the course`
  }
];

let adjavaCourses = [
    {
      "id": 1,
      "module": "Advanced java : Introduction to Java",
```

"title": "What is Java?",

"content": `Java is a high-level, object-oriented programming language developed by Sun Microsystems in 1995.

It is platform-independent, meaning "write once, run anywhere" (WORA).

Java's syntax is similar to C++ but simpler, making it easier to learn.

It is widely used in web, mobile, and enterprise application development.

Java programs run on the Java Virtual Machine (JVM), ensuring cross-platform compatibility.

Popular for its robustness, Java includes strong memory management features.

Java supports multithreading, allowing simultaneous execution of multiple tasks.

It is backed by a large community and extensive libraries.

Companies like Amazon, Google, and Netflix rely on Java for scalability.

Learning Java opens doors to diverse software development opportunities.`

},

{

"id": 2,

"module": "Introduction to Java",

"title": "Installing Java",

"content": `To install Java, download the latest JDK (Java Development Kit) from Oracle's official website.

Follow the installation wizard for your OS and set up the JAVA_HOME environment variable.

Verify the installation by running "java -version" in your terminal.

An IDE like IntelliJ IDEA, Eclipse, or NetBeans can streamline Java development.

Ensure you also install the Java Runtime Environment (JRE) for running Java applications.

The JDK includes essential tools like javac (compiler) and java (runtime).

Platform-specific installation steps are well-documented in Java's official resources.

Keeping your Java version up-to-date ensures access to the latest features.

Learning the setup process is a foundational step in mastering Java.

Once installed, you are ready to create and execute your first Java program.`

},

{

"id": 3,

"module": "Core Java Concepts",

"title": "Variables and Data Types",

"content": `Java variables store data and must be declared with a specific type.

Common data types include int, double, char, and boolean.

Strings are handled using the String class and are immutable.

Java supports type casting for converting between compatible data types.

Local, instance, and static variables are the main variable types in Java.

Constants are defined using the "final" keyword.

Arrays allow storing multiple values of the same type in a single variable.

Java follows strict type checking to ensure robust code execution.

Understanding data types is key to efficient Java programming.

Practice declaring and using variables in simple Java programs.`

},

{

"id": 4,

"module": "Core Java Concepts",

"title": "Control Flow",

"content": `Java uses control flow statements to direct program execution.

"if-else" statements evaluate conditions to execute specific code blocks.

"switch" statements handle multiple conditions more efficiently.

Loops like "for", "while", and "do-while" enable repetitive task execution.

The "break" and "continue" keywords control loop execution flow.

Java supports logical operators like &&, ||, and ! for complex conditions.

Nested loops and conditional statements add flexibility to code.

Control flow ensures logical sequencing and decision-making in programs.

Efficient use of control flow minimizes errors and improves program clarity.

Practice building programs that incorporate these control statements.`

},

{

"id": 5,

"module": "Core Java Concepts",

"title": "Object-Oriented Programming (OOP)",

"content": `Java is inherently object-oriented, focusing on classes and objects.

A class is a blueprint for creating objects, containing attributes and methods.

Objects are instances of classes, encapsulating state and behavior.

Key OOP principles in Java include inheritance, polymorphism, and encapsulation.

Abstraction allows hiding implementation details from the user.

Interfaces and abstract classes enable flexible designs.

Java promotes code reusability and modular development.

Mastering OOP concepts is essential for effective Java programming.

Real-world examples demonstrate how Java simplifies complex systems.

Building small OOP-based programs helps solidify these concepts.`

},

{

```
  "id": 6,
  "module": "Intermediate Java",
  "title": "Exception Handling",
  "content": `Java uses exceptions to handle runtime errors gracefully.
  Use "try" and "catch" blocks to catch and handle exceptions.
  The "finally" block ensures cleanup actions are executed.
  Common exceptions include NullPointerException and ArithmeticException.
  Use "throw" to explicitly raise an exception in your program.
  Custom exceptions can be created by extending the Exception class.
  The "throws" keyword declares exceptions a method might throw.
  Proper exception handling improves program reliability.
  Java’s robust error-handling features are essential for real-world applications.
  Practice writing programs that handle various exception scenarios.`
},
{
  "id": 7,
  "module": "Intermediate Java",
  "title": "Collections Framework",
  "content": `Java's Collections Framework provides powerful data structures.
  Common interfaces include List, Set, and Map.
  ArrayList and LinkedList are dynamic alternatives to arrays.
  HashSet and TreeSet store unique elements with different orderings.
  HashMap and TreeMap manage key-value pairs efficiently.
  Collections offer methods for searching, sorting, and filtering data.
  Generics enhance type safety within collections.
```

Mastering collections is vital for managing data in Java applications.

Real-world projects rely heavily on Java’s robust collections.

Practice using collections to solve various data-handling problems.`

},

{

"id": 8,

"module": "Intermediate Java",

"title": "Threads and Multithreading",

"content": `Multithreading allows concurrent execution of code in Java.

Threads are created using the Thread class or Runnable interface.

The "synchronized" keyword prevents thread interference and data inconsistency.

Java provides the Executor framework for managing thread pools.

Use "wait" and "notify" methods for inter-thread communication.

Multithreading improves application performance and responsiveness.

Proper synchronization ensures thread safety in shared resources.

Debugging multithreaded programs requires careful analysis.

Real-time applications like games and servers use multithreading extensively.

Start with simple multithreading examples to grasp its concepts.`

},

{

"id": 9,

"module": "Advanced Java",

"title": "Working with Files",

"content": `File handling in Java uses classes from the java.io and java.nio packages.

Use FileReader and FileWriter for reading and writing text files.

BufferedReader and BufferedWriter enhance efficiency with larger files.

ObjectOutputStream and ObjectInputStream handle object serialization.

Java NIO offers advanced features like non-blocking I/O and file channels.

Exception handling is crucial to manage file operation errors.

Practice creating, reading, and writing files in different formats.

Real-world applications often involve processing file data.

Java’s extensive file handling APIs make it suitable for large-scale data management.

Understanding file I/O is essential for building robust applications.`

},

{

"id": 10,

"module": "Projects and Practice",

"title": "Mini Project: Student Management System",

"content": `Build a simple console-based student management system in Java.

Implement features like adding, updating, and deleting student records.

Use ArrayList or HashMap to store student data dynamically.

Practice reading and writing student data to a file for persistence.

Apply exception handling to manage invalid inputs gracefully.

Design the program using OOP principles for modularity.

Add a search feature to retrieve student information by ID or name.

Enhance the program with sorting options for better usability.

Testing and debugging ensure the program works flawlessly.

Completing this project boosts confidence in Java programming.`

}

];