**Basics of Programming**

1. **Interpreted and Compiled Programming Languages**

What are programming languages? Programming languages help us tell computers what to do. Computers don’t use human language; they use their own language, called machine code. Machines understand binary code, that is 1s and 0s. So, to make communicating with computers easier, we have human-readable programming languages.

Two common categories for programming languages are: Interpreted and compiled. These are broad categories, and there are many programming languages that are classified under compiled languages and interpreted languages. The purpose of the project helps a developer determine which programming language is best.

**Interpreted languages**

Interpreted language is also commonly referred to as scripted or scripting language. Programs written in interpreted or scripted language, like Python and HTML, run through the programming interpreter on your computer’s operating system or in your web browser. Remember that programming languages are in human-readable code, so, the interpreter takes the human-readable scripted code and then translates it into machine code, enabling the computer to complete the requested task. As computers and web pages have changed and advanced, some interpreted programming languages are outdated, and they are not as useful or relevant. Some other languages are more versatile and easier to learn, which means they are preferred over other scripting languages and used more often.

All interpreted programming languages need an interpreter to translate the source code. Translators are built into your web browser or they require a program on your computer to translate the code. Several different interpreted programming languages exist. Some examples of common interpreted programming language types are:

• JavaScript, a simpler scripting language that runs through the web browser interpreter

• Python, a language that is popular because it is easy to learn and use for developers

• Lua, a general purpose, lightweight game scripting language that is easy to learn and use

• HTML, a markup language used for formatting web pages

**Compiled programming languages**

Another category of programming languages is the compiled programming languages. Compiled programs are applications and programs, like your music app or your operating system, that you run on your computer or device. The programs are packaged --or compiled-- into one executable file. They are usually larger programs.

Compiled programs are used to help solve more challenging problems, like interpreting source code. Compiled programming languages are often referred to more simply as programming languages. A compiler program creates a program file, which runs the software. Simply put: It piles the code into one file that runs when you double-click on the app on your device. The program runs faster and it can be done repeatedly. The source code is converted from the programming language to machine code. Then it is compiled into one executable file.

Finally, the program runs when you select the icon or file on your device. Some examples of compiled programming languages are: C and its variations C++ and C#, which are used in many operating systems, like Microsoft Windows, Apple’s macOS, and the open-source operating system Linux. Java is another compiled programming language. It shouldn’t be confused with the interpreted language JavaScript. The Android OS is written in Java because it works well across computing platforms. Let’s look at this process: When you update to a new version of your operating system, like Microsoft Windows, your device might download an installation program. That program is made up of many files.

The files are written in a compiled programming language. These files give instructions to your device in machine code. The compiled program is running on your device. A compiled program that you commonly use is your device’s operating system, such as Linux, Microsoft Windows, Apple’s macOS, or Android. Your operating system is written in a compiled programming language, like C, C++, C#, or Java. Interpreted programming languages run scripts that are repetitive and need to be run often. Interpreted programming languages are more versatile and can be used across platforms as long as there is the correct interpreter.

Some examples of interpreted programming languages are JavaScript, Python, and HTML. Compiled programming languages are for more complex programs that complete larger tasks. Compiled programming languages are used for creating executable files that can run directly from your device. And some examples of compiled programming languages are C and Java.

1. **Comparing compiled and interpreted programming languages**

How do developers choose a programming language? Developers choose which programming language is best to use depending on what they have the most experience with and what they trust, what is best for their users, and what is the most efficient to use.

**What are interpreted and compiled programming languages?** Interpreted programming languages are used to create a scripted source code for smaller tasks. The source code goes through an interpreter. The interpreter is built into the operating system on a computer or on a web browser.

Compiled programming languages are used to create files. The files are executable files. They are then grouped in programs that you can run on a computer or device.

Interpreted programming languages are also called script code or scripting, because they are used to automate tasks. Interpreter programs read and execute the source code line by line like someone would read a script. Each time the program runs, the source code needs to be executed to receive the desired output, and source code written in one of the interpreted programming languages runs on almost any operating system with the right interpreter.

For example, imagine you have a client requesting information about webpage views for last month. Which type of programming language would be the most appropriate and efficient to create a program for this task? You would use one of the interpreted programming languages to write code for a script. The script would retrieve the webpage views data and then put it into a table for the client to read.

Compiled programming languages are also called programming languages for short. They are used for more complex programs that complete larger tasks, like running a spreadsheet program on your computer. A compiled programming language is used to write a larger program, usually installed on your device as an executable file.

Writing code in a compiled language does take longer, but the payoff is that the programs code runs faster, because compiled programs are installed on the device. And once the program is coded, the compiled program is grouped into one downloadable file. Remember that client who wanted information about webpage views? Now, they want a spreadsheet program to view and manage the data. A programmer could use the C programming language to create a compiled program like Microsoft Excel.

**So what are the major differences between interpreted programming languages and compiled programming languages?**

First, there's interpreted programming. These programs are available across multiple platforms or in the cloud. The scripting languages are easier to learn and use, and they're better for websites because they tackle smaller, repeated processes.

Then there's compiled programming. These programs are available to users with the same operating system coded in the same language. However, compiled programming languages are more difficult to learn and use, because they are larger programs with more parts, and they're better for larger tasks, like running the operating system on your computer. To better understand more of the similarities and differences of programming languages, let's take a look at some examples. A few of the most common programming languages are C, C++ and C#.

C is a compiled programming language. C is the original language, and C++ and C# are variations. C and its variations are case-sensitive languages. The C programming language is the basis for Windows and other common operating systems, and it takes more time to learn and use for coding but requires less memory and the code runs faster. Another common programming language is Java. It is a compiled programming language. It is also a case-sensitive, object-oriented programming language.

The Java virtual machine or JVM is required to run the code written in Java. It is the primary programming language for some operating systems, like the Android OS. And a benefit of Java is that it is a cross-platform language, which means it runs the same code on macOS, Windows, and Linux. Python is an example of an interpreted programming language. It is also referred to as a scripting language. Python is a popular general-use, case-sensitive programming language. It is used with Windows, macOS, and Linux operating systems, and with server-side web app code, and it requires the Python engine to interpret code.

JavaScript is another interpreted programming language. It is a scripting language that runs on the client side in web browsers. JavaScript is case sensitive. Simple scripts are run with HTML. Complex scripts are run in separate files. And while it sounds similar, it shouldn't be confused with Java, the compiled programming language.

Another example of a common programming language is HTML. It is an interpreted programming language. HTML stands for Hypertext Markup Language. HTML is mostly case insensitive with some exceptions, and it uses tags to format webpages on client-side web browsers. In this video, you learned that interpreted programming languages create source code that runs through an interpreter in your device's operating system or on your web browser. Compiled programming languages create executable files that are grouped in programs on your device. Compiled programming languages like C and Java are used to write larger programs, like operating systems and other executable files. And interpreted programming languages like Python and HTML are used to write code that can complete repetitive tasks within a web browser or a computer.

1. **Query and Assembly programming languages**

Let’s explore query and assembly programming languages. But first, we need to categorize programming languages into two levels – high-level and low-level. A high-level programming language is more sophisticated and uses the common English language to make its code more understandable and to increase the speed of coding and debugging programs. Examples of high-level programming languages include query languages such as Structured Query Language (or SQL), structured programming languages such as Pascal, and object-oriented programming languages such as Python.

In contrast, a low-level programming language uses a set of symbols to represent machine code. And examples of low-level programming languages include assembly languages such as ARM, MIPS, and X86.

A query is a request for information from a database. The database handles the query and searches its tables for the information requested and returns the results to the querying entity. When querying a database, it is important that both the user application making the query and the database handling the query are speaking the same language. In programming terms, writing a query means using predefined and understandable instructions to make the request to a database. This is achieved using programmatic code and this is what we refer to as a query language.

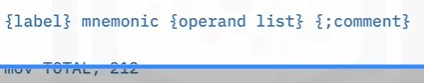
A query language may also be referred to as a database query language. By far the most prevalent query language for database queries and database management is SQL. However, there are other query languages available such as AQL, CQL, Datalog, and DMX. In addition to SQL databases, there is another type of database called NoSQL, which stands for Not Only SQL. The key difference between these two types of databases is their data structures. While SQL databases are relational and use structured, predefined, schemas, NoSQL databases are non-relational in nature and have dynamic schemas for unstructured data. A query language is predominantly used to request data from a database or to create, read, update, and delete data in a database. You will likely see the term CRUD used to refer to these last four key database operations.

Typically, a user enters a command to either make a query or perform a CRUD operation using syntax that is understandable to the database management system hosting the database. And a database typically consists of structured tables made up of multiple rows and columns of data. When a user performs a query, the database retrieves the data from the relevant rows and columns in the table and arranges it into some sort of order, ready to be returned and presented in the query results. Database queries are either a select command, or an action command, such as CREATE, INSERT, UPDATE, or a mixture of both. The term “statement” is more commonly used to describe these commands. Select queries request data from a database, whereas action queries manipulate data in a database. Query statements can also be used to perform other administrative functions such as creating users and modifying permissions.

This table lists some of the most common SQL query statements. Here are some simple syntax examples of common SQL statements.

As mentioned earlier, assembly languages are less sophisticated than query languages, structured programming languages, and object-oriented programming languages. As an assembly language uses a simple set of symbols to represent the 0s and 1s of machine code, it is categorized as a low-level programming language. Assembly languages are closely tied to the processor architecture from hardware manufacturers and therefore, each CPU type will typically have its own assembly language. For this reason, there are a large number of assembly languages in use today, which vary among hardware manufacturers. Assembly languages use a simple readable format for their statements, and they are entered one line at a time, with one statement per line.

Assembly language statements use the standard format shown here. In this syntax all fields in curly brackets { } are optional, and the statement has two main parts. The first part is the instruction (or the mnemonic), and the second part includes the parameters (or the operands). There may also be optional useful comments added on the end of the statement.



One other key difference with assembly languages is that they are translated using an assembler instead of a compiler or interpreter, and one statement translates into just one machine code instruction, as opposed to high-level languages where one statement can be translated into multiple machine code instructions. Assemblers translate assembly language into machine code using mnemonics such as Input (INP), Output (OUT), Load (LDA), Store (STA), and Add (ADD). The statements consist of opcodes that tell the processor what to do with the data, and operands that tell the processor where to find the data.

Query languages, structured programming languages, and object-oriented programming languages are categorized as high-level programming languages. Assembly languages are categorized as low-level programming languages. A query language is predominantly used to request data from a database or to manipulate data in a database. The most prevalent query language for database queries and database management is Structured Query Language (SQL). Select queries request data from a database, whereas action queries manipulate data in a database. You also learned that: Assembly languages use a simple set of symbols to represent the 0s and 1s of machine code. Assembly languages are closely tied to the processor architecture from hardware manufacturers.

Assembly languages are translated using an assembler instead of a compiler or interpreter. And assembly language instructions have a one-to-one association with their machine code counterpart.

1. **Understanding code organization methods**

Organizing is very important when it comes to reading, maintaining, and configuring code. Well-planned-out software design, usually using one of several methods, helps programmers write cleaner and more reliable code. Planning out code in a visual format helps improve the code base once it is written and reduces the chance of bugs and errors throughout the lifespan of a project. Organizing code before programming has a positive impact on the quality of the program and helps provide a consistent and logical format to use while coding.

There are two main methods of organizing code: flowcharts and pseudocode. The main difference between pseudocode and a flowchart is that the first is a basic, high-level description of an algorithm. An algorithm is a step-by-step sequence of solving a given problem. A flowchart is a pictorial representation of an algorithm showing the steps as boxes of various shapes and colors connected by arrows that indicate their order. Flowcharts are used in designing or documenting a process or program. Pseudocode provides a beneficial bridge to the project code because it closely follows the logic that the code will. Pseudocode also helps programmers share ideas without spending too much time creating code, and it provides a structure that is not dependent on any one programming language.

Flowcharts are especially beneficial for smaller concepts and problems, while pseudocode is more efficient for larger programming projects. And flowcharts provide an easy method of communication about the logic and offer a good starting point for the project because they are easier to create than pseudocode in the beginning stages. A flowchart is the graphical or pictorial representation of an algorithm using different symbols, shapes, and arrows in different sizes and colors to demonstrate a process or a program. The main purpose of using a flowchart is to analyze different methods of solving a problem or completing a process. Several standard symbols are applied in a flowchart, and you can easily highlight certain elements and the relationships between each part in the process. Some traditional flowchart shapes used for programming concepts are:

* Start/End (a capsule)
* Process (a rectangle)
* Decision (a diamond)
* Data (a parallelogram)
* And Connecters (as arrows).

A simple flowchart to represent how to add two numbers is easy to create.

A screenshot of a computer

AI-generated content may be incorrect.

The procedure starts with the capsule, and has a connector, or arrow, progressing to the next step, the data input (the parallelogram), in this case, input n1 and input n2. Then, another connector, or arrow, goes to the process (the rectangle) of adding the two inputs, Sum = n1 + n2. Another connector, or arrow, moves to the data (the parallelogram) showing the results, in this case, Print Sum. The last connector, or arrow, moves to the end of the process, the end capsule.

Flowchart software is an application that provides various functionalities to create flowcharts by providing the ability to drag shapes into the desired order using an easy-to-use editor. Flowchart software also provides team collaboration for creating flowcharts. Some of the well-known flowchart software programs include:

* Microsoft Visio,
* Lucidchart
* Draw.io
* DrawAnywhere.

Pseudocode is an informal type of programming description that does not require any strict programming language syntax or underlying technology considerations. System designers write pseudocode to ensure that programmers understand a software project's requirements and align code accordingly. Pseudocode is used for creating an outline or a rough draft of a program that summarizes a program’s flow but excludes underlying details. Pseudocode acts as the bridge between the programmer’s brain and the computer’s code executor that provides the ability to plan instructions that follow a logical pattern, without including all the technical details.

Pseudocode is a great way of getting started with software programming as a beginner without worrying about coding syntax. Pseudocode helps both programmers and non-programmers agree about the program’s goal and the basics of how the task should be done. Here’s one example of pseudocode written to check if the user entered an odd or even number: The next example displays the pseudocode written in C++: There are many advantages of using pseudocode over flowcharts.

A screenshot of a computer

AI-generated content may be incorrect.

The main benefit of pseudocode is that it’s simple and explains exactly what each line of an application should do. The coder can focus more on logic than on program language syntax. Removing the distractions of coding when using pseudocode makes the code development stage easier. Words and phrases in pseudocode represent lines of basic computer operations that simplify translation from the pseudocode algorithm to the specific programming language. Pseudocode allows programmers working in different computer languages to interact with each other. Pseudocode can be reviewed by different development groups easier than real code. Pseudocode is easier for non-programmers to read and enables quick and easy translation to any computer language. Writing pseudocode is more concise and easier to modify, so changes to the design can be easily incorporated. And lastly, unlike some flowcharts, pseudocode is usually less than one page.

Organizing and planning out software design enables programmers to write cleaner and more reliable code, and organized code is very important from a readability, maintainability, and scalability standpoint. Two main methods of organizing and planning software code are by developing flowcharts or writing pseudocode. A flowchart is a pictorial representation of an algorithm showing the steps as boxes of various kinds connected by arrows that indicate their order. And the main goal of pseudocode is to explain exactly what each line of a program should do, making the code construction phase easier for the programmer.

1. **Welcome to Branching and Looping Programming Logic.**

There are two major types of programming logic: branching and looping. Both types use Boolean expressions and variables: A Boolean expression is a type of programming statement with only two values, either "true" or "false." And variables have assigned values that are passed into a function or subroutine within a more extensive program. Computers use Boolean logic to make decisions. The computer takes one action if a Boolean expression is true and a different action if the expression is false.

Typically, a program consists of instructions that tell the computer what to do and data that the program uses when it is running. A variable has a value that can change, depending on conditions or information passed to the program. Boolean logic, along with variables, form the basis of programming. Branching logic is where a computer program makes a decision following a different set of instructions, depending on whether certain conditions are met during the program's execution. Each possible code pathway creates another branch. The branch of code that runs depends on the values assigned to the parameters of the branching procedure. There is no limit to the number of branches to implement complex logic.

The values of these parameters may be input by the user or generated by the output from a previous procedure. Branching contains constructs that occur and are processed to determine the path a program takes when running. Branching statements (also known as constructs) allow the execution flow to jump to a different part of the program. The common branching statements used within other control structures include: if, if-then-else, Switch, and GoTo. The if statement is a decision-making construct that guides a program to make decisions based on specified criteria. The if statement executes one code set when a specific condition is met (TRUE) or another code set if the condition is not met (FALSE). The if-then-else is a conditional construct that executes its substatement, which follows the “then” keyword. This only occurs if the provided condition is true.

The if-else statement extends the “if” statement by specifying an action if the “if” (true/false expression) is false. With the if-else statement, the program will execute either the true code block or the false code block, so something is always performed with an if-else statement. In computer programming languages, a switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via search and map.

GoTo is a statement found in many computer programming languages that performs a one-way transfer of control to another line of code. In contrast, a function call typically returns control.

Now, let’s look at the logic of looping programming. A loop is a sequence of instructions that continually repeats until reaching a specific condition. Typically, a particular process is performed, such as retrieving and changing data, and then some conditions are checked, such as whether a counter has reached a prescribed number. If it has not, the next instruction in the sequence is to return to the first instruction in the series and repeat the sequence. If the condition is reached, the next instruction "falls through" to the next sequential instruction or branches outside the loop. A loop is a fundamental programming idea commonly used in writing programs.

There are three basic loop statements: While For And Do-while. In a While loop, a condition is evaluated before processing the body of the loop. If a condition is true then and only then the body of a loop is executed. In a For loop, the initial value is performed only once, then the condition tests and compares the counter to a fixed value after each iteration, stopping the For loop when false is returned. In a Do-while loop, the condition is always executed after the body of a loop. It is also called an exit-controlled loop. There are two major types of programming logic: branching and looping. Both types of logic use Boolean expressions and variables. Boolean expressions have only two possible values, either true or false, and variables have assigned values that are passed into a function or subroutine within a more extensive program. Variables have values that can change, depending on conditions or information passed to the program. And branching is deciding what actions to take, while looping is deciding how many times to perform a certain action.

1. **Introduction to Programming Concepts**

To fully understand software programming, there are some fundamental programming concepts you need to know first. The first concepts you will look at are identifiers. Software developers use an identifier to reference a program component such as a stored value, a method, an interface, or a class, by assigning a custom-named label to it. If the identifier stores data, then the data values in the program can be one of two types: either a constant, or a variable.

A constant is a data item whose value does not change within a program. This could be a numerical constant such as the mathematical value of Pi, or it could be a text string that remains constant such as a player’s name within a game. Constants are also referred to as ‘named constants.’ You assign a value to a constant when you define it. For example, you might want it to refer to a numerical value within your program such as Pi, or a cost price, or a rate of tax, and then, rather than repeatedly retyping the same numerical value, you can just declare a constant for that value instead, and name it something meaningful like ‘pi\_value’, or ‘cost\_price’, or ‘tax\_rate’. There are a couple of major benefits to using constants in your programs; one is for ease of readability in your code, and the other is that if the specified value changes in the future, you only need to change it once on the constant, rather than finding every instance of that numerical value within your code and changing it. The other type of identifier used in programming is a variable.

As the name suggests, this kind of identifier is not constant; its value can change during the program’s execution, such as a user entering their age in an application or a high score in a game. Variables can be strings of text, numerical values, or any other type of data. Using a variable as an identifier is a useful way to refer to program items that are unknown to you, such as a username, a service, or a file name for instance. If you don’t use a variable, then you will need to hard code all the names and values in your program, which is not considered best practice. Variables can be declared and assigned a data type and initial value as they are defined, or you can decide not to assign an initial value when you define a variable, and instead have the value assigned later by instructions within the program. In addition to the identifier data structures already discussed, there are also special kinds of identifiers that can reference multiple elements in a program, and these are referred to as containers. Being able to specify multiple elements means that you don’t have to create a variable for every individual element.

This makes it faster and more efficient. For example, if you only want to store six numerical integers, then you could argue that you could just create six variables, one for each. But what if you need to store 1,000 integers or more? In such a case, defining 1,000 variables would obviously be an unwieldly and inefficient programming technique, so in this case you would use a container. There are two types of containers to specify multiple elements: arrays and vectors. The simplest of these container types is an array. In an array, a fixed number of elements of the same type are stored in sequential order, starting from index zero.

When you declare an array, you specify the data type of the values it contains, such as an integer, or boolean, or a string, and then the maximum number of elements it can contain. The syntax for declaring an array is to specify the data type first, then the name of the array, then the maximum size of the array in square braces. In contrast to arrays that have a fixed size, vectors have a dynamic size, and they will automatically resize themselves as you add elements to them or remove elements from them. For this reason, you may also see them referred to as dynamic arrays. Because they are dynamic in nature, vectors take up more memory space than arrays, and their elements also take a little longer to access than elements in an array, as they are not stored in sequential memory locations. The syntax for declaring a vector is to specify the container type of the vector first, then specify the data type in angle brackets, then the name of the array. Note that because it is a vector, you do not need to specify a maximum number of values it can contain because the size is not fixed, it is dynamic.

In this video, you learned that: Software developers use an identifier to reference a program component. If an identifier stores data, then it can either be a constant or a variable. A constant is a data item whose value does not change within the program. A variable is not constant; it can change during the program’s execution. In an array, a fixed number of elements of the same type are stored in sequential order, starting from zero. And vectors have a dynamic size, and they automatically resize themselves as elements are added or removed.

Next, let’s look at the fundamental concept of functions. Functions are a consequence of the modular programming software development methodology that encourages the separation of a program into multiple modular components, where each performs a specific task within a program. So, a function is essentially a piece of structured, stand-alone, and reusable code that will perform a single specific action. This enables software developers to take a substantial, complex program and divide it into smaller, more manageable, and focused pieces. Although some programming languages may refer to them as something else, such as subroutines, procedures, methods, or modules, most modern programming languages refer to them as functions.

Functions take in data as an input, then process the data, and then return the result as an output. There are essentially two types of functions. Standard library functions are the built-in functions provided by the programming language. Common examples include the ‘If’, ‘Else’, ‘While ’ and ‘Print’ functions. But programming languages also allow you to write your own functions. And once you’ve written a function, you can use it over and over again. The way that the blocks of code that make up a function are identified is different across programming languages.

Some use braces, some use begin and end statements, and others use indentations for example. There are a few steps to using functions. The first thing you need to do is define (or create) a function. When you define a function, you provide a function keyword, then give the function a unique name, and you provide the statements that make up the body of the function. Once a function has been defined, it then needs to be called (or invoked). When you call a function, the specified actions within the function are performed using any specified parameters. While defining and calling functions are common to all programming languages, some programming languages, such as C and C++ , also require you to declare a function.

Next, let’s look at the concept of objects. Understanding what objects are is key to understanding object-oriented programming. Object-oriented programming (or OOP) is a programming methodology that is focused on objects rather than functions, which is what procedure-oriented programming is focused on. The objects themselves will contain data in the form of properties (or attributes) and code in the form of procedures (or methods). The key distinction between the two methodologies is that where procedural programming uses methods to operate on separate data structures, OOP packages them both together, so an object operates on its own data structure. Consider the real-world objects in your life such as your car, bike, TV, or your washing machine, and ask yourself the following two questions: “What states can the object be in?” and “What behaviors can the object perform?”

When you think about your answers to these questions, you will find that the answers to these questions vary across the objects. More complex objects will typically have more potential states they can be in and will be able to perform more behaviors. In programming, a software object is similar to a real-world object, conceptually speaking, in that they too consist of states (or properties) and behaviors (or methods). Software objects can be anything, such as a Windows service, a user account, a database table, or a system folder. Objects store their properties in fields (referred to as variables in some programming languages), and expose their behaviors through methods (referred to as functions in some programming languages). In this video, you learned that: • A function is a piece of structured, stand-alone, and reusable code that will perform a single specific action. • The defining and calling of functions is common to all programming languages.

• Object-oriented programming is a programming methodology that is focused on objects rather than functions. • Software objects consist of properties and methods.

* Interpreted programming languages create source code that runs through an interpreter and is built into your operating system (OS) on your computer or on your web browser.
* Compiled programming languages create executable files that are grouped in programs on your computer or device.
* Query languages, structured programming languages, and object-oriented programming languages are categorized as high-level programming languages and assembly languages are categorized as low-level programming languages.
* The two main methods of organizing and planning code are by developing flowcharts and by writing pseudocode. Flowcharts are pictorial representations of algorithms and pseudocode is an explanation of the function of each line of a program.
* To reference a program component, software developers use an identifier, which can either be a constant or a variable.
* A function is a piece of structured, stand-alone, and reusable code that will perform a single specific action.
* Object-oriented programming is a programming paradigm based on the concept of objects, which contain data and behavior through attributes and methods.