

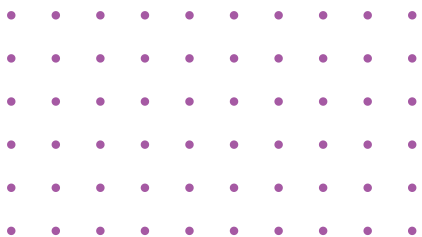
# WEEK 3 - PART 1

## INTRO TO CODE -

### OVERVIEW AND ESSENTIAL CONCEPTS

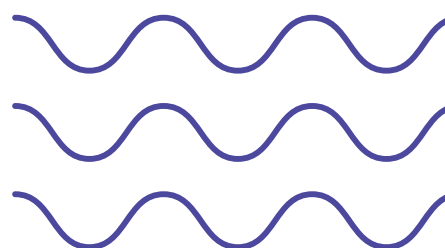
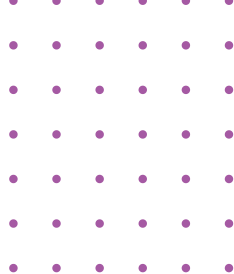


LINNAEUS  
UNIVERSITY  
SUMMER 2024



# TABLE OF CONTENTS

<b>3</b>	what is code?	
<b>5</b>	bits - basic units of data	
<b>6</b>	programming languages	
	human language vs code	..... 7
	programming languages - a comparison	..... 9
	syntax - examples	..... 10
	translating code	..... 12
<b>13</b>	code evolution	
<b>15</b>	abstraction - code libraries & functions	
<b>17</b>	thinking in code	
	the coding process	..... 18
	what is a program	..... 19
	the essence of coding	..... 20

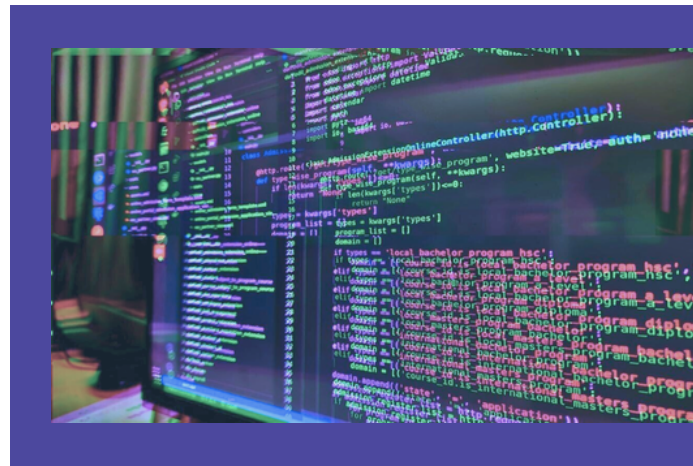


# WHAT IS CODE?

Code is the language we use to communicate with computers. At its core, **coding is about translating human intentions into a form that machines can understand and execute.** When we write code, we are essentially creating a set of instructions that will be converted into binary language, following a format of simple yes and no decisions that the computer can process.

**To communicate with the computer, programmers break down complex tasks into these simple binary-like decisions,** writing instructions using these straightforward yes-or-no / on-and-off choices.

Each instruction is translated into a binary format - after all computers operate solely on binary code, consisting of 1s and 0s.



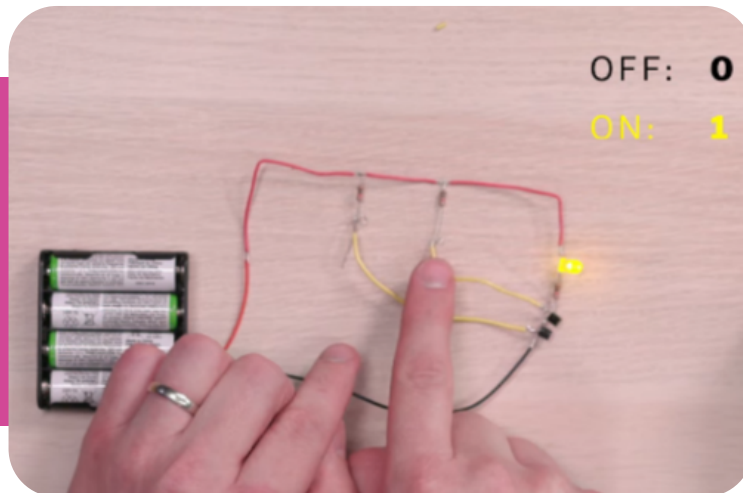
A 1 might mean "yes" or "on," while a 0 might mean "no" or "off. These binary digits serve as signals that the computer reads to perform tasks and process information.

In the realm of computing, the binary system's use of just two symbols—1 and 0—demonstrates a remarkable capacity for complexity.

Morse code, which employs only two signals—a short beep and a long beep—manages to encode and express an entire alphabet and complex messages using just these two simple sounds.

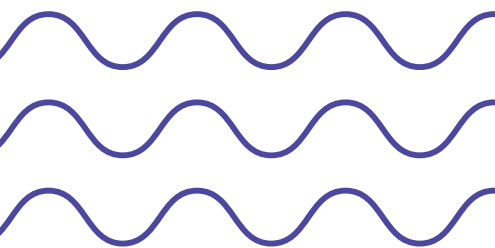
Similarly, binary code leverages its two states to represent all forms of digital data. **Each combination of 1s and 0s in binary code can encode vast amounts of information, from simple text to intricate graphics and sophisticated software.**

This simplicity reflects how computers function at a fundamental level. **Computers are designed to recognize and process electrical signals. A binary system perfectly suits this requirement because it maps the two states—on and off—into binary digits.**



When you write code in a programming language, it is initially in a human-readable format, such as Python or JavaScript.

This **code is designed to be understandable by people**, with syntax and semantics that make logical sense.



However, **computers cannot directly execute this human-readable code.** To bridge the gap between human-readable code and what a computer can execute, the code must be translated into a form the computer understands—machine language.

**The code gets converted to machine code, which is a series of binary digits (1s and 0s)** that the computer's hardware can process directly.

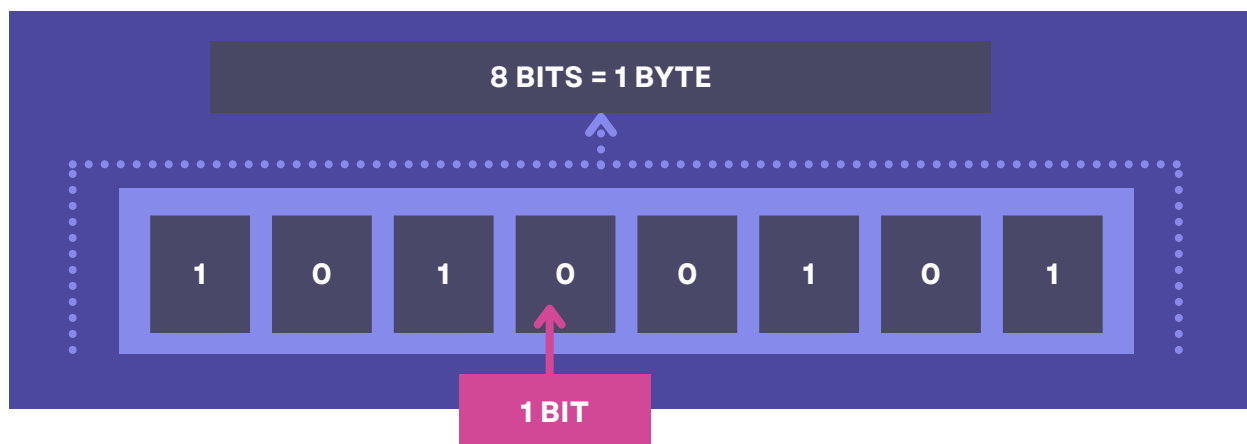
# BITS - BASIC UNITS OF DATA

**These binary digits are called bits.**

Each bit, short for binary digit, **represents a single 1 or 0**, and it is through these bits that the computer carries out the instructions provided by the program. Bits are the most basic units of data in computing—the atoms of a computer. When millions of bits work together, they power everything a computer does.

**Typically, these bits are grouped into sets of eight, known as a byte.**

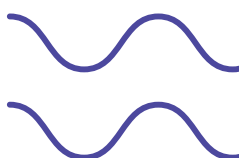
Consider an image file on your computer. The image is stored as a series of bytes. **For instance, a 1.1-megabyte image file contains about 8.8 million bits.** Each bit contributes to the overall image data, allowing the computer to display the picture correctly.



**A computer's hardware is made up of millions of tiny electrical circuits, each of which can either carry an electrical current (on/1) or not (off/0).**

Computers operate solely on electrical signals, meaning that **everything programmers do ultimately boils down to a series of on-and-off electrical charges.**

When a computer receives a binary instruction, it uses transistors—tiny electronic switches—to control the flow of electricity. Transistors are fundamental to this process, acting as gates that either allow current to pass through (on) or block it (off).



A simple instruction like adding two numbers might be translated to a binary sequence like 11001010. **When a binary instruction (like 11001010) comes in, it tells specific transistors to either let electricity flow (turn on) or block it (turn off).** The result of all this switching on and off is that the computer performs the desired action, like calculating a result or displaying an image on your screen.

# PROGRAMMING LANGUAGES

**Writing programs in zeros and ones doesn't scale well**—imagine trying to remember the exact bits needed to represent a simple character like "h."

As coding evolved, the need for a more accessible way to program became clear. **The history of coding has seen an evolution towards making code more and more like human language**, culminating in programming languages.

CHARACTER	H	O	P	E
BINARY VALUE	01101000	01101111	01110000	01100101
BITS	8	8	8	8

**A programming language is a formal system used to communicate commands** to a computer, enabling us to create programs that control its behavior.



They use **structured logic** to perform tasks that allows a program to make decisions and execute different actions based on certain conditions.

Developers write code using programming languages such as Python, Java, or C++, which **allow them to write instructions in a more human-readable form**

## HUMAN LANGUAGE VS CODE

Code itself consists of a series of statements and commands that specify the desired behavior or actions to be taken by the computer, written in a **syntax that is specific to the programming language being used.**


While programming languages differ in syntax, rules, and keywords—**much like how English and Swedish differ in grammar and vocabulary—the core concepts remain similar.**

For example, whether you're speaking English or Spanish, you can ask someone to "please pass the salt" or "por favor, pásame la sal."



Just as human languages use verbs, nouns, and tenses to convey meaning, **programming languages use similar structures** to express algorithms, perform computations, manipulate data, and control computer systems.

While the core idea behind code is the same, **different programming languages present this concept in various ways.**



**Just as spoken languages have different ways to express the same idea, programming languages use various methods to give instructions to computers.**

**In programming languages, you might use different syntax to perform a task, but each language is built on similar principles.**

Still, comparing computer languages to human languages reveals several key differences.

Computer languages require **strict adherence to syntax and grammar**, where even a minor error can disrupt the code.

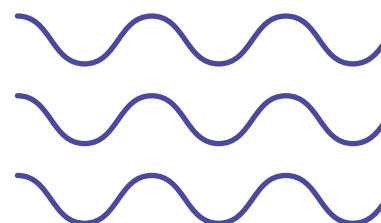
**Human languages, by contrast, are more flexible and can handle variations in grammar and syntax.**

**Ambiguity also differentiates the two.**

Human languages can be ambiguous, with words or phrases having multiple meanings depending on context.

For example, asking "Do you want to do something?" can be interpreted in many ways based on tone and context.

In contrast, computer languages aim to eliminate ambiguity with clear and precise instructions.





# PROGRAMMING LANGUAGES - A COMPARISON

Different programming languages excel at different tasks, even though most can be used for a variety of purposes.



## Python

is a versatile and beginner-friendly language known for its simplicity and readability. It is widely used for web development, scientific computing, data analysis, artificial intelligence, machine learning, and automation.



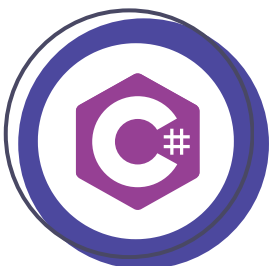
## JavaScript

is primarily used for web development, enabling dynamic and interactive features on websites. It is also commonly used for frontend and backend web development, mobile app development (using frameworks like React Native), and server code.



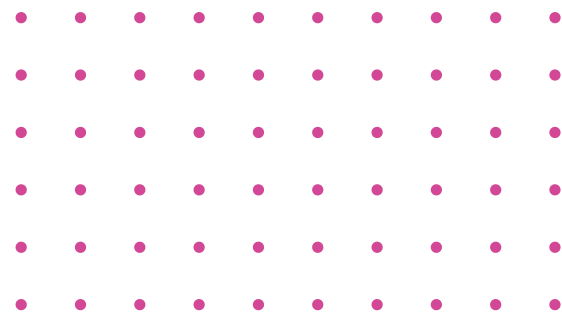
## Java

is a general-purpose language known for its platform independence. It is widely used for building enterprise-level applications, Android app development, large-scale systems, and server-side development.



## C#

(pronounced "C sharp") is a language developed by Microsoft and is mainly used for Windows application development, game development using the Unity engine, and building enterprise-level applications on the .NET framework.





## C++

is a powerful and efficient language used for system programming, game development, high-performance software, embedded systems, and complex applications that require low-level control and performance optimization.



## Swift

is a language developed by Apple for iOS and macOS app development. It is known for its safety features, modern syntax, and performance. Swift is used to create applications for iPhones, iPads, Macs, Apple Watches, and Apple TV.

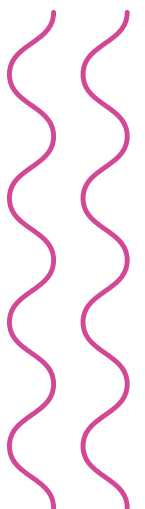
Despite these differences, at the end of the day, all programming languages are ultimately translated into machine code—the binary code that computers can execute.

## SYNTAX - EXAMPLES

**On the next page is a very simple line of code that prints "Hello, World!" to the screen, written in Python, JavaScript, Java, C#, C++, and Swift.**

This is one of the most basic examples you'll encounter when starting with any programming language.

**Don't worry if there are parts you do not understand - this is simply to visually highlight differences and similarities between the different programming languages!**





## PYTHON

```
print("Hello, World!")
```



## JAVASCRIPT

```
console.log("Hello, World!");
```



## JAVA

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```



## C#

```
using System;  
  
class Program {  
    static void Main() {  
        Console.WriteLine("Hello, World!");  
    }  
}
```



## C++

```
#include <iostream>  
int main() {  
    std::cout << "Hello, World!" << std::endl;  
    return 0;  
}
```



## SWIFT

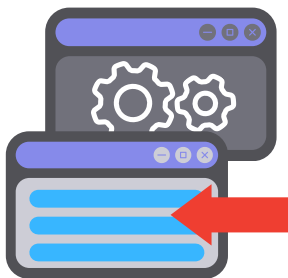
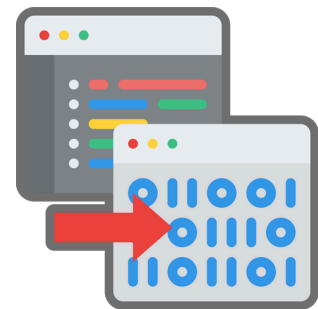
```
print("Hello, World!")
```

# TRANSLATING CODE

Despite their different syntax, all programming languages must ultimately be translated into binary code—1s and 0s—for the computer to understand and execute the instructions. This translation bridges the gap between human-readable code and the binary language that computers use.

This translation is performed by tools called compilers and interpreters.

A compiler takes the entire code and translates it into machine language all at once, resulting in an executable file. This file contains the binary instructions that the computer can run directly. For example, a compiler might convert a C++ program into a binary file that can be executed on a computer.

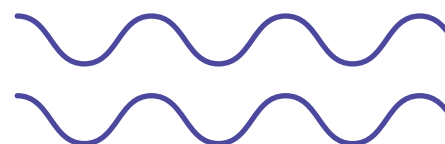


An interpreter translates code line-by-line, executing each line in real-time, much like a real-time translator who translates a conversation on the fly. This allows for immediate feedback and easier debugging. For example, Python uses an interpreter to run code interactively, translating each instruction into binary as it is executed.

For example, when you write JavaScript code for a website, your **web browser acts as an interpreter**. It reads and executes the JavaScript code on the fly, line-by-line, directly in the browser. If there's an error or if the code needs to change, you see the results immediately.

A compiler, on the other hand, is like a translator who first translates an entire book into another language before anyone reads it.

For example, when you write Java code, a compiler takes all of your Java code at once translates it into machine code, and packages it into a file, such as an .exe file. Once a program is compiled into an executable file, you can run it on any computer without needing the original code or any special software to translate it.



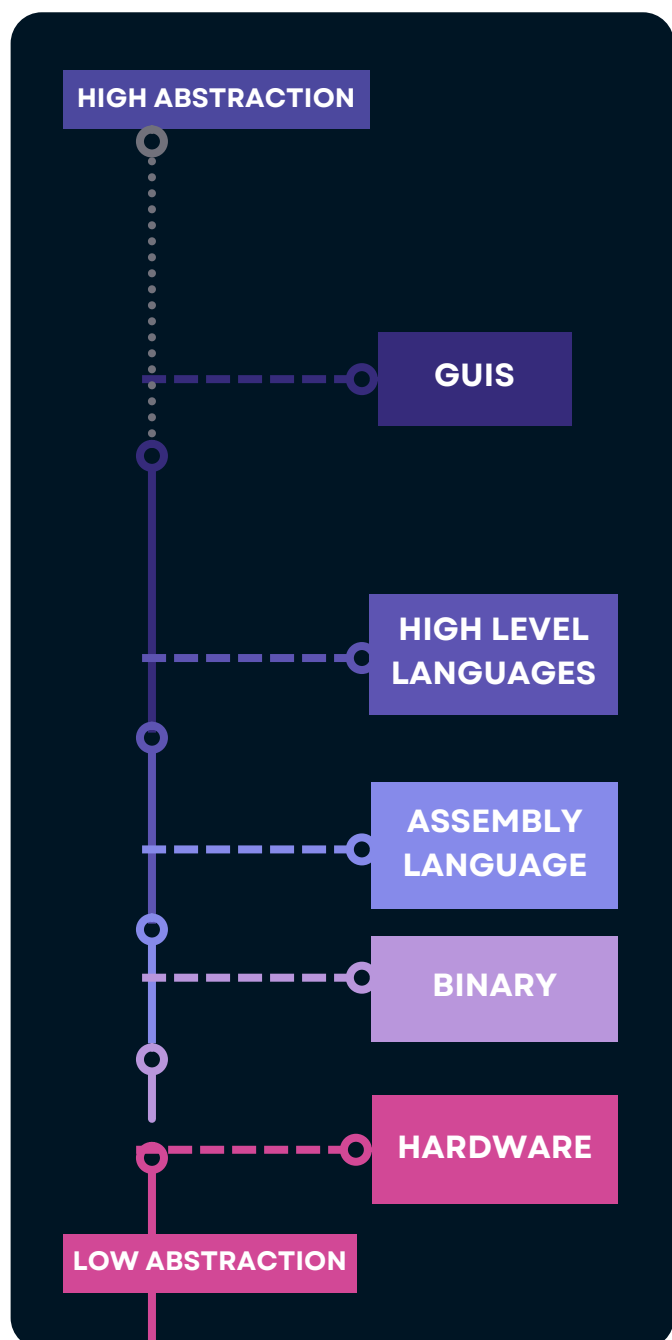
# CODE EVOLUTION


Programming languages vary in how closely they resemble the binary code that computers understand directly. **This variation is referred to as the level of abstraction.**

Starting from binary code and low-level languages, **we have evolved to high-level languages**, which simplify programming by abstracting away complex hardware details and making code more intuitive and easier to write.

**High-level programming languages are designed to be more user-friendly**, providing a simplified, user-friendly interface, and are much closer to human language.

**These are called "high-level" because they provide a higher level of abstraction away from the hardware.** They are designed to be closer to human language, making them easier for programmers to understand and use. High-level languages handle complex details of the computer's hardware and allow programmers to write code that is more intuitive and less concerned with the underlying machine operations.





In contrast, **low-level programming languages are called "low-level" because they operate much closer to the hardware.** The lower down we are on the chain of abstraction, the closer we get to binary code—the ones and zeroes at the core of computing.

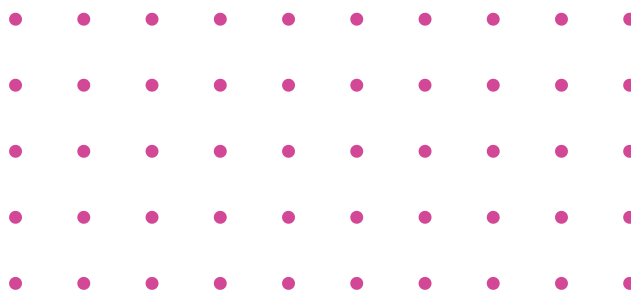
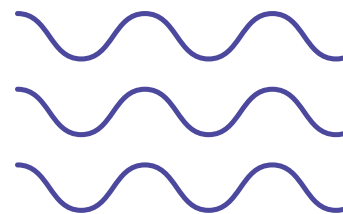
Low-level languages provide direct access to the computer's hardware, meaning that the details of how the computer works are exposed and must be managed by the programmer. They require more detailed and specific instructions about how to control the computer's components, such as the memory and processor.

**Consider a simple task like adding two numbers. In a high-level programming language like Python, you can write `result = 5 + 3` to get the sum.**

This code is easy to read and understand, thanks to its high level of abstraction. These higher-level languages also help us take care of "low-level details"—like how data is stored in memory, how to manage the processor's time between tasks, etc.

In contrast, **when working with low-level languages such as Assembly or machine language (binary code), you would need to write multiple lines of code** to perform tasks like loading numbers into registers, executing the addition, and then storing the result. This low-level code requires more detailed and precise instructions compared to higher-level languages.

**The more abstract languages still all boil down to the same ones and zeros - we've just found better ways to organize them, and "hide them" behind easier to understand, human-like language.**



# ABSTRACTION - CODE LIBRARIES & FUNCTIONS

As programming languages evolved, programmers frequently encountered similar tasks and challenges. **To address this, they began adding pre-defined functionality into the language that could handle these common tasks.**

This way, when other programmers use the language, they can access these built-in functions, which makes coding more efficient and reduces the need to repeatedly write the same code. **These functions are included in libraries-collections of pre-written code designed to perform common tasks.** When you download a programming language, it typically comes with a set of these libraries, giving you access to a range of built-in functions right away.



Without using functions, you might have to **manually write out every step of the addition process, including converting numbers to binary, performing bitwise operations, and handling carry-over values.**

**Think about a coffee machine.** When you use a coffee machine, you press a button for “coffee,” and the machine takes care of grinding beans, heating water, and brewing the coffee. **You don’t need to understand how each part of the machine works internally. You just enjoy the result—your coffee.** Similarly, in programming, using `math.add` is like pressing the coffee button. You don’t need to manage the intricate details of how addition is performed internally; you just use the function and get the result you need.

When coding, writing “add” is simpler than working with binary code “01000111”.

Now, if we group the functionality for “add” and “subtract” under a single category called “math,” we simplify how we think about these operations - and how we use them.

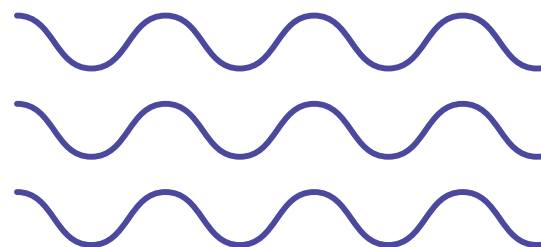
By using “math.add” or “math.subtract,” we place all related mathematical functions under one easy-to-understand label, “math.”

This approach makes the code simpler and more organized, as it lets you access these functions without having to write out all the detailed code yourself.

These functions are **included in the language's library**, so you just need to know how to use “math.add” or “math.subtract” to get the results you need, rather than coding each operation from scratch.

This approach is **more abstract because it groups these operations under a user-friendly label** - “hiding” these detailed instructions inside other code or functions - making the code easier to read and manage.

**In programming, using math.add is like pressing the coffee button.** You don't need to manage the intricate details of how addition is performed internally; you just use the function and get the result you need.



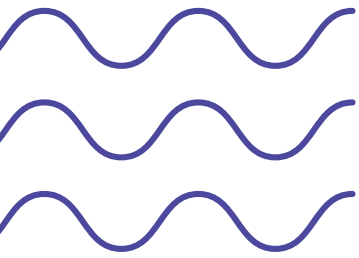


# THINKING IN CODE

Computers only understand 0 or 1, yes and no, on and off.

This binary approach translates into a "yes or no" logic - a way of thinking and communicating without ambiguity. After all, a switch cannot be partially turned on or off.

When programming, expressing your instructions in this binary logic helps us to frame conditions and decisions in a way computers can understand.



Suppose you want to control a light based on whether it's day or night.

The condition you're checking is whether it's daytime or nighttime.

**You need to formulate this condition in a way the computer can understand, using a binary approach where the condition results in either true (1) or false (0).** For simplicity, let's say you check if it's daytime. If it is, you want the light to be off; if it's nighttime, you want the light to be on.

**Daytime Check: Is it daytime? (Yes = 1, No = 0).**

**Based on the result of this binary check, you instruct the light to turn on or off. Think of the condition as a question with a clear "yes" or "no" answer.**

For instance, "Is it daytime?" translates to a binary outcome:

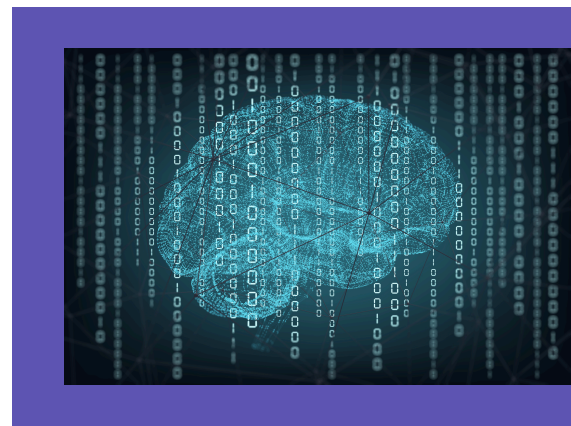
Daytime (Yes = 1), Nighttime (No = 0).

**In code, it might behave like this:**

**Check if `it_is_daytime` is true (Yes = 1).**

**If Yes (1): Turn off light.**

**If No (0): Turn on light.**



# THE CODING PROCESS

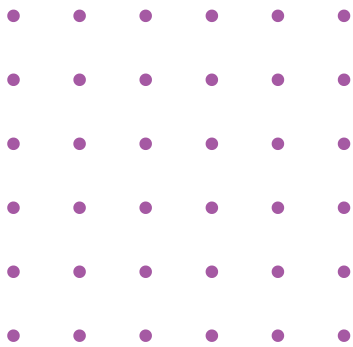
**Coding is a systematic, multi-stage process that involves several key stages: writing instructions, testing, and deploying them to various environments.**

Initially, developers write code using a programming language, such as Python, JavaScript, or Java. **This code, written in an Integrated Development Environment (IDE)**, consists of statements and commands that direct the computer to perform specific tasks.

**The IDE operates within a runtime environment necessary for the IDE to execute the code**—like Node.js for JavaScript—installed on the developer's computer. For instance, when building a website, developers can code and immediately test their work within this local environment.

While coding, they can run the code to see if it behaves as they want.

**If working on a website, they can start up their website locally on their computer with the code they've written. It will behave like the finished website**, and the programmer can test it to make sure that everything is working properly by executing the code inside the IDE to see how the finished website will look and to identify and correct any errors.

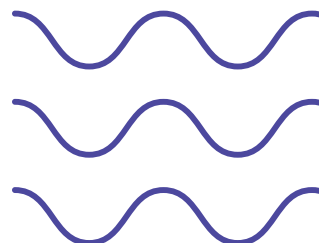


**Once the code has been tested and refined, it is deployed to various environments where it will be used.**

Deployment involves transferring the code from the development environment to its intended operating environment - moving the code to where it will be used.

**This may involve placing the code on a web server for a website**, uploading it to a cloud platform for an application, or transferring it to other computing devices.

**Deployment makes the executable code available for users to interact with.**



# WHAT IS A PROGRAM

A program is a collection of instructions that are written in code whose job is to carry out a specific task or set of tasks - a **sequence of coded commands** that are designed to be executed by a computer or other computing devices.

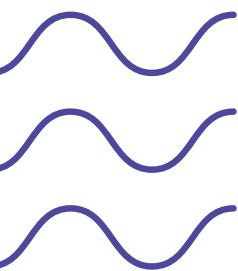
It can be as simple as a few lines of code or as complex as a large software application, and perform a wide range of tasks, such as mathematical calculations, data processing, file manipulation, user interaction, and more. Each instruction has a specific purpose and action associated with it. Programs frequently handle data, including user inputs, stored files, and information produced during their execution. They use this data to perform tasks like processing, modifying, and analyzing it.

```
python Copy code  
  
name = input("What is your name? ")  
print("Hello, " + name + "!")
```

This code snippet is a basic example of how programming languages allow us to write instructions that the computer can execute to perform specific tasks - **a simple Python program that asks for your name, then greets you with "Nice to meet you, [Your Name]!"**

When you run this code, it will **prompt the user to enter their name and then greet them with a personalized message.**

**Had we put the code to say "Nice to meet you, [Your Name]!" first, the computer would not have had any input from the user yet, and would not have saved any name to the computers memory.** When it would have looked for the name in it's memory, there would have been nothing to retrieve from it's saved data - and the program would have failed and given an error.





# THE ESSENCE OF CODING

**That's what coding is at its essence.**

**It's taking something we understand, like a set of instructions, and translating it into a language that a computer can execute.**

In this way, coding bridges the gap between human thought and machine action.

**Some people see coding as a form of logic**, where the act of coding involves translating what we want to achieve into a series of logical statements.

**Others view it as a way of communicating with machines**, where coding is essentially the process of speaking the machine's language to convey instructions.

Each programming language, from Python to Java, has its own syntax and set of rules, but they all serve the same fundamental purpose. By learning to code, we gain the ability to communicate with machines, create software, and build systems that power everything from smartphones to space exploration.