WEEK 3 - PART 2

INTRO TO CODE -

PROGRAMMING BASICS

LINNAEUS
UNIVERSITY
SUMMER 2024

TABLE OF CONTENTS

CODE - THE NITTY GRITTY

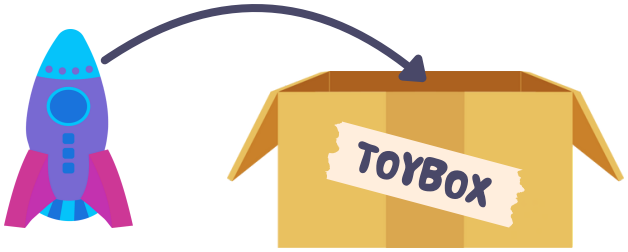
In programming, mastering the basics of how code works and how to structure it logically is key to creating effective and reliable software.

Previously we have said that code consists of statements and commands.

When we say that code is composed of statements that define and control data, we're referring to the different types of instructions that make up a program. Each statement in a program serves a specific purpose, such as defining a variable. In essence, code is composed of statements that define and control data and expressions or instructions that perform operations or actions using that data.

VARIABLES

In the world of programming, variables are like **containers that hold different values - different types of data.** They allow us to **store and manipulate data**, making our programs dynamic and adaptable.



variable name = toybox

Retrieve toybox to access the rocket

Imagine variables as **labeled boxes** where you can **store various things**, such as numbers, text, or complex data. These **boxes have names**, called identifiers. An identifier is the name you as the programmer give to a variable (or other elements) in your code. It acts as a label or a **tag that you can use to refer to the stored data.**

Just like a label on a physical box helps you identify what's inside without opening it, an identifier helps you identify the data associated with that variable in your code.

VARIABLES are of specific **DATATYPES**, have specific **VARIABLE NAMES** (identifiers), and contain a **VALUE**.

toy toybox = rocket

DATATYPE is **toy.** This box is made to fit a toy.

Stored in your **garage** when not used

When you **need the ROCKET** you go to the garage and **look for TOYBOX**

int age = 21;

DATATYPE is **int** (integer). This variable is made to fit a whole number.

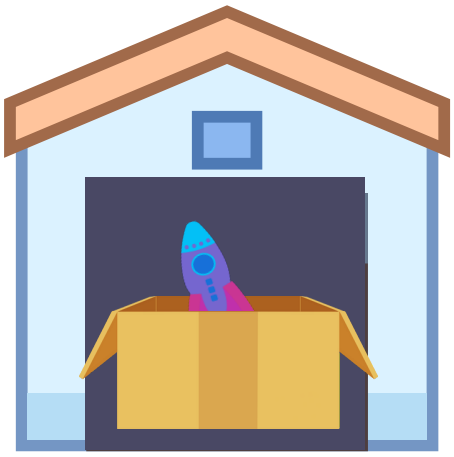
Stored in your computers **RAM** when coding.

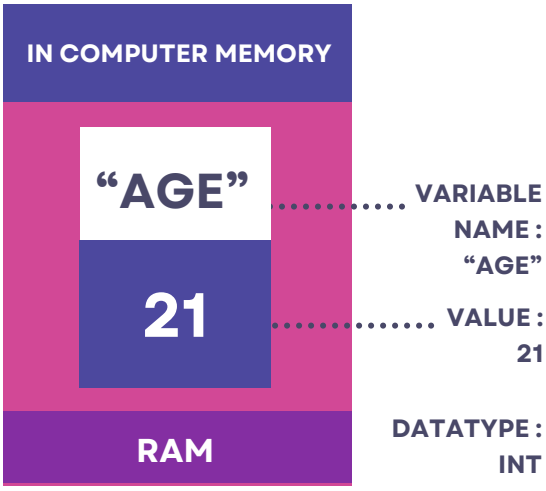
When you **need the VALUE** of a variable, you reference the **VARIABLE NAME**

VARIABLE NAME: "TOYBOX"

VALUE: ROCKET

DATATYPE: TOY





Instead of buying a new rocket anytime you want to play, you store it for whenever the need arises. You remember you put the rocket in a box labeled **"toybox"** in your garage, where go **whenever you want the rocket out.**

Instead of writing "20" all the time, the computer remembers what's in the container named **"age"**. We can refer to this "container" stored in the computers memory in order to **access the VALUE 20.**

```
let toy;
```

Variable Declaration is when you **introduce a variable** in your code.
For example, **int age; in Java** or **let toy; in JavaScript** defines a variable named age without yet giving it a value. When you declare a variable, you are essentially telling the computer to **set aside a space in memory** for that variable. However, if you **don't assign a value to the variable at the time of declaration**, the space is reserved, but the value stored in that memory space is either undefined, uninitialized, or set to a default value, depending on the programming language you're using.

```
toy = rocket;
```

Variable Assignment is when **you assign a value** to a variable.
For example, **age = 20; assigns the value 20 to the variable age.** Assignment is how you define what each variable holds at any given moment.
Once you assign values to variables, you can use them throughout your code.

```
let toy =
```

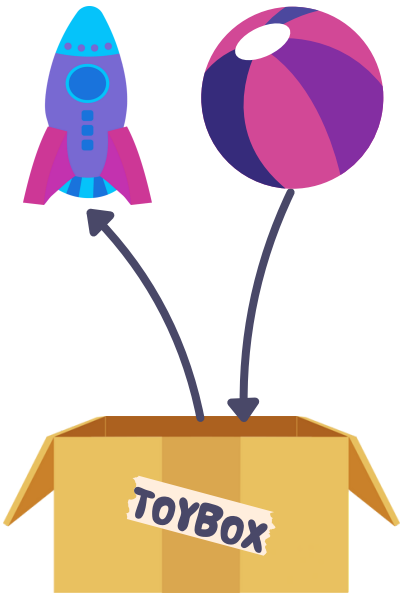
Initializing a variable refers to the process of **assigning an initial value to a variable at the time it is declared. This combines two actions**: declaring the variable (which tells the program to set aside a space in memory) and assigning it a value (which puts something into that space).

Every variable you declare consumes memory, as it needs space to store its value. Before a variable can be used in your program, it must be declared so that the computer can allocate memory for it. **Once declared, the variable is stored in the computer's memory, allowing you to access and manipulate its value throughout the execution of your program.**

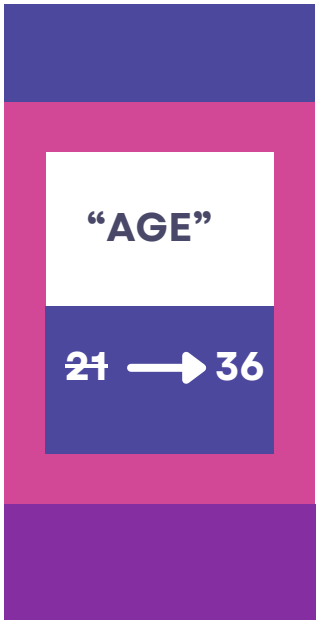
By declaring a variable, you effectively reserve a chunk of memory that **persists for the duration of the program,** enabling you to reuse the variable and its value multiple times as needed.

You **can also change the VALUE of a variable** once its value has been assigned. This is called **reassigning.**

Reassigning a variable involves **changing its value after it has been initially set.** The **new value replaces the old one**, and any further usage of the variable will reflect this updated value. This capability allows variables to be flexible and dynamic, adapting to the needs of your program as it runs.



You keep the box, and the label.
You remove the rocket and put a ball inside instead.
This changes the variable ‘toybox’, whose VALUE was rocket to ball



let age = 21;
age= 36

This changes the variable ‘age’, whose VALUE was 21 to 36

Remember, you DO NOT create a new variable - you change the VALUE of the old one.

DATATYPES

In programming, **data types define the kind of data that a variable can hold.** They help the computer understand what kind of data it is working with and how to handle it.

```
int age = 21;
```

In this image the **DATATYPE is int,** which stands for **integer.**
When you declare a variable as an int, **you're telling the computer that this variable will store specifically integers** - whole numbers (with no decimals).

This allows the computer to allocate the appropriate amount of memory for this type of data and to perform arithmetic operations correctly.

Java and C# use specific data types like int (integer) to provide precise control over data and memory, whereas **JavaScript handles numbers differently.**

In JavaScript, the **"number" type covers both integers and floating-point numbers (numbers that have decimal points - 0.7 for example).**
In languages like Java and C#, you need to specify whether a number is an integer (int) or a floating-point number (float).

Different languages may use different terminology and types, but the **fundamental concepts of handling whole numbers and decimal numbers are consistent.**

STATICALLY VS. DYNAMICALLY TYPED LANGUAGES

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

Java and C# are statically typed, meaning the data types of variables must be explicitly defined at compile-time (remember, they require compilation).

By specifying data types, Java and C# can **allocate appropriate amounts of memory and optimize performance** for different operations.
Specifying datatypes also allows the compiler to check for type errors before the code runs, improving code safety and reliability.

JavaScript is a dynamically typed language, which means you don't need to define data type when you declare a variable.
Datatypes are more flexible and less type-specific - variables in JavaScript do not have a fixed type, and the type can change at runtime.
The type is determined by at runtime based on the assigned value.

The dynamic typing system **offers more flexibility and simplicity** for developers, especially in a language designed primarily for web development, where ease of use and rapid development are often prioritized. Python is also dynamically typed, which means that variable types are determined and checked at runtime rather than during compilation. This flexibility can simplify coding but **may also lead to unexpected behavior if not managed carefully.**

In JavaScript, we don't specify DATATYPE.
Instead, you declare variables using either var, let, or const.
Regardless of which keyword you use, you do not specify the type of data the variable will hold - JavaScript deduces the data type from the value assigned.

CODE IS SEQUENTIAL

When you write a computer program, the code runs from the top of the file to the bottom, one line at a time. This means that the order in which you write your instructions matters.

The **computer follows the instructions EXACTLY as you give them**, so if something is written in the wrong order, it can cause the program to not work as expected or even crash.

The code must be **logically structured so that each instruction follows a sequence** to achieve the desired outcome.

Imagine you want to bake a cake. The correct steps are -

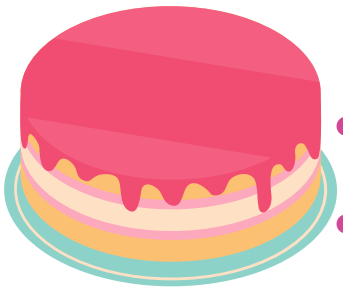
- Preheat the oven.
- Mix the ingredients.
- Pour the batter into a pan.
- Bake the cake.

If you follow these steps in order, you'll get a cake.

But what if you mixed up the order like this?

- Pour the batter into a pan.
- Bake the cake.
- Preheat the oven.
- Mix the ingredients.

This doesn't make sense, right?
You'd end up baking an empty pan and trying to mix ingredients after they've already been "baked."



```
# Wrong Order

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

In this example, the program tries to print the message before it's created. Because of this, it will give you an error.

```
# Correct Order

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

Now, the message is created first, and then the program prints it, so everything works as expected.

Always think about the order of your instructions in a program.
Just like following the right steps in a recipe, the order of your code is crucial to getting the correct outcome.

LOGIC & ALGORITHMS

So, what do we mean by logic?
If bits and bytes are building blocks, **logic can represent how we chose to arrange these blocks.** It serves as the concepts that decide how to organize our code, and is is essential for organizing and structuring code effectively. **Firstly, logic determines the sequence in which statements and commands** are written within your code.

Logic also controls execution based on conditions and decisions, and enables your program to **make decisions by evaluating conditions and choosing different paths or actions based on those conditions.**

For example, you might want your program to perform a specific action only if a condition is true. **This is one of many conditional statements in programming, called an IF - THEN statement.**

IF - THEN

In programming, an **if-then statement is a way to make a decision based on a certain condition**. It tells the computer: "If something is true, then do something; otherwise, don't do anything."
An if-then statement is a basic logical structure that presents conditions and their resulting actions.

To illustrate, consider common scenarios like pedestrians following traffic lights. If the light is green, we walk, but otherwise, we wait.
If we **break down the decisions that go into walking across a street** using programming terms, **we get an if-then statement.**

This captures a logical sequence – **if a specific condition is met (the light is green)**, then a particular **action should be taken (pedestrians can cross).**

IF-THEN IN ACTION

Let's zoom into the scenario of crossing a street to illustrate this a bit more - and lets see how code handles this conditional logic.

PYTHON

```python
if traffic_light_is_green:
    cross_street()

else:
    wait_at_the_corner()
```

if traffic_light_is_green:
checks whether the condition "traffic_light_is_green" is true.

If the condition is true
(meaning the traffic light is green), the **program will execute the cross_street() action.**

If the condition is not true
(meaning the traffic light is not green),
**the program will skip over the cross_street() action,
and execute the wait_at_the_corner() action instead .**

JAVASCRIPT

```javascript
if (trafficLightIsGreen) {
    crossStreet();

} else {
    waitAtTheCorner();
}
```

if (trafficLightIsGreen)
Checks whether the condition trafficLightIsGreen is true.

If the condition is true
(meaning the traffic light is green), the **program will execute the crossStreet() function.**

If the condition is not true
(meaning the traffic light is not green),
**the program will skip over the crossStreet() function,
and execute the waitAtTheCorner() function instead.**

This is similar to how we make decisions in real life.

If the traffic light is green, we cross the street. If not, we wait.

In programming, we use "if-then" statements to replicate this decision-making process in a logical way that the computer can understand.

This is an algorithm - systematic, step-by-step instructions engineered to solve problems or attain objectives.

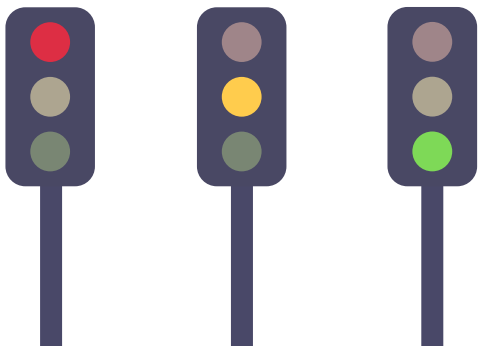
Consider an algorithm for efficient grocery shopping.
This involves a meticulous sequence of steps: compiling a shopping list, navigating the store, handpicking items, and proceeding to checkout.

Algorithms are engineered to streamline processes elegantly, akin to **guiding someone to the nearest store through a well-structured set of directions.**

The purpose of an algorithm is to solve a problem as quickly and simply as possible, representing the most straightforward and efficient way to provide instructions. **Note that since computers are binary we must formulate our conditions in a binary way** - there is only true or false in programming!

In its essence, logic means sketching out a **sequence of actions into its smallest components,** crafting efficient pathways that deftly handle tasks or tackle challenges in the most optimal way.

How would you instruct someone to reach the nearest bus stop?
Would you TELL them to put shoes on, or assume they would think of that themselves?



COMPARISON OPERATORS > AND <

In programming, comparison operators are used to compare two values. The two most common comparison operators are > and <

The main purpose of these operators is to compare values. They help determine the relationship between two numbers (or other comparable values). When you use these operators, the **result is always a boolean value: true or false.** If the comparison is correct (e.g., 7 > 3), the result is true.If the comparison is incorrect (e.g., 3 > 7), the result is false.

Greater Than >

- **This operator checks if the value on the left side is greater than the value on the right side.**
- **Example: 5 > 3 would return true because 5 is greater than 3.**

Less Than <

- **This operator checks if the value on the left side is less than the value on the right side.**
- **Example: 5 < 3 would return false because 5 is greater than 3.**

```
let a = 10;
let b = 20;
console.log(a > b);
// This will print: false, because 10 is not greater than 20

console.log(a < b);
// This will print: true, because 10 is less than 20
```

These operators are **often used in if statements** and other control structures where you need to make **decisions based on the comparison of values.**

For example, you might use > or < to check if a user's score is higher than a certain threshold, or to compare the sizes of two objects.

INDENTATION

When you're writing code, the structure of your program is crucial. **Different programming languages use different methods to organize and define blocks of code.**

JavaScript uses {} to group code, and indentation is just for readability. In JavaScript, blocks of code are grouped together using curly braces {}. For example, when you write an if statement, the code that should run if the condition is true is enclosed within {}.

Although **not required by the language, indentation (spacing) is typically used to make the code more readable.** Most developers indent the code inside the curly braces so that it's easy to see which code belongs to which block.

Python is different from JavaScript. **Python uses indentation to define code blocks, and no curly braces are needed.**

It doesn't use curly braces {} to define blocks of code - instead, Python relies on indentation. **This means that the spaces or tabs at the beginning of a line are crucial to the structure of the program.** For example, in an if statement, the code that should run if the condition is true must be indented (usually by four spaces). Since Python uses indentation to group code, there's no need for {}.

PYTHON

```
x = 10
y = 5
if x > y:
    print("x is greater than y")
```

JAVASCRIPT

```
let x = 10;
let y = 5;
if (x > y) {
    console.log("x is greater than y");
}
```

COMPUTERS AREN'T ACTUALLY SMART, JUST FAST

Computers are INCREDIBLY fast. A modern-day computer processor, operating at gigahertz (GHz) speed, can perform billions of calculations per second.

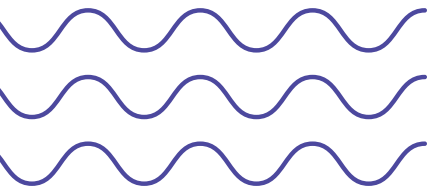
But computers lack intelligence. If you string all the tasks computers do together they give the illusion of a supersmart piece of technology.

In reality, everything a computer does is built of millions and millions of **VERY small calculations, done in fast succession.**

What is worse, they do exactly what you tell them to do.

The challenge in coding is not in matching the computer's intelligence - but in **communicating instructions at the computer's basic, literal level.** In essence, the programmer needs to get down to the "dumb" level of the computer.

While binary logic provides a basic foundation for decision-making, in programming, it often struggles with the complexity of real-world problems, which cannot be easily reduced to simple yes/no questions. Real-world scenarios often involve multiple factors, nuances, and exceptions that require more sophisticated decision-making than a binary approach intuitively provides.



For example, instructing someone to reach the nearest hospital involves factors like current location, traffic conditions, and available routes. A binary decision-making approach may oversimplify these complexities.

The challenge lies in expressing what you want the computer to do in such a precise, concrete, and incremental way that leaves no room for the computer to misinterpret your intent.

Crafting effective algorithms requires leveraging basic binary logic, while incorporating more nuanced ways to express instructions that reflect the complexity of tasks the programmer wants to accomplish. This ensures that programs can accommodate a variety of scenarios and make accurate decisions.

CODING - EXAMPLES & HOW TO

DATATYPES TABLE

Data Type	JavaScript	C#	Python	Java
Integer	number	int	int	int
Floating Point	number	float or double	float or double	float or double
String	string	string	str	String

INTEGER
Represents whole numbers.
let my_whole_number = 1;

FLOATING POINT
Represents numbers with decimal points.
let my_decimal_number = 1.5;

STRING
Represents sequences of characters
let my_text = “Hello!”;

BOOLEAN
Can only hold one of two values - true or false.
let true_or_false = true;

CODING - EXAMPLES & HOW TO

PYTHON

```python
age = None     # Declare the variable 'age' without assigning a specific value

# Variable Declaration and Initialization
age = 20  # Here, 'age' is initialized with the value 20.

# Variable Reassignment
age = 25  # The value of 'age' is updated to 25.

# Print the value
print(age)  # Output: 25
```

```python
# Variable Declaration
traffic_light_is_green = True  # or False, depending on the scenario

# Conditional Statement
if traffic_light_is_green:
    print("Cross the street.")  # Executes if the light is green
else:
    print("Wait at the corner.")  # Executes if the light is not green
```

```python
x = 10
y = 5

if x > y:
    print("x is greater than y")
```

CODING - EXAMPLES & HOW TO

JS JAVASCRIPT

```javascript
let age;        // Declare the variable 'age' without assigning a value
age = 10;       // Assign the value 10 to 'age'
age = 11;       // Reassign 'age' to a new value 11
console.log(age);  // Output: 11

// Variable Declaration and Initialization
let age = 20;  // Here, 'age' is initialized with the value 20.
```

```javascript
// Variable Declaration
let trafficLightIsGreen = true;  // or false, depending on the scenario

// Conditional Statement
if (trafficLightIsGreen) {
    console.log("Cross the street.");  // Executes if the light is green
} else {
    console.log("Wait at the corner.");  // Executes if the light is not green
}
```

```javascript
let x = 10;
let y = 5;

if (x > y) {
    console.log("x is greater than y");
}
```

CODING - EXAMPLES & HOW TO

PRINT() / CONSOLE LOG();

PYTHON

```python
print("Hello, World!")  # Output: Hello, World!
```

The print function outputs text or data to the standard output, typically the terminal or console.

Syntax: print(value), where value can be a string, number, or any other data type.

JAVASCRIPT

```javascript
console.log("Hello, World!");  // Output: Hello, World!
```

The console.log method outputs text or data to the web browser's console or a JavaScript runtime environment like Node.js.

Syntax: console.log(value), where value can be a string, number, object, or any other data type.

CODING - EXAMPLES & HOW TO

//COMMENTS

Comments are lines in the code that are ignored by the interpreter or compiler, allowing developers to leave notes or explanations within the code. They are used to improve readability and provide context or instructions without affecting the execution of the program.

In both examples, **the text following # in Python or // in JavaScript is ignored by the program** and serves only as a note for anyone reading the code.

python

```python
# This is a single-line comment in Python
x = 10  # This comment explains that x is being assigned the value 10
```

javascript

```javascript
// This is a single-line comment in JavaScript
let x = 10;  // This comment explains that x is being assigned the value 10
```