

Full Stack Web Development

Intro to programming, variables and data types

- Introduction to Programming
- Introduction to Algorithm
- Introduction to JavaScript
- Variable
- Data Types
- Type Conversion
- Operator

Introduction to Programming

- **What** is programming ?
- **Why** learn programming ?
- **What** is programming language ?



What is Programming ?

Programming is **the process of creating a set of instructions that tell computer to perform a task.**



Why Learn Programming ?

- Improve problem solving & logical thinking
- Grow your creativity
- Level-up your career
- Great earning potential
- Technology are ruling the world

“Everybody in this country should learn how to program a computer ... because it teaches you how to think”

–Steve Jobs–



What is Programming Language ?

A programming language is a **vocabulary** and set of **grammatical** rules for instructing a computer or computing device to perform specific tasks.

Example :

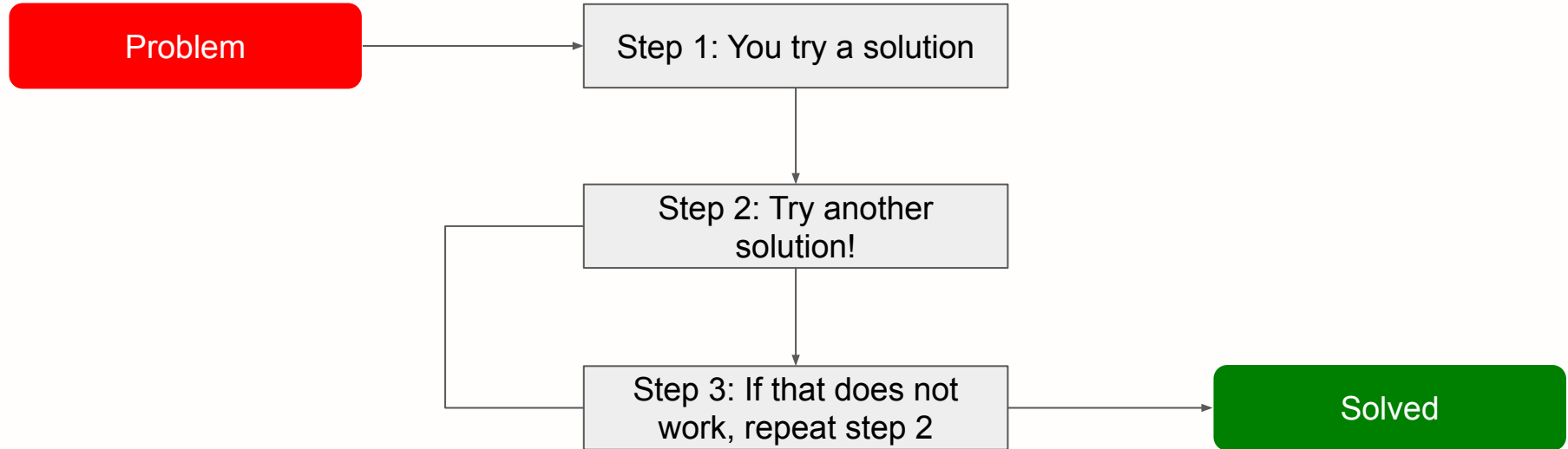
Javascript, Java, Golang, PHP, C, C++, C#, etc.



Before we jump in to the Algorithm, let's talk about problem solving first.

Problem-solving skills are the ability to identify problems, brainstorm and analyze answers, and implement the best solutions. Programmers with good problem-solving skills is both a self-starter and a collaborative teammate; they are proactive in understanding the root of a problem and work with others to consider a wide range of solutions before deciding how to move forward.

Think Like a Programmer



- What Is An Algorithm?
 - An algorithm is a set of step-by-step procedures, or a set of rules to follow, for completing a specific task or solving a particular problem. Algorithms are all around us.
- Why are Algorithms Important to Understand?
 - Algorithmic thinking, or the **ability to define clear steps to solve a problem**, is crucial in many different fields. Even if we're not conscious of it, we use algorithms and algorithmic thinking all the time. Algorithmic thinking allows you to break down problems and conceptualize solutions in terms of discrete steps. Being able to understand and implement an algorithm requires you to **practice structured thinking and reasoning abilities**.

- Where are Algorithms Used as a Web Developer?

Algorithms are used in every part of IT industries. They form the field's backbone. Algorithm gives the computer a specific set of instructions, which allows the computer to do everything. **Algorithms written in programming languages that the computer can understand.**

Computer algorithms play a big role in how social media works: which posts show up, which ads are seen, and so on. These decisions are all made by algorithms.

Google's programmers use algorithms to optimize searches, predict what users are going to type, and more. **In problem-solving, a big part of computer programming is knowing how to formulate an algorithm.**

Introduction to JavaScript

- **What** is JavaScript ?
- **Why** use JavaScript ?
- Setting up development environment
- Let's write our first code !
- JavaScript **code structure**



What is JavaScript ?

JavaScript is a programming language. It is lightweight and **most commonly used as a part of web pages**. It is an interpreted programming language with object-oriented capabilities.

JavaScript can execute **not only in the browser, but also on the server, or actually on any device** that has a special program called the JavaScript engine.

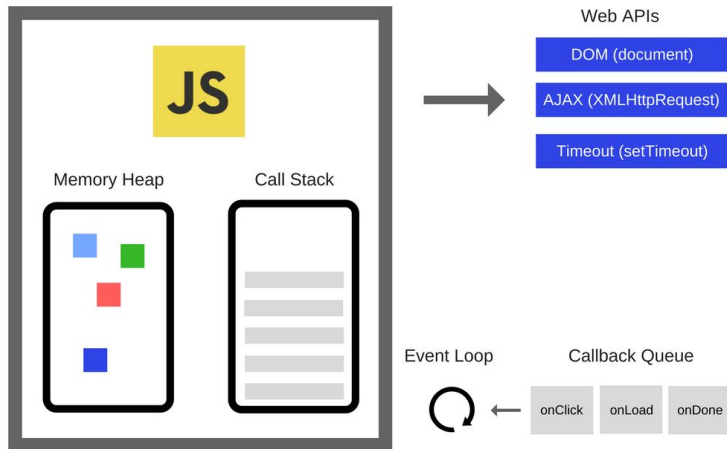


What is JavaScript ?

Javascript is **single-threaded, non-blocking, asynchronous, concurrent** language.

- **Single-threaded** means that it runs only one thing at a time.
- **Non-blocking & Asynchronous** means that it doesn't wait for the response of an API call, I/O events, etc., and can continue the code execution.
- **Concurrent** means executing multiple tasks at the same time but not simultaneously. E.g. two tasks works in overlapping time periods.

For more information, watch this [video](#)



Why Use JavaScript ?

-
- Easy to Learn
 - Popularity
 - Large Community
 - Speed
 - Versatility
 - Interoperability


Setting Up the Development Environment

Install the following tools & extensions :

- [Visual Studio Code](#)
 - IDE / Code editor
- [QuokkaJS](#)
 - VS Code Extension, to run your code with instant result / feedback
- [Git](#)
 - Source code management

Hello World !

Let's write our first code !

A stylized illustration of a code editor window. It has a light blue outer frame and a dark gray inner area. At the top left of the inner area are three colored circles (red, yellow, green) representing window control buttons. The text "console.log('Hello World');" is displayed in a light blue monospace font.

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


Javascript Code Structure

Single Statement



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

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains the JavaScript code `console.log("Hello World");`. A green arrow points from the text "Single Statement" above to the code. A green circle highlights the semicolon at the end of the line, with a green arrow pointing from it to the text "Semicolon" on the right.

Semicolon



```
// this is comment  
/* this is  
   multi-line comment */  
console.log("Hello World");
```

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains the following JavaScript code:
`// this is comment`
`/* this is`
 `multi-line comment */`
`console.log("Hello World");`

Variable

Variable is a “named storage” for data. We can use variable **to store any data** you need.



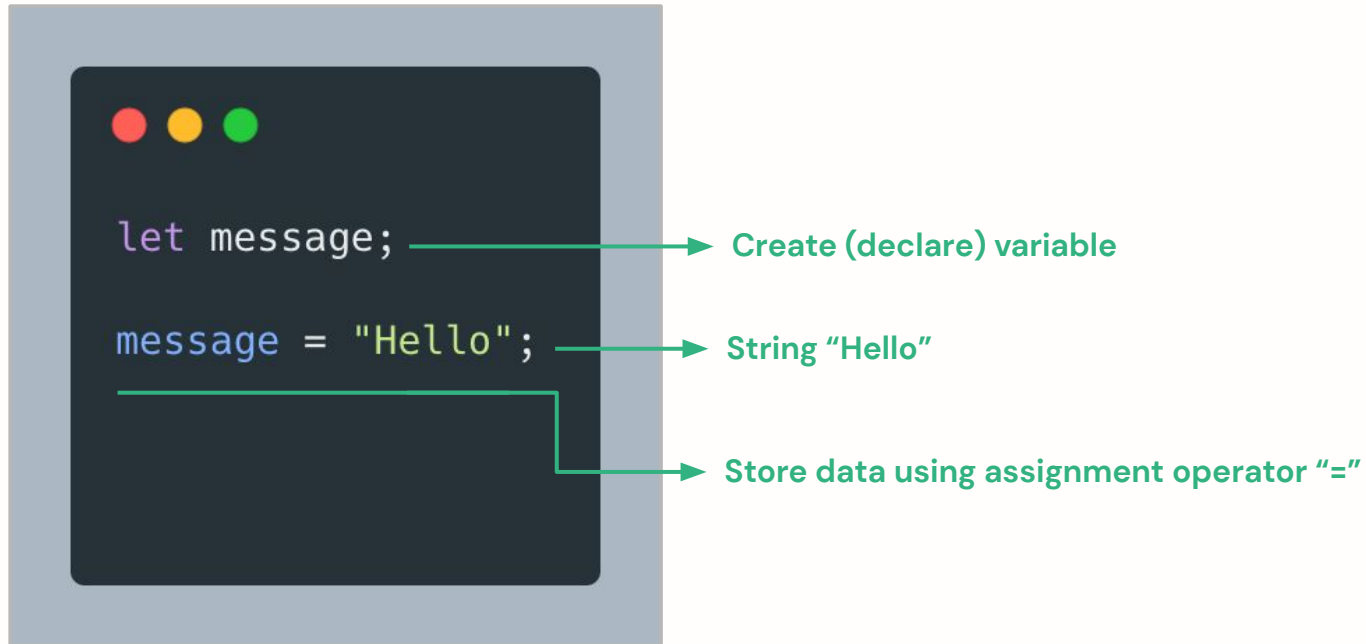
Example

In package delivery apps, there's information about package details, address, sender's name, etc.

Variable are used to store all the information.




Code Example



Variable Declaration

Different ways to declare variable :

- **var** : To create global variables
- **let** : To create scoped, replaceable variables
- **const** : Can't be updated or redeclared within the scope



```
var globalVariable = "Hello World!";

let replaceable = "This value will be replaced";
replaceable = "I love JavaScript";

const constant = "Hello Purwadhika!";

console.log(globalVariable);
console.log(replaceable);
console.log(constant);
```

Variable Naming

-
- Must contain only letters, digits, or the symbols "\$" and "_"
 - The first character must not digit
 - Case-sensitive
 - Can't use reserved words

A value in JavaScript is always of a certain **type**.

Primitive data types : The predefined data types provided by JavaScript.

Non-primitive data types : The data types that are derived from primitive data types.

Primitive	
String	Used to represent textual data
Number & BigInt	Used to hold decimal values as well as values without decimals
Boolean	Represents a logical entity and can have two values: true and false
Null	Has exactly one value: null . Represents the intentional absence of any object value
Undefined	A variable that has not been assigned a value has the value undefined

Non Primitive	
Object	Is an entity having properties and methods (keyed collection) → Will be explained in the next session
Array	Used to store more than one element under a single variable → Will be explained in the next session

Data Types



```
const message = "Javascript";           //string
const count = 1;                        //number
const bigNumber = 9007199254740991n;    //bigint
const isTrue = true;                   //boolean
const noData = null;                   //object
let noAssigned;                         //undefined
const person = {                       //object
  name: "joni",
  age: 26
}

console.log(typeof message);
console.log(typeof count);
console.log(typeof bigNumber);
console.log(typeof isTrue);
console.log(typeof noData);
console.log(typeof person);
```


Mutable vs Immutable

Primitive	
String	Used to represent textual data
Number & BigInt	Used to hold decimal values as well as values without decimals
Boolean	Represents a logical entity and can have two values: true and false
Null	Has exactly one value: null . Represents the intentional absence of any object value
Undefined	A variable that has not been assigned a value has the value undefined

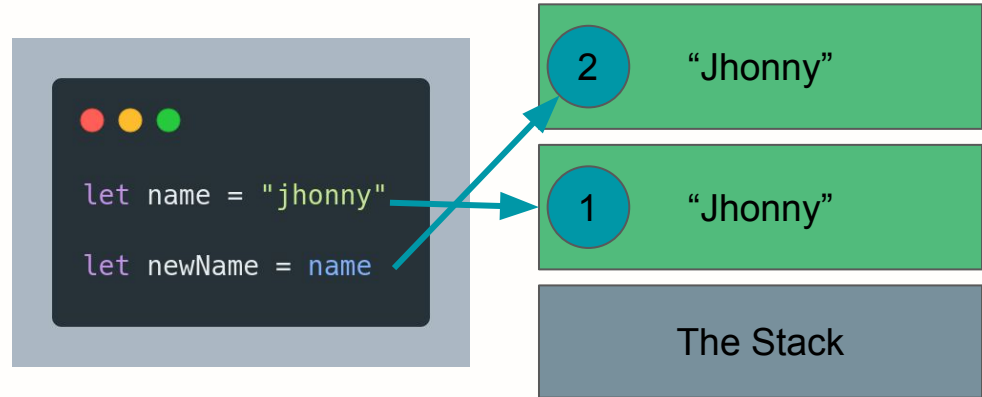
Non Primitive	
Object	Is an entity having properties and methods (keyed collection) → Will be explained in the next session
Array	Used to store more than one element under a single variable → Will be explained in the next session

- **Mutable** is a type of variable that can be changed. (contains of: non primitive) it is also called as **reference type**
- **Immutable** are the objects whose state cannot be changed once the objects is created. (contains of: primitive) immutable it is also called as **value type**.
- Declaring variable with **const** doesn't make the value immutable. **It depends on data type.**

Mutable vs Immutable

Value types are been stored on the Stack in our memory.

When storing a value type in memory, it adds an element to the top of the stack with the value of the newly created variable. When creating a new variable and assigned the first one to the new one, it adds a new element on top of the stack with the value of the new variable.

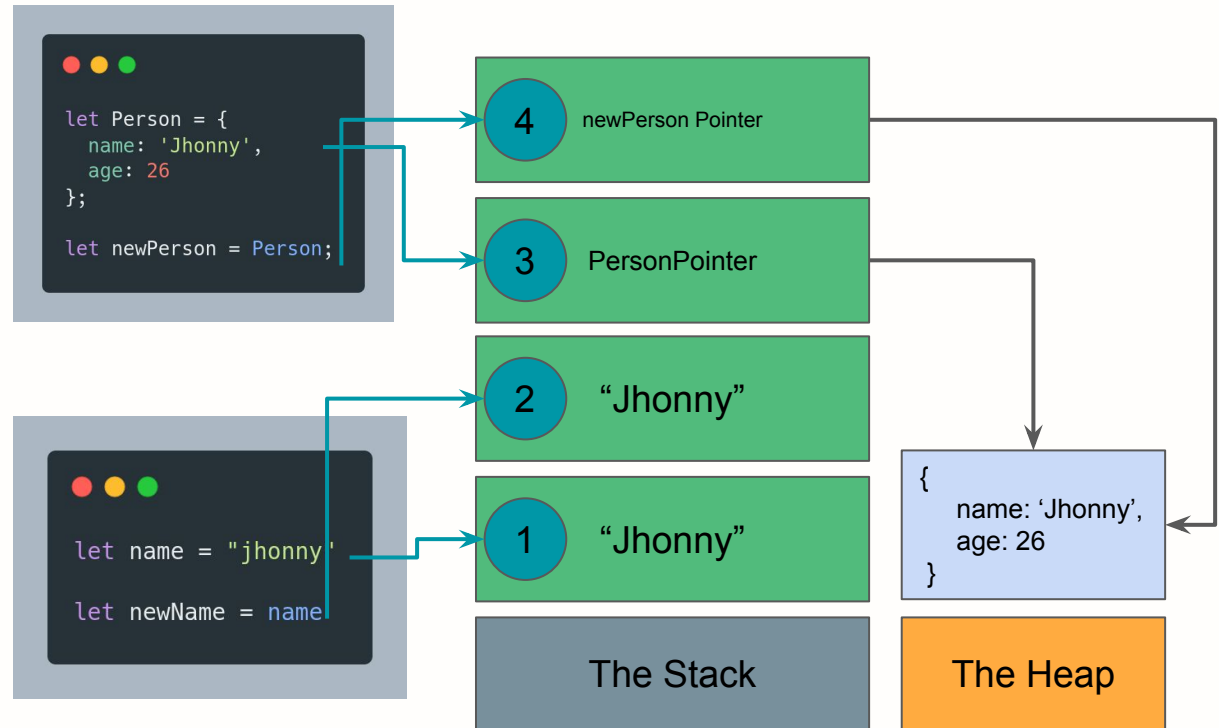


The first variable — **name** gets into the stack with the value of the variable data. Then, the **newName** gets into the stack in a new memory location with the value of the variable data.

Mutable vs Immutable

Reference types are been stored on the Heap. The Heap, indifference from the stack, has no order of how to store the data.

When storing a reference type in memory, it adds a new element to the top of the stack, when its value is a pointer/reference to the address of the object that has been stored on the heap.

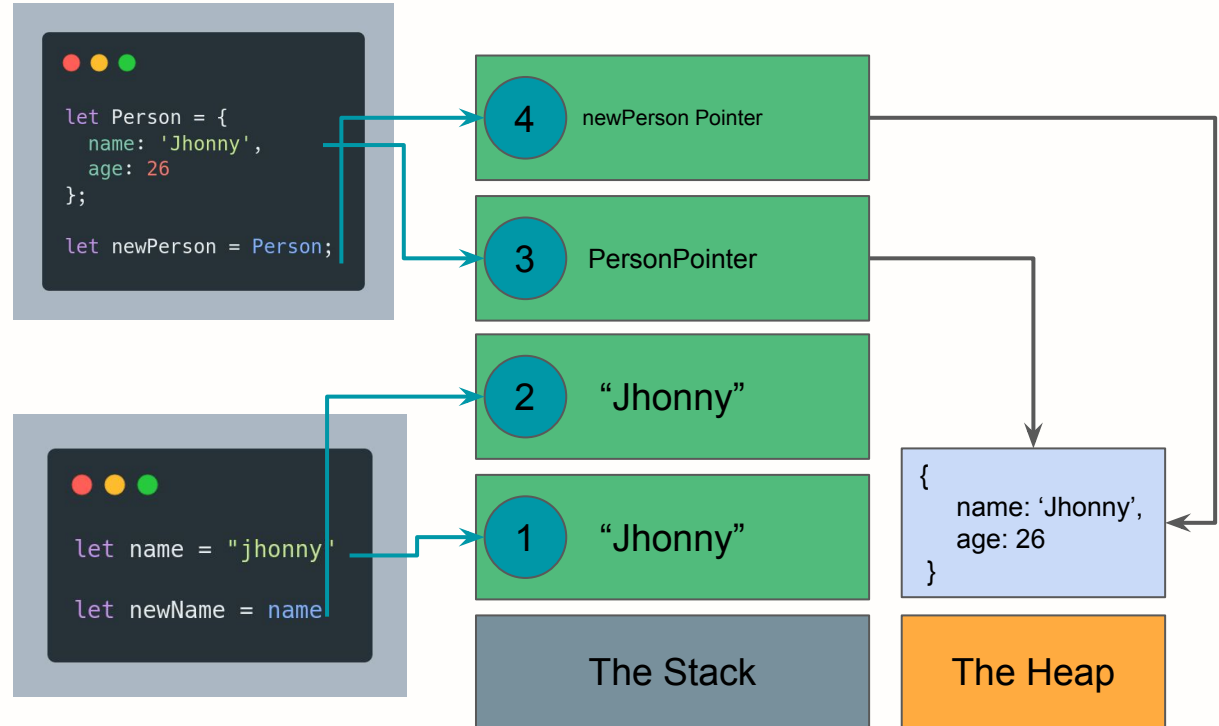


Mutable vs Immutable

Immutable

Try to change value in **newName** variable, check out the result!

Does that change **name** variable value?

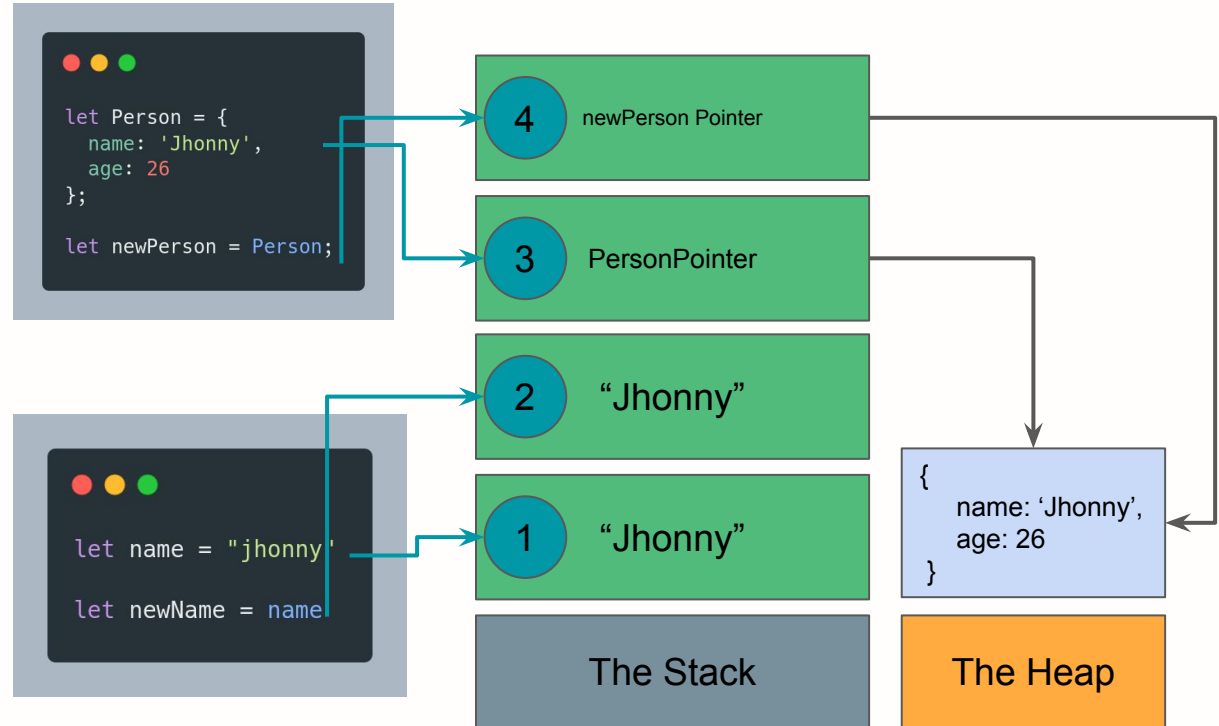


Mutable vs Immutable

Mutable

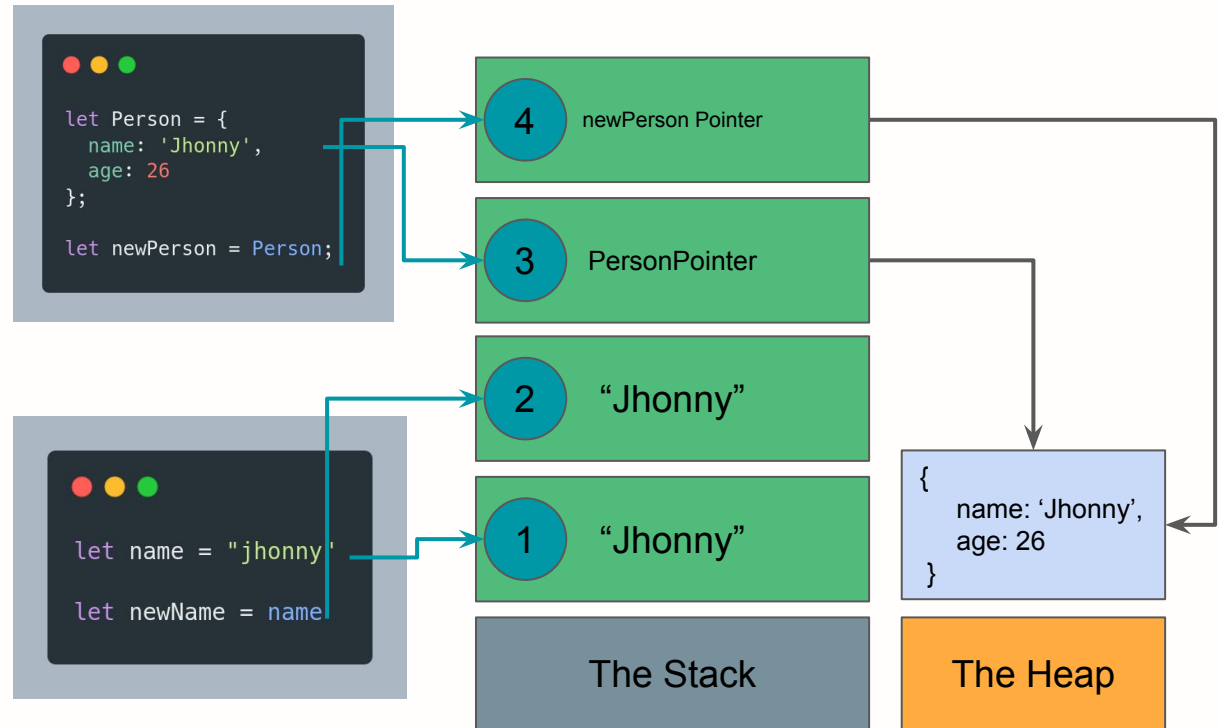
Try to change value in **newPerson** variable, check out the result!

Does that change **Person** variable value?



Mutable vs Immutable

With that in mind, we can say that a **value type** is **immutable** where a **reference type** is **mutable**.



String Built-in Method

-
- slice
 - substring
 - substr
 - replace
 - toUpperCase
 - toLowerCase
 - concat
 - trim
 - padStart
 - padEnd
 - charAt
 - charCodeAt
 - split
 - indexOf
 - lastIndexOf
 - search

Template Literals

- **Template literals** (template strings) allow you to use strings or embedded expressions in the form of a string.
- Template literals are enclosed by backtick (`) characters instead of double or single quotes.
- With template literals, you can get :
 - A **multiline string** → a string that can span multiple lines.
 - **String formatting** → the ability to substitute part of the string for the values of variables or expressions. This feature is also called **string interpolation**.
 - **HTML escaping** → the ability to transform a string so that it's safe to include in HTML.



```
// String interpolation
const name = "David";
const message = `Welcome, ${name}`;
console.log(message);
```


Number built-in method

- toString
- toExponential
- toFixed
- toPrecision
- valueOf

Global built-in method & property

- Number
- parseInt
- parseFloat

- MAX_VALUE
- MIN_VALUE
- POSITIVE_INFINITY
- NEGATIVE_INFINITY
- NaN

- String Conversion

- `String(123)` `// return a string from a number literal 123`

- Numeric Conversion

- `const num = "3" * "3"` `// return 9 in number`

- `Number("3.14")` `// return 3.14 in number`

- Boolean Conversion

- `Boolean(1)` `// return true`

- `Boolean(0)` `// return false`

- `Boolean("Hello")` `// return true`

- `Boolean("")` `// return false`

Date Data Type

It stores the date, time and provides methods for date/time management.

```
// shows current date/time
let now = new Date();

// 0 means 01.01.1970 UTC+0
let Jan01_1970 = new Date(0);

// now add 24 hours, get 02.01.1970 UTC+0
let Jan02_1970 = new Date(24 * 3600 * 1000);

// The time is not set, so it's assumed to be midnight GMT and
// is adjusted according to the timezone the code is run in
let date = new Date("2017-01-26");
```

Get Methods

- `getFullYear`
- `getMonth`
- `getDate`
- `getHours`
- `getMinutes`
- `getSeconds`
- `getMilliseconds`
- `getTime`
- `getDay`
- `Date.now`
- `Date.parse`

Set Methods

- `setDate`
- `setFullYear`
- `setHours`
- `setMilliseconds`
- `setMinutes`
- `setMonth`
- `setSeconds`
- `setTime`

Basic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (modulo)
**	Exponentiation

Unary, Binary and Operand

- An **operand** is what operators are applied to. For instance, in the multiplication of $5 * 2$ there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.
- An **operator** is **unary** if it has a single operand. For example, the unary negation `-` reverses the sign of a number.
- An **operator** is **binary** if it has two operands. The same minus exists in binary form as well.

```
let x = 1;  
x = -x;  
console.log(x);  
// -1, unary negation was applied
```

```
let x = 1, y = 3;  
console.log(y - x);  
// 2, binary minus subtracts values
```

String Concatenation with binary "+"

```
let s = "Hello" + "World";  
console.log(s); // HelloWorld  
  
console.log("1" + 2); // "12"  
console.log(2 + "1"); // "21"  
console.log(1 + 1 + "1"); // "21" not "111"  
console.log("1" + 1 + 1); // "111" not "12"
```

Note : Only work with binary "+". Other arithmetic operators work only with numbers and always convert their operands to numbers.

Modify in Place

We often need to apply an operator to a variable and store the new result in that same variable.



```
let n = 2;  
n += 5; // now n = 7 (same as n = n + 5)  
n *= 2; // now n = 14 (same as n = n * 2)  
  
console.log(n); // 14
```


Increment & Decrement

- Increasing or decreasing a number by one is among the most common numerical operations.
- Increment **++** increases a variable by 1.
- Decrement **--** decreases a variable by 1



```
let counter = 2;  
counter++; // 3  
counter--; // 2
```

Postfix & Prefix Form

- The operators `++` and `--` can be placed either before or after a variable.
- When the operator goes after the variable, it is in “**postfix form**”: `counter++`.
- The “**prefix form**” is when the operator goes before the variable: `++counter`.
- If we’d like to increase a value and immediately use the result of the operator, we need the prefix form
- If we’d like to increment a value but use its previous value, we need the postfix form



```
let preCounter = 0;
console.log(++preCounter); // 1

let postCounter = 0;
console.log(postCounter++); // 0
```

Comparisons

Operator	Description
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than equal
<=	Less than equal
===	Strict Equality

Introduction to Pseudocode

Before we start our exercise on the next slides, let's talk about Pseudocode first.

Pseudocode is basically an "easier-to-understand" version of programming languages, usually in the form of simple natural language.

Through pseudocode, you can express detailed step-by-step process in a much simpler language.

Problem:

Write a code to find area of rectangle!

Hint:

1. Find out how to count area of rectangle
2. $\text{length} \times \text{width} = \text{area of rectangle (the formula)}$

Solutions in Pseudocode:

1. define variable and assign value to variable

```
const rectangleLength = 10
const rectangleWidth = 5
```
2. define variable to keep the result and implement the formula area of rectangle

```
const rectangleArea = rectangleLength * rectangleWidth
```

Exercise

Since you already know about pseudocode, let's solve this exercise through pseudocode first, and then convert it into a programming code!

- Write a code to find area of rectangle.
- Write a code to find perimeter of rectangle.
- Write a code to find diameter, circumference and area of a circle.
- Write a code to find angles of triangle if two angles are given.
- Write a code to get difference between dates in days.
- Write a code to convert days to years, months and days.
 - Example : 400 days → 1 year, 1 month, 5 days
 - 1 year : 365 days, 1 month : 30 days

Thank You!

