# INTRODUCTION TO JS

ONYENSO GOODLUCK C.

# OUTLINES

- What is JavaScript
- Ways to Run JavaScript
-

# Introduction

## WHAT IS JAVASCRIPT?

JavaScript was initially created to “make web pages alive”.

The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.

# How To Run JavaScipt

1. Browser Console
   - Open your browser
   - Right Click and select INSPECT
   - Open the Developer Consle
2. Node.js (Backend)
   - Using your terminal launch the file
   - on the directory which was created
   - type Node ‘filename’
3. Script (Using Browser)
   - Using your html flle, link your JS
   - <script src =’jsfilename’></script>

# What makes JavaScript unique?

There are at least three great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things. That’s what makes JavaScript unique. That’s why it’s the most widespread tool for creating browser interfaces.

That said, JavaScript can be used to create servers, mobile applications, etc.

# CODE STRUCTURE

- **Statements**

Statements are syntax constructs and commands that perform actions.

We've already seen a statement, **`console.log('Hello, world!')`**, which shows the message "Hello, world!".

We can have as many statements in our code as we want. Statements can be separated with a semicolon.

For example, here we split "Hello World" into two console.logs:

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

Usually, statements are written on separate lines to make the code more readable

A semicolon may be omitted in most cases when a line break exists.

This would also work:

```
console.log('Hello')
console.log('World')
```

Here, JavaScript interprets the line break as an "implicit" semicolon. This is called an automatic semicolon insertion.

# Comments

As time goes on, programs become more and more complex. It becomes necessary to add comments which describe what the code does and why.
Comments can be put into any place of a script. They don’t affect its execution because the engine simply ignores them.
**One-line comments** start with two forward slash characters **//**.
The rest of the line is a comment. It may occupy a full line of its own or follow a statement.
Like here:

```
// This comment occupies a line of its own
console.log('Hello');console.log('World'); // This comment follows the statement
```

**Multiline comments** start with a forward slash and an asterisk **/*** and end with an asterisk and a forward slash ***/.**
Like this:

```
/* An example with two messages.
This is a multiline comment.
*/
console.log('Hello');console.log('World');
```

The content of comments is ignored, so if we put code inside **/* ... */**, it won't execute.
Sometimes it can be handy to temporarily disable a part of code:

```
/* Commenting out the code
console.log('Hello');
*/
console.log('World');
```

Please, don't hesitate to comment your code.Comments increase the overall code footprint,

# Variables

A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the:

- var
- let
- const

The **var** keyword is almost the same as **let**. It also declares a variable but in a slightly different, “old-school” way.

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

# Variable naming

There are two limitations on variable names in JavaScript:

- The name must contain only letters, digits, or the symbols $ and _.
- The first character must not be a digit.
-

Examples of valid names:

- **let userName;**
- **let test123;**

When the name contains multiple words, **camelCase** is commonly used.

That is: words go one after another, each word except first starting with a capital letter: **myVeryLongName.**

# Constants

To declare a constant (unchanging) variable, use const instead of let:

**const myBirthday = '28.08.1993';**

Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

**const myBirthday = '28.08.193';**
**myBirthday = '01.01.2001'; // error, can't reassign the constant!**

# Proper Way To Name Variables

Talking about variables, there’s one more extremely important thing.
A variable name should have a clean, obvious meaning, describing the data that it stores.
Variable naming is one of the most important and complex skills in programming. A glance at variable names can reveal which code was written by a beginner versus an experienced developer.

# Some good-to-follow rules are:

- Use human-readable names like userName or shoppingCart.
- Stay away from abbreviations or short names like a, b, and c, unless you know what you’re doing.
- Make names maximally descriptive and concise. Examples of bad names are data and value. Such names say nothing. It’s only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
- Agree on terms within your team and in your mind. If a site visitor is called a “user” then we should name related variables currentUser or newUser instead of currentVisitor or newManInTown.

# Tasks

- Declare two variables: admin and name.
- Assign the value "John" to name.
- Copy the value from name to admin.
- Show the value of admin using alert (must output “John”).

**solution**

```
let admin;
let name;
name = "John";
admin = name;
console.log( admin ); // "John"
```

# Tasks

- Declare two variables: admin and name.
- Assign the value "John" to name.
- Copy the value from name to admin.
- Show the value of admin using alert (must output “John”).

**solution**

```
let admin;
let name;
name = "John";
admin = name;
console.log( admin ); // "John"
```

# Data types

There are eight basic data types in JavaScript

- Number

The number type represents both integer and floating point numbers.

**let n = 123;**

**n = 12.345;**

Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: I**nfinity, -Infinity and NaN.**

- Infinity represents the mathematical Infinity ∞. It is a special value that’s greater than any number.
- We can get it as a result of division by zero:

console.log( 1 / 0 ); // Infinity
Or just reference it directly:
console.log Infinity ); // Infinity

- **String**

A string in JavaScript must be surrounded by quotes.
let str = "Hello";
let str2 = 'Single quotes are ok too';
In JavaScript, there are 3 types of quotes.
**Double quotes: "Hello".**
**Single quotes: 'Hello'.**
**Backticks: `Hello`.**

Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.
Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:

```
let name = "John";
console.log( `Hello, ${name}!` );
```

- **Boolean (logical type)**

The boolean type has only two values: true and false.
This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.
For instance:

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

Boolean values also come as a result of comparisons:

```
let isGreater = 4 > 1;
console.log( isGreater ); // true (the comparison result is "yes")
```

- **The “null” value**

The special null value does not belong to any of the types described above.
It forms a separate type of its own which contains only the null value:
let age = null;
In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.
It’s just a special value which represents “nothing”, “empty” or “value unknown”.
The code above states that age is unknown.

- **The “undefined” value**

The special value undefined also stands apart. It makes a type of its own, just like null.

The meaning of undefined is “value is not assigned”.

If a variable is declared, but not assigned, then its value is undefined:

```
let age;
console.log(age); // shows "undefined"
```

- **The typeof operator**

The typeof operator returns the type of the operand.
It’s useful when we want to process values of different types differently or just want to do a quick check.
A call to typeof x returns a string with the type name:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof true // "boolean"
typeof "foo" // "string"
typeof console.log// "function"  (3)
```

# THANK YOU