**Learning Objectives**

1. JavaScript Fundamentals
   * Variables & Data Types
   * Identify equality operators
   * Conditionals (Control Flows) and Operators.
   * Logical Operators

**📚 Readings**

[Summary of ES6 Features (Links to an external site.)](https://github.com/lukehoban/es6features#readme)

**Lecture Notes**

Node. js Read-Eval-Print-Loop (REPL) is **an easy-to-use command-line tool, used for processing Node.** **js expressions**. It captures the user's JavaScript code inputs, interprets, and evaluates the result of this code. It displays the result to the screen, and repeats the process till the user quits the shell.

**Variables**

**What is a Variable?** We use variables to store data temporarily.

**How do we declare Variables?** There are 3 variables identifiers we can use: var, let and const

Let’s quickly recap  var, let and const with respect to their scope and use, and hoisting. Take note of the differences between them that I'll point out. In JavaScript, we can declare variables in three different ways like this:

// Without keywords. It is essentially the same as var

// and not allowed in 'strict' mode.

name = 'Samora';

// Using var

var price = 1000;

// Using let

let hasGraduated = false;

// Using const

const PUBLICATION = 'freeCodeCamp'

**Const**: Variables declared with the const maintain constant values.

let personalId = 33009123; // Change to const

personalId = 5;

console.log(personalId);

Before the advent of ES6, vardeclarations ruled. There are issues associated with variables declared with var, though. That is why it was necessary for new ways to declare variables to emerge. The major issue being around Scope, which defines accessibility of Variables.

The scope is global when a var variable is declared outside a function. This means that any variable that is declared with var outside a function block is available for use in the whole window.

{

let f\_name = 'Alex';

const ZIP = 500067;

var age = 25;

}

console.log(f\_name); // Uncaught ReferenceError: f\_name is not defined

console.log(ZIP); // Uncaught ReferenceError: ZIP is not defined

console.log(age); // 25

var is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

var greetings = "Hi There. Welcome to Phase 1";

function newFunction() {

var hello = "hello";

}

Here, greetings is globally scoped because it exists outside a function while hello is function scoped. So we cannot access the variable hello outside of a function. So if we do this:

var greetings = "Hi There. Welcome to Phase 1";

function newFunction() {

var hello = "hello";

}

console.log(hello); // error: hello is not defined

Variables declared outside of any functions and blocks are global and are said to have Global Scope. This means you can access them from any part of the current JavaScript program.

let greetings = "Hi There. Welcome to Phase 1";

var hello = "hello";

const ZIP = 50080;

function newFunction() {

console.log(greetings);

console.log(hello);

console.log(ZIP);

}

newFunction();

**Data Types**

Strings: *Strings* are collections of alphanumeric characters and symbols.

Numbers: *Numbers* are just what they sound like. They’re numbers, including both integers and decimals. We will often use numbers to perform mathematical operations.

Booleans: *Booleans* can only have two values. True and false. They represent all data that only has two states, like a light switch. On or off.

Undefined: The *undefined* data type means that the variable has been created, but has never been given a value.

Null: *Null* is similar to undefined, except it has to be set intentionally. It also means empty or nothing, but it’s that way because a developer told it it was empty or nothing.

let f\_name = 'Samora Yommie'; // String

let age = 25; // Number

let graduated = false; // Boolean

let email; // undefined. Exist

console.log(l\_name); // Null doesn't exist. Not created / declared.

**Null and Undefined:** The difference is undefined exists when we haven’t given a value.

**Concatenation and Interpolation.** Concatenation is a way to take two strings and add one to the other, creating a single, longer string. String interpolation lets us dynamically insert values in the middle of a string. To do this, we need to use [template literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals), which are simply strings wrapped in backticks rather than single or double quotes.

Concatenation

// "High " + "five!";

const studentName = 'Samora Yommie';

const welcomeMessage = 'Welcome to Flatiron, ' + studentName;

Interpolation

const studentName = 'Samora Yommie';

const welcomeMessage = `Welcome to Flatiron, ${studentName}!`;

**Identify equality operators**

In this section we'll learn about comparison operators, which enable us to check to see if a value is what we're expecting.

JavaScript includes four equality operators. These operators allow us to compare values and determine whether they are the same.

* **strict equality operator** (===) - returns true if two values are equal *without performing type conversions*. Even if the values on both sides of the operator look similar (e.g., '42' === 42), the ===  operator will only return true  if the data types also match:

42 === 42;

// => true

42 === "42";

// => false

* **strict inequality operator** (!==) - returns true if two values are *not* equal *without* performing type conversions:

9000 !== 9001

// => true

9001 !== '9001'

// => true

9000 !== 9000

// => false

* **loose equality operator** (==) - returns true if two values are equal and not strict on the data types:

42 == 42;

// => true

42 == "42";

// => true

* **loose inequality operator** (!=) - opposite of ==. It returns true if two values are *not* equal, performing type conversions as necessary:

9000 != 9001

// => true

9001 != '9001'

// => false

**JavaScript Conditionals**

*JavaScript conditionals* are logical expressions that evaluate to either *true* or *false*. Conditionals are used to determine program flow through if statements and switch statements.

**if statements:** The if statement executes a block of code if the conditional expression inside a pair of parentheses evaluates to true. The conditions typically involve comparison of variables for equality or inequality. Here is a simple example:

let age = 32;

if (age >= 18) {

console.log ("Adult");

console.log ("Allowed to vote");

}

**if else statement:** The if-statement may contain an optional else clause that specifies an alternative course of action. The else clause is executed if the expression in the if-statement is evaluated to false.

let age = 32;

if (age >= 18) {

console.log ("Adult");

console.log ("Allowed to vote");

} else {

console.log("Infant");

console.log("Not allowed to vote");

}

**Multiple else if:** If you need to test multiple conditions then if…else if…else is what you can use. If the condition associated with the if statement is true, the statements inside the if block are executed. If the condition is false, each else-if statement is evaluated in the order in which they appear.

var age = 34;

if (age < 13) {

console.log("child");

} else if (age < 19) {

console.log("Teen");

} else {

console.log("adult");

}

**Alternative way to writing If Statements**

A ternary operator can be used to replace an if..else statement in certain situations.

condition ? expressionIfTrue : expressionIfFalse;

In this syntax, the condition is an expression that evaluates to a Boolean value, either true or false.

If the condition is true, the first expression (expresionIfTrue) executes. If it is false, the second expression (expressionIfFalse) executes.

We can replace this code

// check the age to determine the eligibility to vote

let age = 15;

let result;

if (age >= 18) {

result = "You are eligible to vote.";

} else {

result = "You are not eligible to vote yet.";

}

console.log(result);

With this:

// ternary operator to check the eligibility to vote

let age = 15;

let result = (age >= 18) ? "You are eligible to vote." : "You are not eligible to vote yet";

console.log(result);

**Switch**

As an alternative to a large number of if .. else if .. else statements you can use a switch statement which makes your code a bit more concise and readable.

The switch keyword is followed by an expression you wish to evaluate. This expression is referred to as the control variable that controls the flow of program execution. This expression is evaluated and its value is compared to the value following each of the case labels. Once the matching case label is found, the block of code contained inside that case is executed. The last statement in a block of code is almost always a break statement that signals the end of the block which is used to stop the switch statement from continuing to look at case statements once it finds a match.

let learning\_model = 'Remote';

switch (learning\_model) {

case 'Remote':

console.log("Enrolled In Online Classes.");

break;

case 'Hybrid':

console.log("Enrolled on In Person Classes.");

break;

default:

console.log("Student not Found in Records.");

break;

}

**Logical Operators**

We use Logical Operators to negate and combine expressions.

* **Logical AND**: true if both the operands/boolean values are true, else evaluates to false
* **Logical OR**: true if either of the operands/boolean values is true. evaluates to false if both are false
* **Logical NOT**: true if the operand is false and vice-versa.

const a = true, b = false;

// logical AND

console.log(a && a); // true

console.log(a && b); // false

// logical OR

console.log(a || b); // true

console.log(b || b); // false

// logical NOT

console.log(!a); // false

console.log(!b); // true