PRIME INTUIT
Finishing School

JavaScript Level 1

PRIME INTUIT
Finishing School

Introduction

Introduction

- ✓ JavaScript is used to add interactive functionality to our websites / Web applications!
- ✓ JavaScript support is built directly into modern web browsers
- ✓ We can run JavaScript directly into the browser console or as a full .js script connected to an HTML file

PRIME INTUIT
Finishing School

Introduction

Introduction

- ✓ JavaScript is a full programming language, meaning unlike HTML and CSS, JS supports things such as arrays, loops and general logic.
- ✓ JavaScript support is built directly into modern web browsers
- ✓ We can run JavaScript directly into the browser console or as a full .js script connected to an HTML file

PRIME INTUIT
Finishing School

Introduction

Built in and external APIs

- ✓ Application Programming Interfaces (APIs) built into web browsers, providing functionality such as dynamically creating HTML and setting CSS styles; collecting and manipulating a video stream from a user's webcam, or generating 3D graphics and audio samples.

- ✓ Third-party APIs that allow developers to incorporate functionality in sites from other content providers, such as Twitter or Facebook.

- ✓ There are also third-party frameworks and libraries that you can apply to HTML to accelerate the work of building sites and applications

PRIME INTUIT
Finishing School

First program- Building blocks

Building Blocks

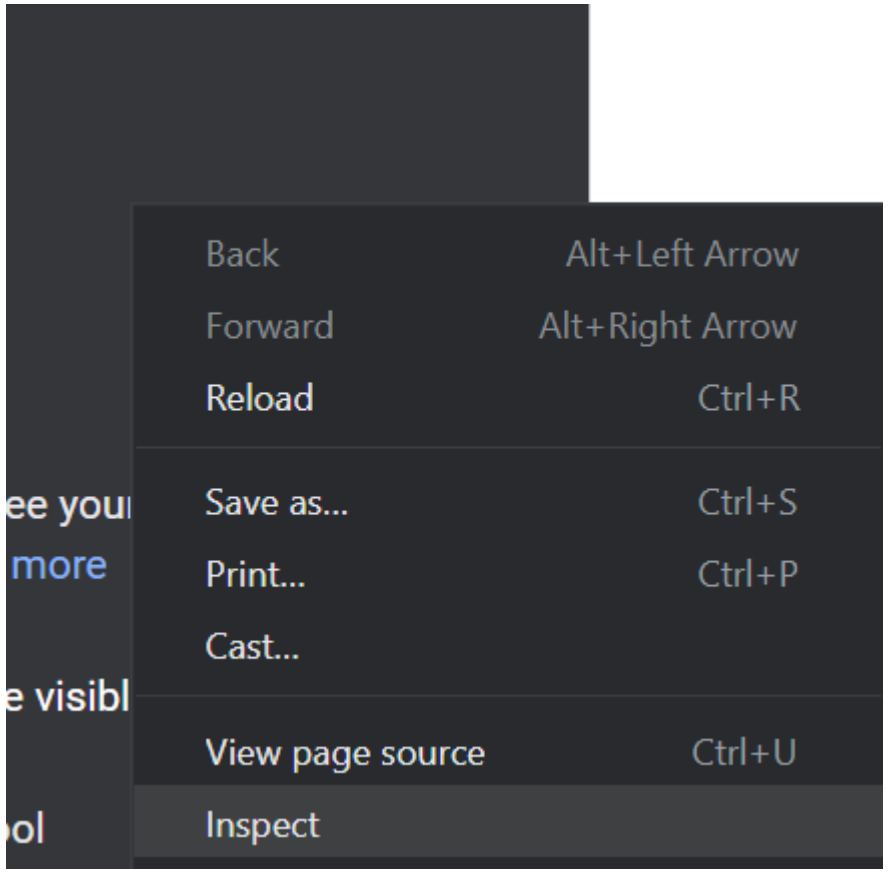
```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
<h1>Hi my first JS program</h1>
<script type="text/javascript" src="D:\ATOM\PI_B3_HTML\Java_script_1.js"></script>
  </body>
</html>
```

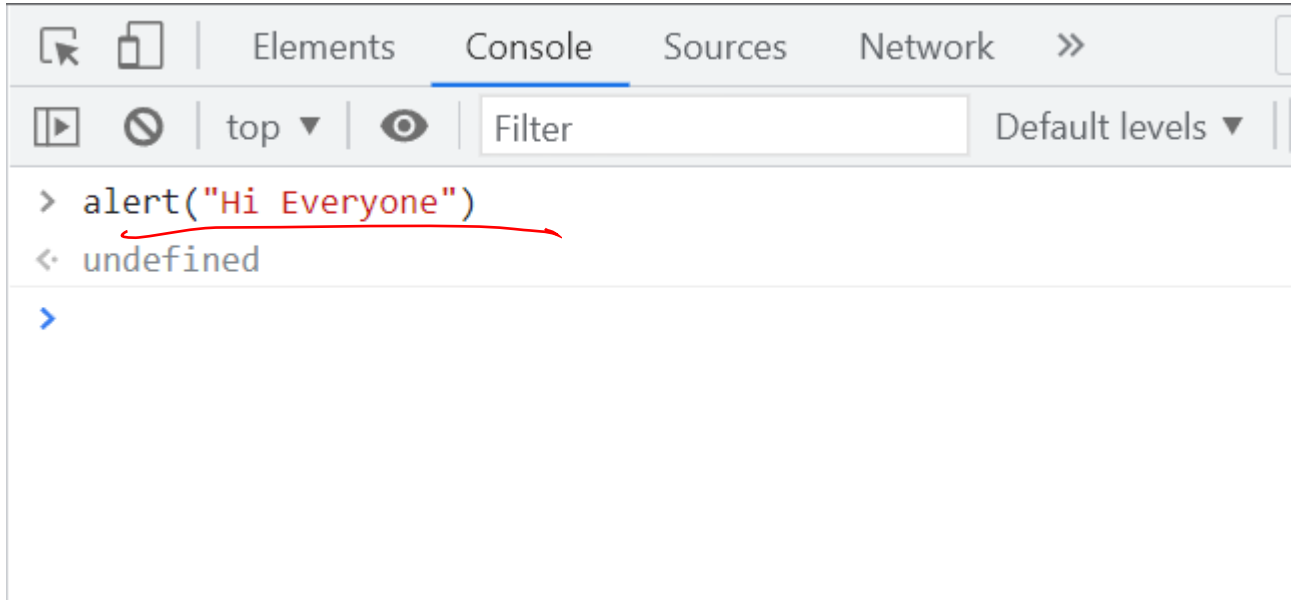
```
const myHeading = document.querySelector('h1');
myHeading.textContent = 'Hello world!';
```

PRIME INTUIT
Finishing School

Introduction

Introduction





PRIME INTUIT
Finishing School

Variables

```
let myVariable;
```

- ✓ Variables are containers that store values. You start by declaring a variable with the let keyword, followed by the name you give to the variable:
- ✓ A semicolon at the end of a line indicates where a statement ends. It's good practice to have semicolons at the end of each statement.

- ✓ You can name a variable nearly anything, but there are some restrictions.

PRIME INTUIT
Finishing School

Variables

Variable Naming:

- ✓ A JavaScript identifier must start with a letter, underscore (_), or dollar sign ($). Subsequent characters can also be digits (0–9).

- ✓ Because JavaScript is case sensitive, letters include the characters "A" through "Z" (uppercase) as well as "a" through "z" (lowercase).

- ✓ You can declare a variable in two ways:
  - With the keyword var. For example, var x = 42. This syntax can be used to declare both local and global variables, depending on the execution context.
  - With the keyword const or let. For example, let y = 13. This syntax can be used to declare a block-scope local variable.

PRIME INTUIT
Finishing School

Data structures and types

Data Types:

Seven data types that are primitives:

- ✓ **Boolean.** true and false.
- ✓ **null.** A special keyword denoting a null value. (Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.)
- ✓ **undefined.** A top-level property whose value is not defined.
- ✓ **Number.** An integer or floating point number. For example: 42 or 3.14159.
- ✓ **BigInt.** An integer with arbitrary precision. For example: 900719925474099 2n.
- ✓ **String.** A sequence of characters that represent a text value. For example: "Howdy"
- ✓ **Symbol.** A data type whose instances are unique and immutable.

❖ There are also Objects and Literals

PRIME INTUIT
Finishing School

Data structures and types

Data Type conversion:

JavaScript is a dynamically typed language. This means you don't have to specify the data type of a variable when you declare it. It also means that data types are automatically converted as-needed during script execution.

```
var result= 101;
result  = "you are lucky"
```

Converting strings to numbers

typecasting

In the case that a value representing a number is in memory as a string, there are methods for conversion.

parseInt()

parseFloat()

parseInt only returns whole numbers, so its use is diminished for decimals.

PRIME INTUIT
Finishing School

Operators:

Operators:

JavaScript has the following types of operators.

- Assignment operators
- Comparison operators
- Arithmetic operators
- Bitwise operators
- Logical operators
- String operators
- Conditional (ternary) operator
- Comma operator
- Unary operators
- Relational operators

PRIME INTUIT
Finishing School

Operators:

Assignment Operators:

Name	Shorthand operator	Meaning
Assignment	`x = f()`	`x = f()`
Addition assignment	`x += f()`	`x = x + f()`
Subtraction assignment	`x -= f()`	`x = x - f()`
Multiplication assignment	`x *= f()`	`x = x * f()`
Division assignment	`x /= f()`	`x = x / f()`
Remainder assignment	`x %= f()`	`x = x % f()`
Exponentiation assignment	`x **= f()`	`x = x ** f()`
Left shift assignment	`x <<= f()`	`x = x << f()`
Right shift assignment	`x >>= f()`	`x = x >> f()`
Unsigned right shift assignment	`x >>>= f()`	`x = x >>> f()`
Bitwise AND assignment	`x &= f()`	`x = x & f()`
Bitwise XOR assignment	`x ^= f()`	`x = x ^ f()`
Bitwise OR assignment	`x |= f()`	`x = x | f()`
Logical AND assignment	`x &&= f()`	`x && (x = f())`
Logical OR assignment	`x ||= f()`	`x || (x = f())`
Logical nullish assignment	`x ??= f()`	`x ?? (x = f())`

PRIME INTUIT
Finishing School

Comparison Operators:

Operator	Description	Examples returning true

Equal (`==`)	Returns `true` if the operands are equal.	`3 == var1`
		`"3" == var1`
		`3 == '3'`

Not equal (`!=`)	Returns `true` if the operands are not equal.	`var1 != 4` `var2 != "3"`
Strict equal (`===`)	Returns `true` if the operands are equal and of the same type.	`3 === var1`
Strict not equal (`!==`)	Returns `true` if the operands are of the same type but not equal, or are of different type.	`var1 !== "3"` `3 !== '3'`
Greater than (`>`)	Returns `true` if the left operand is greater than the right operand.	`var2 > var1` `"12" > 2`
Greater than or equal (`>=`)	Returns `true` if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (`<`)	Returns `true` if the left operand is less than the right operand.	`var1 < var2` `"2" < 12`
Less than or equal (`<=`)	Returns `true` if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

PRIME INTUIT
Finishing School

Arithmetic Operators:

Operator	Description	Example
Remainder (`%`)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
Increment (`++`)	Unary operator. Adds one to its operand. If used as a prefix operator (`++x`), returns the value of its operand after adding one; if used as a postfix operator (`x++`), returns the value of its operand before adding one.	If `x` is 3, then `++x` sets `x` to 4 and returns 4, whereas `x++` returns 3 and, only then, sets `x` to 4.
Decrement (`--`)	Unary operator. Subtracts one from its operand. The return value is analogous to that for the increment operator.	If `x` is 3, then `--x` sets `x` to 2 and returns 2, whereas `x--` returns 3 and, only then, sets `x` to 2.
Unary negation (`-`)	Unary operator. Returns the negation of its operand.	If `x` is 3, then `-x` returns -3.
Unary plus (`+`)	Unary operator. Attempts to convert the operand to a number, if it is not already.	`+"3"` returns `3`. `+true` returns `1`.
Exponentiation operator (`**`)	Calculates the `base` to the `exponent` power, that is, `base^exponent`	`2 ** 3` returns `8`. `10 ** -1` returns `0.1`.

PRIME INTUIT
Finishing School

Operators:

Logical Operators:

Operator	Usage	Description
Logical AND (`&&`)	`expr1 && expr2`	Returns `expr1` if it can be converted to `false`; otherwise, returns `expr2`. Thus, when used with Boolean values, `&&` returns `true` if both operands are true; otherwise, returns `false`.
Logical OR (`||`)	`expr1 || expr2`	Returns `expr1` if it can be converted to `true`; otherwise, returns `expr2`. Thus, when used with Boolean values, `||` returns `true` if either operand is true; if both are false, returns `false`.
Logical NOT (`!`)	`!expr`	Returns `false` if its single operand that can be converted to `true`; otherwise, returns `true`.

PRIME INTUIT
Finishing School

Basic Data Types and Operators

Introduction

```
alert("Hello World")
// anything here is a comment

//Basic data types
//Numbers
10          Int
10.2        Float
-13.4       neg float and int

// Strings

"Hello World"
"10"

// Booleans

true
false
connected to an HTML file
```

```
// undefined and null
undefined
Null

// console clear
clear()

// operations

2+2          addition
5-1          subtraction
3*2          multiplication
10/2         division
2/5          division
2**3         exponent
15 % 4       Modulus
6%2
```

PRIME INTUIT
Finishing School

Strings and String operators

Strings

```
// Stings operations
" I am from Mysore"
"I am form Mysore" + "But currently"
"Mysore".length
" I am alive".length

"Hello \n I am beginning a new line"

"Hello \t give me a tab"

" Hello \ quotes\ "

" Hello \ " Snoopy"

"Hello"[4]
```

```
"I am form Mysore"[8]
```

PRIME INTUIT
Finishing School

Variables

Web browser: inspect / console

```
// Variables
// syntax var varname = value;

var bankAccount = 100;
bankAcccount

Var Depost = 100;
Depost

Var Balance = bankAccount + Depost;
Balance
```

```
Var greetings = “Welcome Back”
Var Name = “ Niranjan”

Alert (greetings + Name)

Var myVariable

Var Bonus = null;
```

PRIME INTUIT
Finishing School

Methods

Web browser: inspect / console

Alert(“Hey !”)

Console.log(“ Hey! Man now your really living”)

Prompt(“Enter something”)

Var age = prompt(“Enter your Age”)

Age

PRIME INTUIT
Finishing School

Connecting JavaScript to an HTML file

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Your Bank</title>
    <script src ="myscript.js"></script>

  </head>
  <body>

  </body>
</html>
```

Atom Editor js

```
alert("welcome to your bank")
var deposit = prompt(" How much would you like to
deposit today:")
alert("you have deposited : " + deposit)
console.log("you are a rock star !")
```

covered

PRIME INTUIT
Finishing School

Exercise

Create a webpage that will take a number in Pounds (lbs) and return Kilograms (Kg)

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>lbs2kgs</title>
    <h1>Pounds to KG Convertor</h1>
  </head>
  <body>
    <p>Pounds to KG Convertor</p>
    <script src ="Convertor.js"></script>
  </body>
</html>
```

Atom Editor js

```
var Pounds = prompt("Please enter the weight in
pounds to convert: ");
var Kg = Pounds * 0.454;
alert("The converted weight in Kilograms is : "+
Kg);
console.log("Conversion Complete");
```

covered

PRIME INTUIT
Finishing School

Operators

Comparison and Logical Operators

```
// Comparison operators
// compare 2 operators and returns a Boolean

// Greater Than
3>2;
2>3;

//Less than
1<2;
2<1;

// Greater then or equal to
2 >= 2

// Less then or equal to
2 <= 1
```

```
//equality

2 == 2
"user" == "user"

2 == "2"

2 === "2"

// inequality

5 != "5"

5 !== "5"
```

True

True

Check it

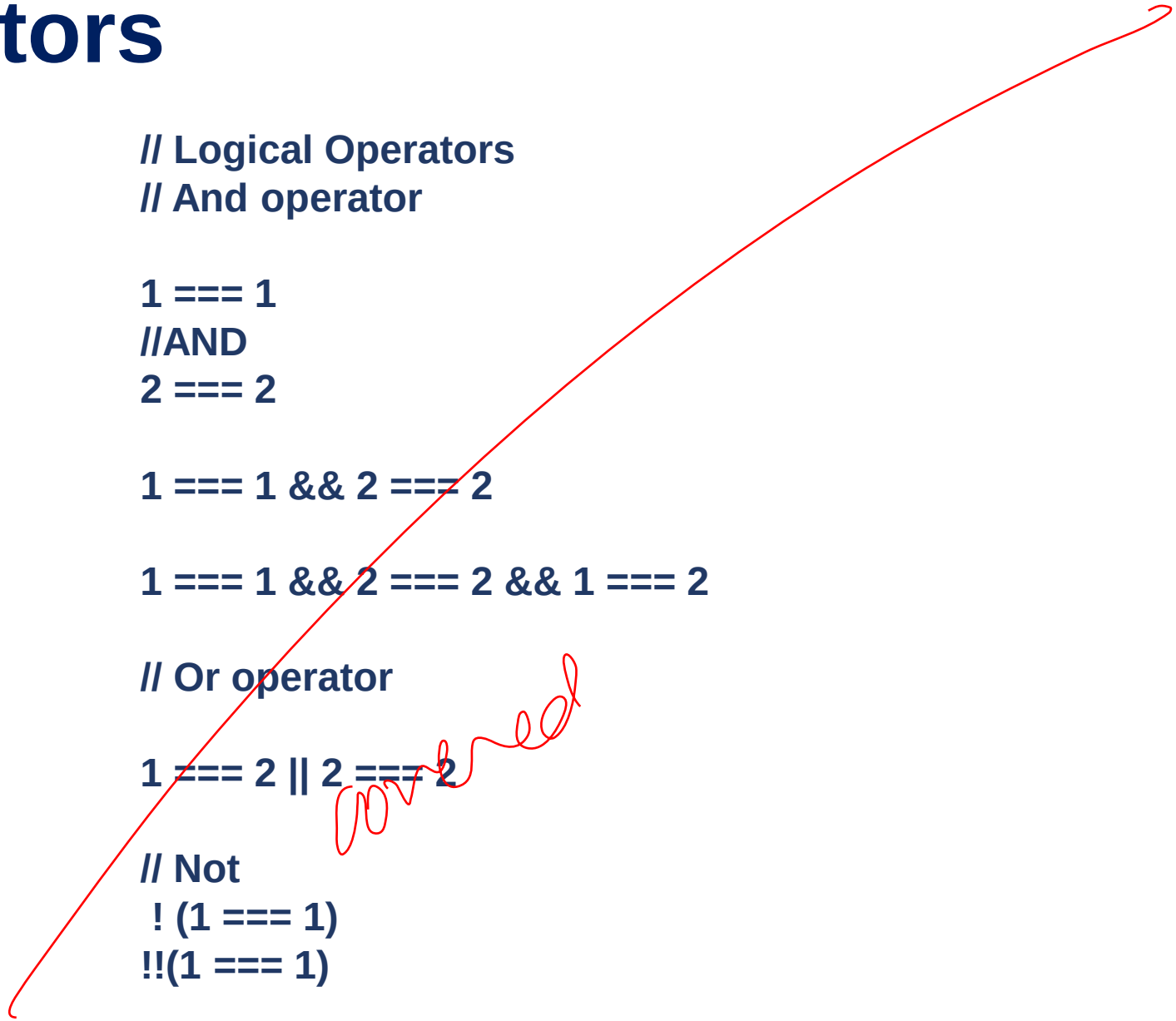
PRIME INTUIT
Finishing School

Operators

Comparison and Logical Operators

```
// Compare True or False to numbers
True == 1
True === 1

False == 1
False === 1

Weird Behavior for null

Null == undefined
Null === undefined

NaN == NaN
```

```
// Logical Operators
// And operator

1 === 1
//AND
2 === 2

1 === 1 && 2 === 2

1 === 1 && 2 === 2 && 1 === 2

// Or operator

1 === 2 || 2 === 2

// Not
 ! (1 === 1)
!!(1 === 1)
```



PRIME INTUIT
Finishing School

Operators

Comparison and Logical Operators

// Examples

Var x = 1
Var y = 2

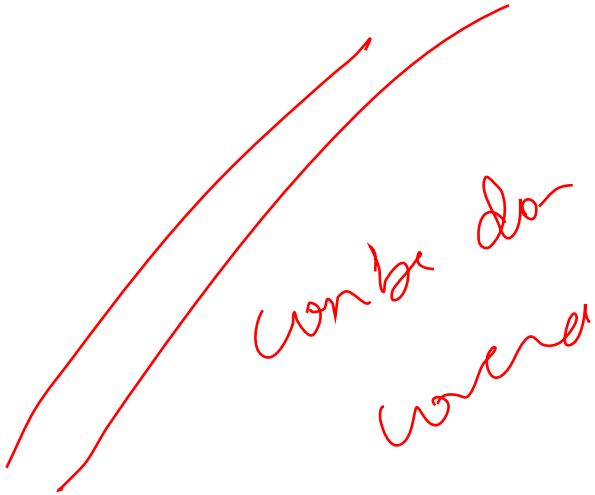
‘2’ == y && x === 1;

X > = 0 || y === 2

!(x != 1) && y ===(1+1);

Y ! == ‘2’ && x < 10;

Y ! == x || y ==‘2’ || x === 3;



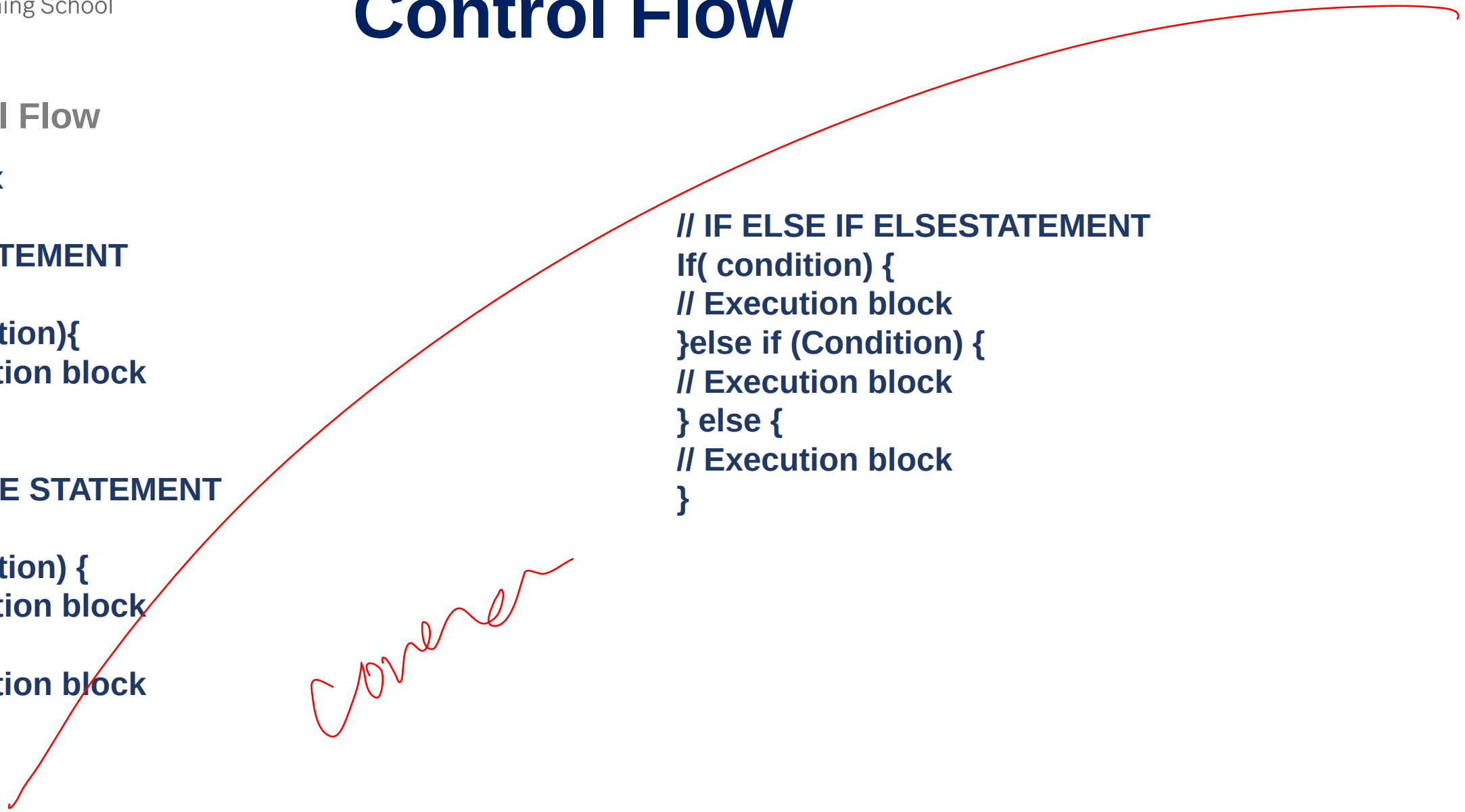
PRIME INTUIT
Finishing School

Control Flow

Control Flow

```
// Syntax

// IF STATEMENT

If (condition){
// Execution block
}

// IF ELSE STATEMENT

If( condition) {
// Execution block
}else{
// Execution block
}
```

```
// IF ELSE IF ELSESTATEMENT
If( condition) {
// Execution block
}else if (Condition) {
// Execution block
} else {
// Execution block
}
```



PRIME INTUIT
Finishing School

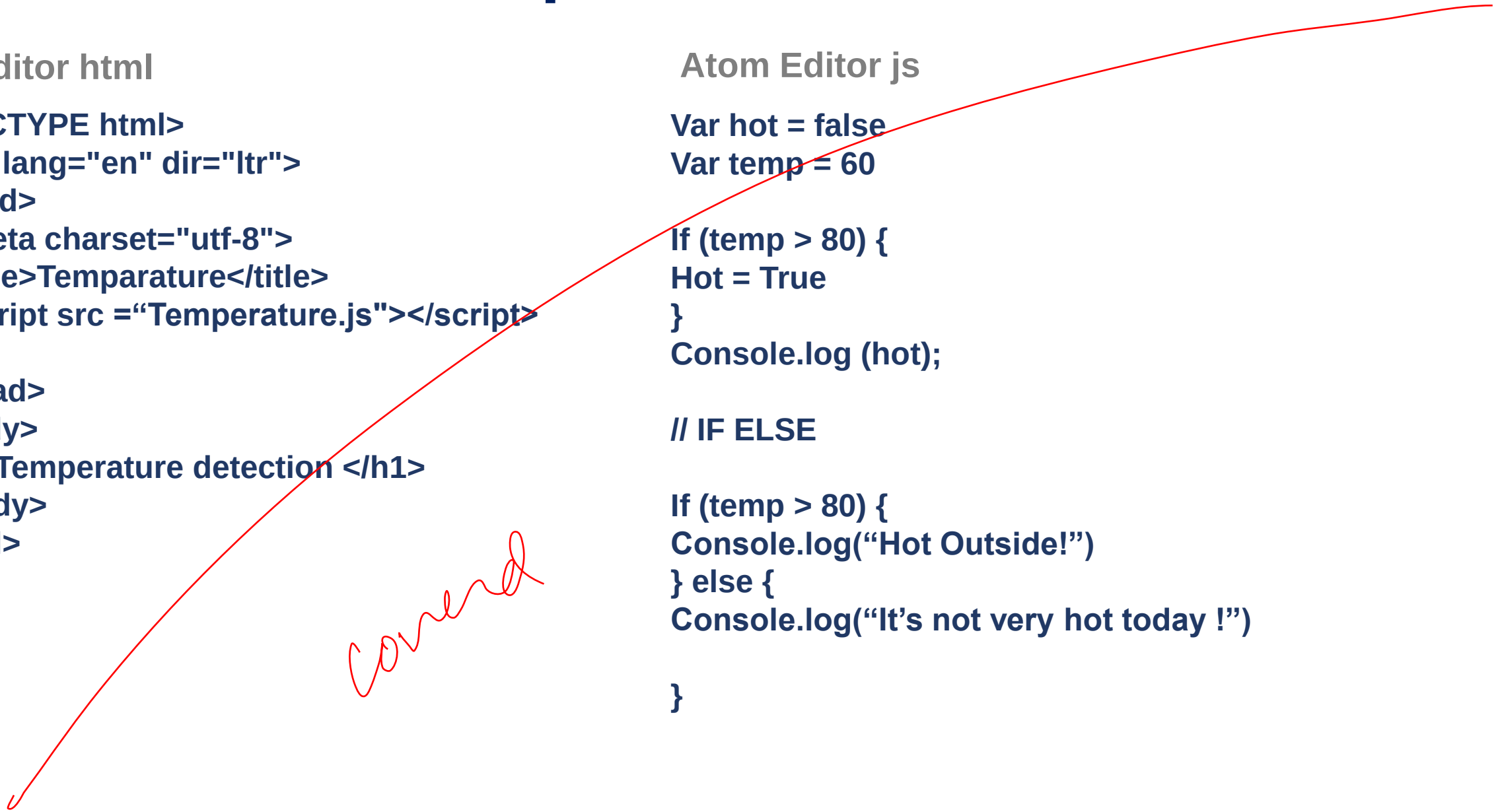
Examples of IF, IF ELSE IF

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
Var hot = false
Var temp = 60

If (temp > 80) {
Hot = True
}
Console.log (hot);

// IF ELSE

If (temp > 80) {
Console.log(“Hot Outside!”)
} else {
Console.log(“It’s not very hot today !”)

}
```



PRIME INTUIT
Finishing School

Examples of IF, IF ELSE IF

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
If (temp > 80) {
Console.log(“Hot Outside!”)
} else if (temp <= 80 && temp >=50){
Console.log(“ average temp outside”)
} else if (temp <= 50 && temp >=32){
Console.log(“ Its pretty gold outside ”)
} else {
Console.log(“ Its cold out there”)

}
```

PRIME INTUIT
Finishing School

Switch Statement

they take a single expression/value as an input, and then look through a number of choices until they find one that matches that value, executing the corresponding code that goes along with it.

```
// Syntax

switch (expression) {
  case choice1:
    run this code
    break;

  case choice2:
    run this code instead
    break;

  // include as many cases as you like

  default:
    actually, just run this code
}
```

- The keyword switch, followed by a set of parentheses.
- An expression or value inside the parentheses.
- The keyword case, followed by a choice that the expression/value could be, followed by a colon.
- Some code to run if the choice matches the expression.
- A break statement, followed by a semi-colon. If the previous choice matches the expression/value, the browser stops executing the code block here, and moves on to any code that appears below the switch statement.
- As many other cases (bullets 3–5) as you like.
- The keyword default, followed by exactly the same code pattern as one of the cases (bullets 3–5), except that default does not have a choice after it, and you don't need to break statement as there is nothing to run after this in the block anyway. This is the default option that runs if none of the choices match.

PRIME INTUIT
Finishing School

Switch Statement

```
let day = 3;
let dayName;

switch (day) {
  case 1:
    dayName = 'Sunday';
    break;
  case 2:
    dayName = 'Monday';
    break;
  case 3:
    dayName = 'Tuesday';
    break;
  case 4:
    dayName = 'Wednesday';
    break;
  case 5:
    dayName = 'Thursday';
    break;
  case 6:
    dayName = 'Friday';
    break;
  case 7:
    dayName = 'Saturday';
    break;
  default:
    dayName = 'Invalid day';
}
```

covered

PRIME INTUIT
Finishing School

While Loop

```
// Syntax

While( Condition){
// Execute some Code
}
```

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
Var x = 0
While (x<5){
Console.log(‘x is currently :’ +x);
X = x+1;
};
```

```
var x = 0
while (x<5){
console.log("x is currently :" +x);
if (x === 3){
console.log("x is now equal to 3")
break;
}
x = x+1;
}
```

PRIME INTUIT
Finishing School

While Loop

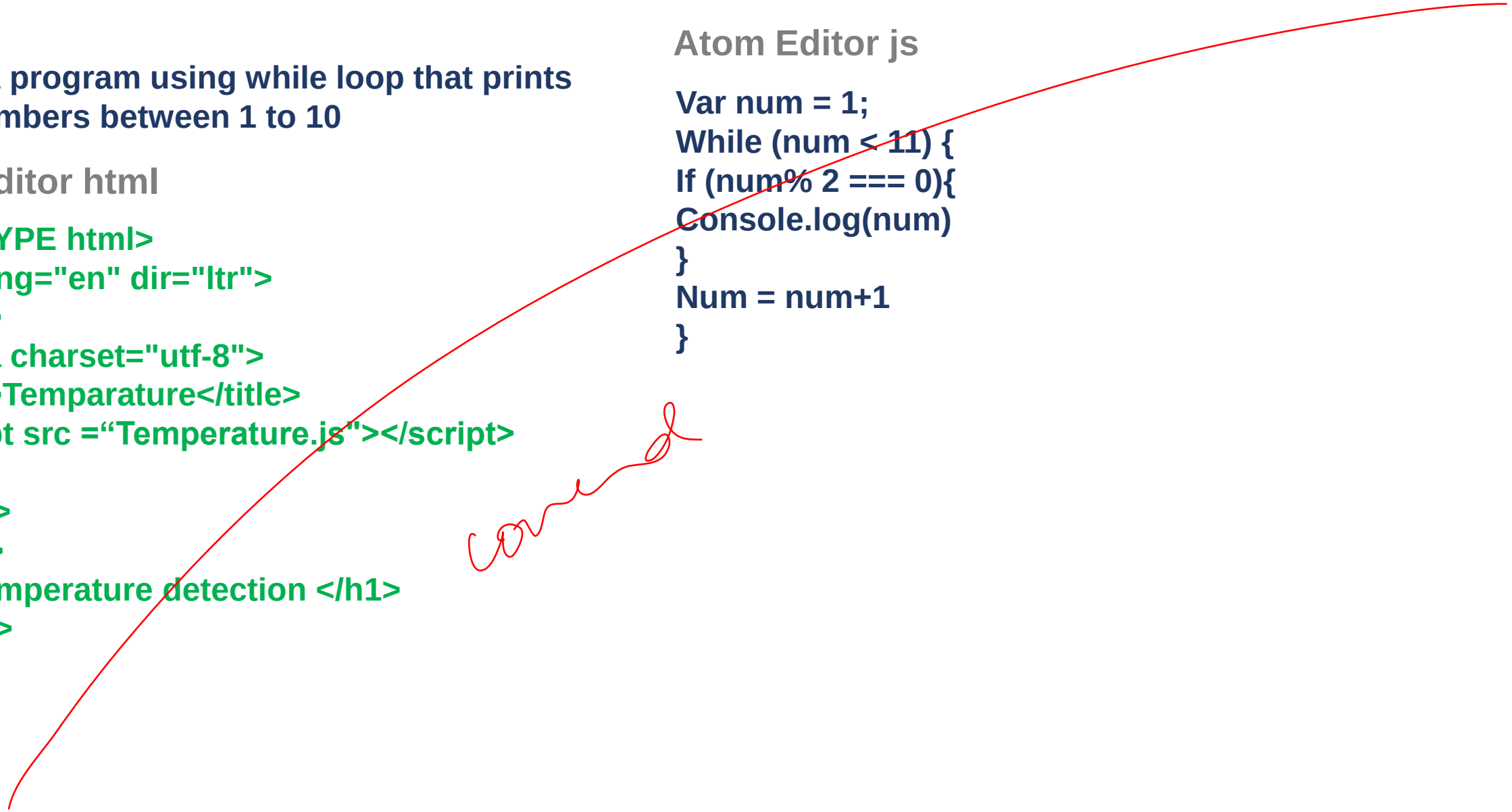
// write a program using while loop that prints even numbers between 1 to 10

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
Var num = 1;
While (num < 11) {
If (num% 2 === 0){
Console.log(num)
}
Num = num+1
}
```



PRIME INTUIT
Finishing School

For Loop

```
// syntax
For (statement 1; statement 2; statement 3 ;){
// Execute some code;
}
```

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

```
// Statement 1 gets executed before the loop (
code block) starts
// statement 2 defines the condition of execution
// statement 3 is executed each time the loop
executes through
// syntax
For (var i = 0; I < 5; i++ ;){
Console.log(“number is = ” +i)
}
```

Atom Editor js

```
Var word = “abcdefghi”;
for (I = 0; I < word.length; i++) {
If (num% 2 === 0){
Console.log(word[i])
}
```

PRIME INTUIT
Finishing School

Putting Loops into practice

// Print Hello 5 times using both the loops

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

```
Var x = 0
While (x < 5) {
Console.log(“hello!”)
X++
}
```

Atom Editor js

```
for (I = 0; I < 5; i++) {
Console.log(“hello!”)
}
```

// Print all odd numbers between 1 to 25

PRIME INTUIT
Finishing School

Mini Project

// build a calculator web page

PRIME INTUIT
Finishing School

Functions

// Syntax

```
Function name( parameter 1, parameter 2)
{ // code to be executed
}
```

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
function hello(){
console.log(“Hello Prime Intuit”);
}

hello()

// you can call the function with in the console also

function helloyou( name){
console.log(“Hello ” + name);
}

hello()

// you can call the function with in the console also
```

PRIME INTUIT
Finishing School

Functions

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
function addnum (num1, num2){
Console.log(num1 +num2)
}

addnum(12+13)
addnum(“Hello”+”Ram”)


function hellosomeone( name=John){
console.log(“Hello ” + name);
}

//hellosomeone(“Rao”)
//hellosomeone( )
// welcome + hellosomeone          ???

// you can call the function with in the console also
```

PRIME INTUIT
Finishing School

Functions

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
function max (num1, num2){
If (num1 > num2){
return num1}
Else{
return num2
}

function timesfive( num){
var result = num * 5
return result
}

//Scope
//var defined within a function is limited to that
function
```

PRIME INTUIT
Finishing School

Functions

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Temparature</title>
    <script src =“Temperature.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
Var v = ‘global variable’
Var stuff = ‘global stuff’

Function fun(stuff){
Console.log(v);
Stuff = ‘reassigned stuff inside function”
Console.log(stuff);
}

Fun()

Console.log(stuff)
```

PRIME INTUIT
Finishing School

Arrays

// Data type used to store a group of items, similar to lists in Python

Syntax: var fruits = ["apple", "orange", "mango"];

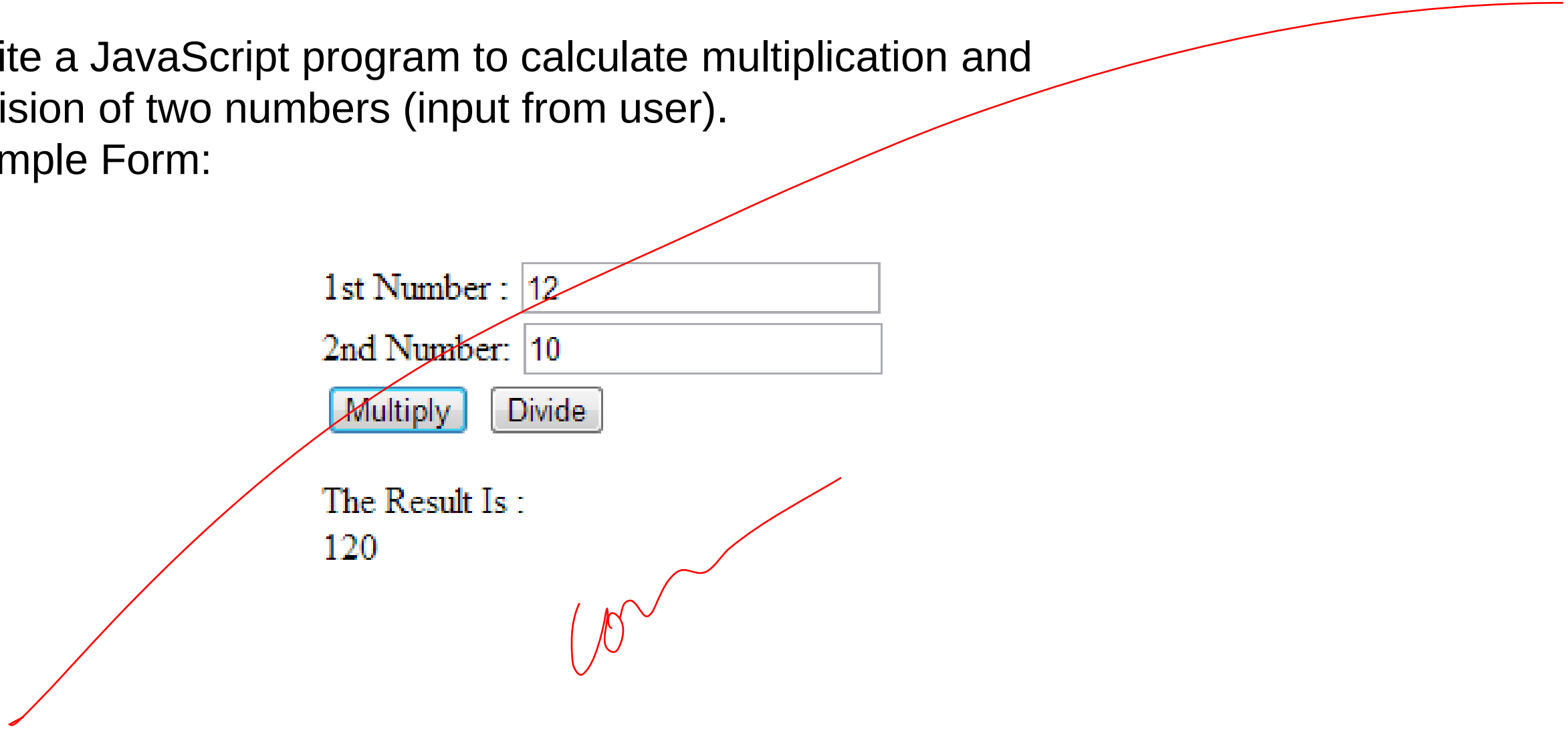
Atom Editor html - Inline script

```
<html>
<head>
<title>JavaScript Array length Property</title>
</head>
<body>
<script type="text/javascript">
 var arr = [ 10, 20, 30 ];
 document.write("arr.length is:" + arr.length);
</script>
</body>
</html>
```

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element

Javascript array length property returns an unsigned, 32-bit integer that specifies the number of elements in an array.

Write a JavaScript program to calculate multiplication and division of two numbers (input from user).
Sample Form:



PRIME INTUIT
Finishing School

Calculate multiplication and Division

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>JavaScript program to calculate
multiplication and division of two numbers </title>
<style type="text/css">
body {margin: 30px;}
</style>
</head>
<body>
<form>
1st Number : <input type="text" id="firstNumber" /><br>
2nd Number: <input type="text" id="secondNumber" /><br>
<input type="button" onClick="multiplyBy()"
Value="Multiply" />
<input type="button" onClick="divideBy()" Value="Divide" />
</form>
<p>The Result is : <br>
<span id = "result"></span>
</p>
</body>
</html>
```

PRIME INTUIT
Finishing School

Calculate multiplication and Division

```
function multiplyBy()
{
	num1 = document.getElementById("firstNumber").value;
	num2 = document.getElementById("secondNumber").value;
	document.getElementById("result").innerHTML = num1 * num2;
}

function divideBy()
{
	num1 = document.getElementById("firstNumber").value;
	num2 = document.getElementById("secondNumber").value;
document.getElementById("result").innerHTML = num1 / num2;
}
```

PRIME INTUIT
Finishing School

Arrays

// Methods

Atom Editor html - Inline script

```
<html>
<head>
<title>JavaScript Array concat Method</title>
</head>
<body>
<script type="text/javascript">
 var alpha = ["a", "b", "c"];
 var numeric = [1, 2, 3];
 var alphaNumeric = alpha.concat(numeric);
 document.write("alphaNumeric : " + alphaNumeric
);
</script>
</body>
</html>
```

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

PRIME INTUIT
Finishing School

Arrays

// Methods

Atom Editor html - Inline script

```
<html>
<head>
<title>JavaScript Array sort Method</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array("orange", "mango", "banana",
"sugar");
var sorted = arr.sort();
document.write("Returned string is : " + sorted );
</script>
</body>
</html>
```

map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

PRIME INTUIT
Finishing School

Arrays

// Methods

Atom Editor html - Inline script

```
<html>
<head>
<title>JavaScript Array slice Method</title>
</head>
<body>
<script type="text/javascript">
var arr = ["orange", "mango", "banana", "sugar", "tea"];
document.write("arr.slice( 1, 2) : " + arr.slice( 1, 2) );
document.write("<br />arr.slice( 1, 2) : " + arr.slice( 1, 3)
);
</script>
</body>
```

toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

PRIME INTUIT
Finishing School

Objects

// JS Object is similar to Dictionaries in python, they are used to store Key and a value pair.
// Syntax: { key1: “Value 1”, key2: “Value 2”, key3: “Value 3”}

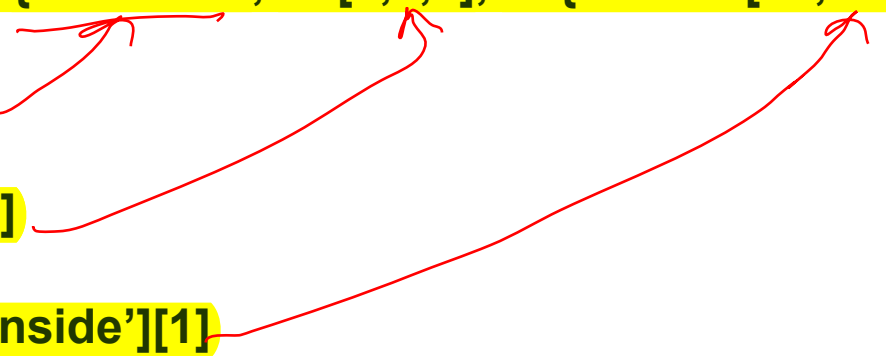
//It is not ordered list of data

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Practice</title>
    <script src =“Practice.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
Var CarInfo = { make: "Toyato", year: 1990, model: "Camry"};
CarInfo
CarInfo["make"]

CarInfo[“year”] = 2006

CarInfo[“year”] += 1

Var myNewO = {a: “Hello”, b: [1,2,3], c: {inside: [‘a’, ‘b’]}};
myNewO
myNewO[“a”]

myNewO[‘b’][1]

myNewO[‘c’][‘inside’][1]

Console.dir(myNewO)
```



// Print all odd numbers between 1 to 25

PRIME INTUIT
Finishing School

Iterating through Objects

// JS Object are not stored in an orderly manner

We your for in to iterate through an object

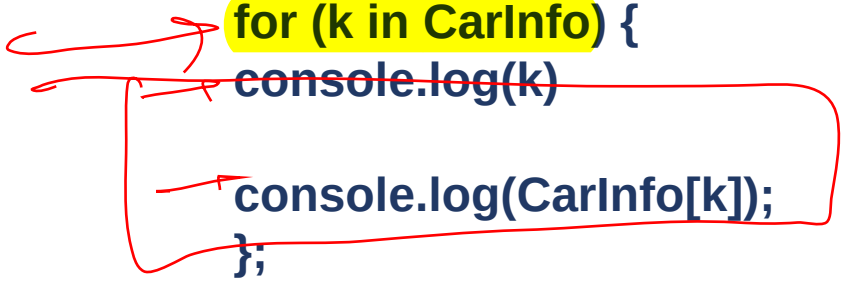
Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Practice</title>
    <script src =“Practice.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
for (k in CarInfo) {
console.log(k)
};
```

```
for (k in CarInfo) {
console.log(k)

console.log(CarInfo[k]);
};
```



//

PRIME INTUIT
Finishing School

Objects Methods

// The This key word is set to the Object the method is called on.

//

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Practice</title>
    <script src =“Practice.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

Atom Editor js

```
var myObj = {
  Prop: 37, reportProp: function(){
    return this.Prop;
  }
}
console.log(myObj.reportProp())


var Simple = {
  Prop: "Hello",
  myMethod: function(){
    console.log("This method was
called");
  }
}
Simple
console.dir(Simple);
Simple.myMethod()
```

PRIME INTUIT
Finishing School

Objects Methods

// The This key word is set to the Object the method is called on.

//

Atom Editor html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Practice</title>
    <script src =“Practice.js"></script>

  </head>
  <body>
<H1> Temperature detection </h1>
  </body>
</html>
```

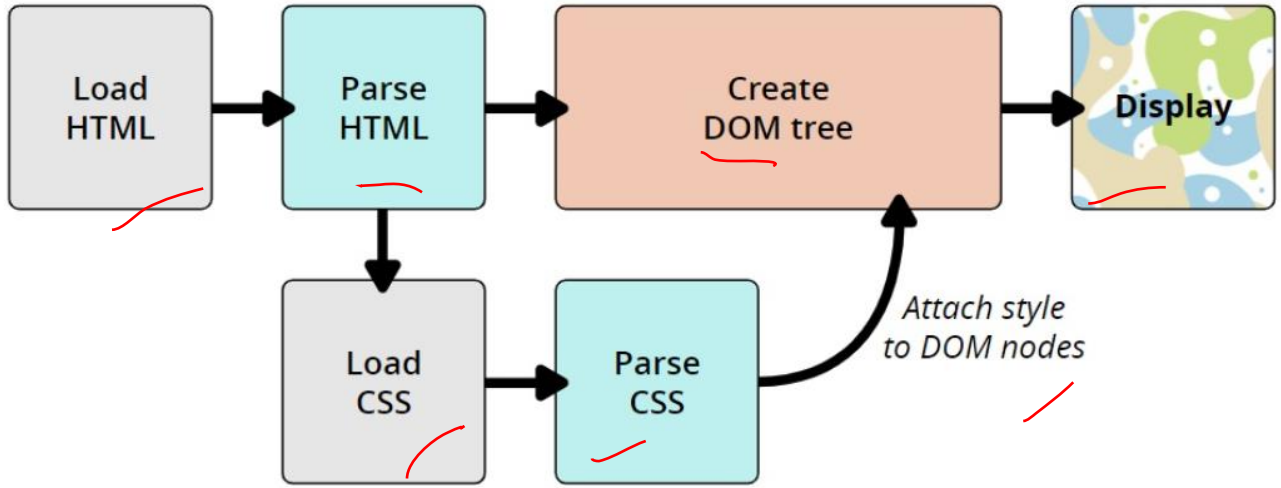
Atom Editor js

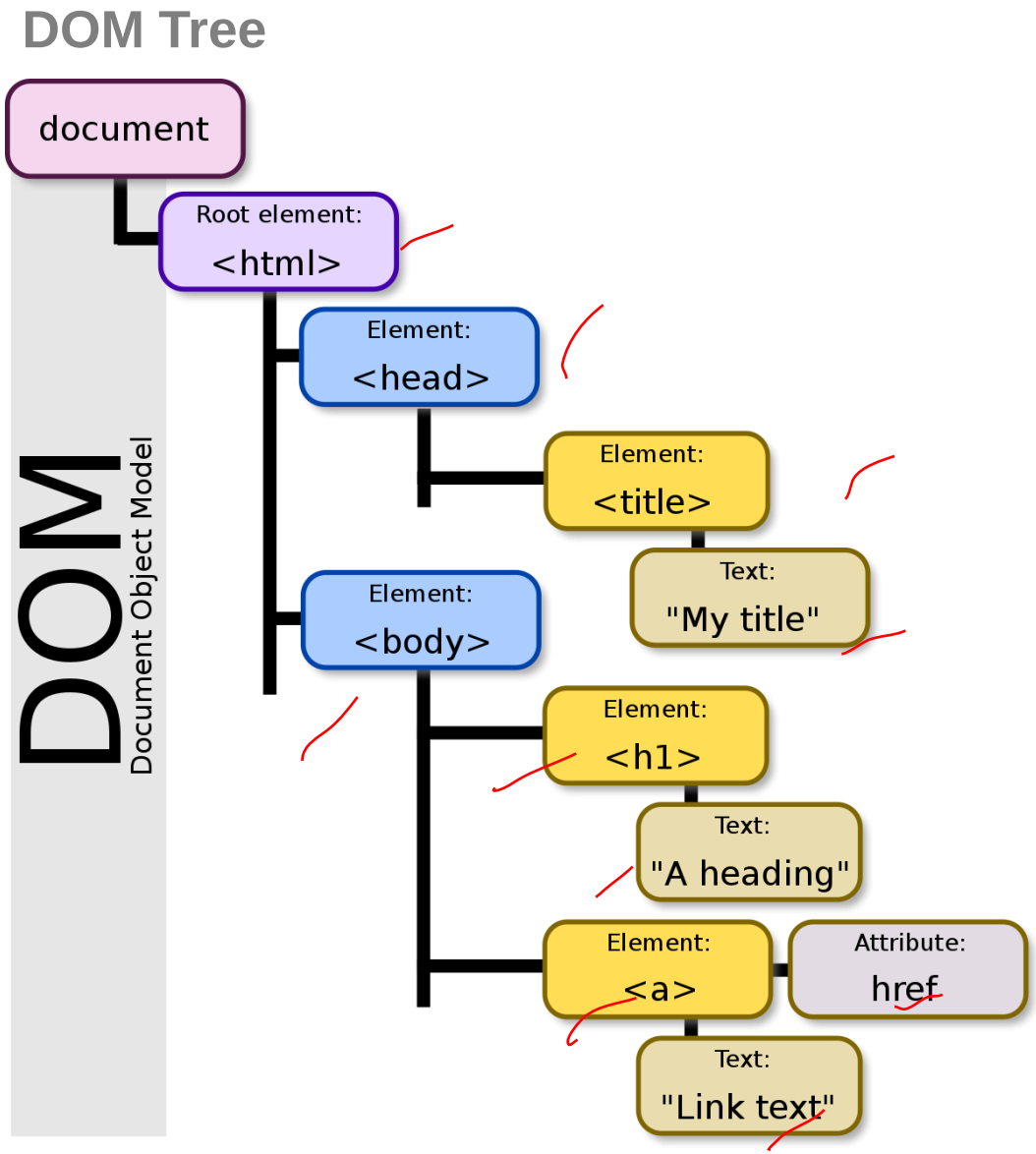
```
var myName = {
  name: "Prime Intuit",
  greet: function(){
    console.log("Welcome to " + this.name);
  }
}

myName
myName.greet()
```

PRIME INTUIT
Finishing School

Document Object Model

// DOM will allow us to directly interface our Javascript code to interact with HTML and CSS.
// Browser will construct the DOM, which basically means storing all the HTML tags as Javascript objects
// We can call this DOMs in our JS
// We can see DOM of any website by simply typing document in Console of that website.
Use console.dir(docment)

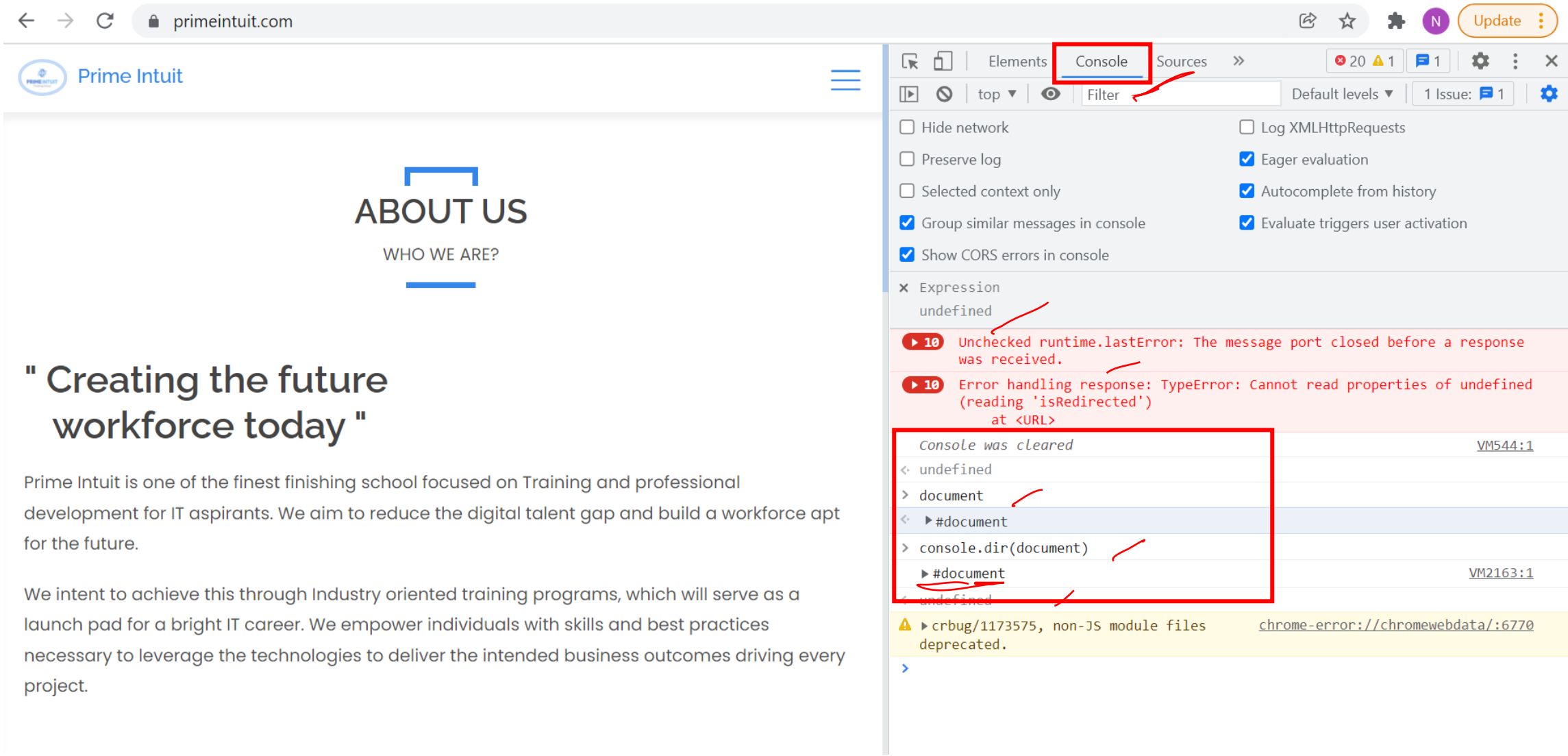




PRIME INTUIT
Finishing School

Document Object Model

DOM Tree



PRIME INTUIT
Finishing School

Document Object Model

Document attributes

```
// How to grab HTML elements from the DOM
// The HTML elements are properties of the DOM
// First we will grab large groups of elements like the entire body or the head of the HTML
// Then grab the specific HTML items like classes or ids

// Some important document attributes:

documents.URL
documents.body
documents.head
documents.links

// Grabing specific HTML items
documents.getElementById()
documents.getElementByClassName()
documents.getElementByTagName()
documents.querySelector()     // first object matching the CSS Style selector
documents.querySelectorAll()
```

PRIME INTUIT
Finishing School

Document Object Model

DOM Tree

```
var myheader = document.querySelector("h1")
myheader

myheader.style.color='red';


var p = document.querySelectorAll('p')

var p = document.querySelector('p.about-content')

p.textContent = "New Text Ginga Lala"

p.innerHTML = "<strong> I'm Bold </strong>"
```

PRIME INTUIT
Finishing School

jQuery Basic

http://code.jquery.com/

```
<script src="https://code.jquery.com/jquery-3.6.0.js" integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk=" crossorigin="anonymous"></script>
```

Copy link into your html file

In the console

$
$('h1')
$('li')

Thank You!