420-905 A20 Introduction to Programming

Arithmetic Operators

Arithmetic operators can be applied to numbers, variables, or object properties with numerical values.

Operator	Example and Comment
`+`	`5 + 5;` // Returns 10 (addition)
`-`	`20 - 5;` // Returns 15 (subtraction)
`*`	`5 * 5;` // Returns 25 (multiplication)
`/`	`12.5 / 5;` // Returns 2.5 (division)
`%`	`9 % 2;` // Returns 1 (remainder of division)

Declaring and Initializing a Variable

Declaring a variable creates a memory space to store a value or object. Use the keyword "***var***" followed by a name.

Optionally, you can assign an initial value to the variable at the time of declaration:

```
var nb1 = 25;     // Creates a variable and initializes it
var nb2 = 5 * 5; // Creates a variable and initializes it
var nb3;          // Creates a variable without initializing it
```

Declaring a Constant

A constant is a memory space to store a value that cannot be modified later. Use the keyword "***const***" instead of "***var***":

```
const PI = 25;
const MOVEMENT = 20;
```

String Concatenation

When at least one of the operands is a string, the **"+"** operator concatenates (joins) them into a new string:

```
var firstName = "Isabelle";
var numDogs = 2;
var result = firstName + " has " + numDogs + " dogs";
// result is "Isabelle has 2 dogs"
```

To preserve the result of an expression, use an assignment operator.

Assignment Operators

Assignment operators modify a variable or object property by assigning it the result of an expression:

Operator	Example and Comment
`=`	`myVar = 25;` // Stores 25 in `myVar`
`+=`	`myVar += 25;` // Equivalent to `myVar = myVar + 25;`
`-=`	`myObj.myProp -= 25;` // Equivalent to `myObj.myProp = myObj.myProp - 25;`
`*=`	`myObj.myProp *= 25;` // Equivalent to `myObj.myProp = myObj.myProp * 25;`
`/=`	`myVar /= 25;` // Equivalent to `myVar = myVar / 25;`
`%=`	`myObj.myProp %= 25;` // Equivalent to `myObj.myProp = myObj.myProp % 25;`
`++`	`myVar++;` // Equivalent to `myVar = myVar + 1;`
`--`	`myObj.myProp--;` // Equivalent to `myObj.myProp = myObj.myProp - 1;`

Functions

Declaring a Function

```
function myFunction() {
    // instructions to execute
    // ...
}
```

Calling a Function

```
myFunction();
```

Generating a Random Number

```
var number = generateRandomNumber(0, 100);
```

Events

Certain objects have events. You can attach an event listener to an object's event to execute a function automatically when the event occurs.

Examples:

- `click` : Triggered when the user presses and releases the left mouse button on an object.

```
anObject.addEventListener("click", functionName);
```

- `mouseover` : Triggered when the mouse pointer is over an object.

```
anObject.addEventListener("mouseover", functionName);
```

- `mouseout` : Triggered when the mouse pointer leaves the surface of an object.

```
anObject.addEventListener("mouseout", functionName);
```

Comparison Operators

These operators compare the results of two expressions, returning a boolean value (“***true”*** or “***false”***):

Operator	Example and Comment
`==`	`5 + 1 == 6;` // Is equal? `true`
`<=`	`5 <= 5;` // Is less than or equal? `true`
`<`	`5 < 5;` // Is less than? `false`
`>=`	`5 >= 5;` // Is greater than or equal? `true`
`>`	`5 > 5;` // Is greater than? `false`
`!=`	`3 != 2 + 1;` // Is not equal? `false`

Logical Operators

Operator	Description		
`&&`	Logical AND (all conditions must be true)		
`		`	Logical OR (only one condition needs to be true)
`!`	Logical NOT (reverses the result of a condition)		

Examples:

```
(age >= 18) && (age < 65);   // AND operator
(age < 12) || (age >= 65);   // OR operator
!(age < 18);                 // NOT operator
```

Conditional Statements: if, if/else, if/else if/else

Conditional statements allow the execution of instruction blocks based on conditions that return boolean values.

Simple Selection: if

```
if (/* condition */) {
    // Instructions executed when the condition is true
}
```

Simple Selection with if/else

```
if (/* condition */) {
    // Instructions executed when the condition is true
} else {
    // Instructions executed when the condition is false
}
```

Multiple Selection with if/else if/else

```
if (/* condition 1 */) {
// Instructions executed when condition 1 is true
} else if (/* condition 2 */) {
//---Executed when condition 1 is false and condition 2 is true
} else {
  //---Executed when conditions 1 and 2 are false
}
```

setTimeout, setInterval, and clearInterval

setTimeout: Schedule a task to execute later:

```
setTimeout(aFunctionToExecute, 1000);  // Executes the after 1 second
```

setInterval: Repeat a task at regular intervals:

```
var gInterval = null;
gInterval = setInterval(aFunctionToExecute, 1000);  // Executes every second (1000ms)
```

clearInterval: Stop a repeated task:

```
clearInterval(gInterval);
gInterval = null;  // Useful to track that the task is stopped
```

The "evt" Object

When a function is called by an event listener, it sends a parameter to the function in the form of an object (named *"**evt**"* here) containing information about the event.

The "target" Property of "evt" The *"**evt**"* object can have a *"**target**"* property during a mouse-related event, containing a reference to the object that triggered the event.

```
gMarioImage.addEventListener("click", clickOnCharacter);
gLuigiImage.addEventListener("click", clickOnCharacter);

function clickOnCharacter(evt) {
    var targetCharacter = evt.target;
    targetCharacter.scale = 1.1;
    targetCharacter.alpha = 0.5;
}
```

The "key" Property of "evt" The evt object can have a key property during keyboard-related events, containing a string representing the key pressed. These events must be attached to the document object.

```
document.addEventListener("keydown", someFunction);

const KEY_LEFT = "ArrowLeft";
const KEY_RIGHT = "ArrowRight";
const KEY_UP = "ArrowUp";
const KEY_DOWN = "ArrowDown";

function someFunction(evt) {
    var key = evt.key;
    if (key == KEY_LEFT) {
        // Move an element left...
    } else if (key == KEY_RIGHT) {
        // Move an element right...
    }
}
```

Using "this" Instead of "evt.target"

For mouse-related events, the object triggering the event is referenced by "***this"***. This is equivalent to "***evt.target***", but simpler.

```
function clickOnCharacter() {
    this.scale = 1.1;
    this.alpha = 0.5;
}
```

Global vs. Local Variables

Variables defined outside braces "***{}***" are global and accessible throughout the application. Variables defined inside braces are local to that block and are removed afterward. Local variables are only accessible within the block where they were defined.

```
var x = 3; // This variable is global

function someFunction() {
    var y = 5;
    // This variable is local to the function
    console.log(y);
}
```

The "*alert*" Function

The "***alert"*** function displays an informational window with a message for the user. It takes a parameter representing the value to display.

```
alert("Message to display");
alert(gMarioImage.alpha);
```

The "console.log" Function

The "***log***" function of the "***console"*** object displays information in the debug console. It is widely used to understand code execution. It takes a parameter representing the value to display.

```
function someFunction(evt) {
    var key = evt.key;
    console.log("The key is: " + key);
}
```

Functions with Parameters

Use parameters to pass information to a function. Add the parameters in the function declaration (after its name in parentheses) with meaningful names. You can then use the passed values by referring to the parameter names.

```
/**
 * Description: Displays the smaller of two numbers
 * @param num1: First of two numbers to compare
 * @param num2: Second of two numbers to compare
 */
function displayMinimum(num1, num2) {
    if (num1 < num2) {
        alert(num1);
    } else {
        alert(num2);
    }
}

// Call the function with 12 and 55
displayMinimum(12, 55);

// Call the function with 44 and 27
displayMinimum(44, 27);

// Call the function with 12 and 12
displayMinimum(12, 12);
```

Best Practices:

- Use functions to group code and avoid repetition.
- Use parameters to pass information to a function.
- Make functions more reusable and general by using parameters.
- A function can have no parameters, one parameter, or multiple parameters.
- Parameters can be boolean values, numbers, strings, or objects.

The "for" Loop

The "***for***" loop has three sections: initialization, execution condition, and increment/decrement.

- Initialization: Executes once before the loop starts.
- Execution Condition: Evaluates before each iteration to determine if the loop continues.
- Increment/Decrement: Usually modifies the variable controlling the loop.

```
for (initialization; condition; increment) {
    // Instructions executed when the condition is true
    // The loop repeats the number of iterations
}
```

Example: Loop with 5 Iterations

```
for (var index = 0; index < 5; index++) {
    // The code inside this loop will execute 5 times
    console.log(index);
}
```

One-Dimensional Arrays

An array stores multiple values in a single variable.

- Define a one-dimensional array containing multiple elements (usually of a similar nature):

```
var myArray = ["first", "second", "third"];
```

- Access a specific array element using an index:

```
var firstElement = myArray[0];
// Retrieves the first element of the array
```

- Replace a specific element using an index and an assignment operator:

```
myArray[2] = "new value";
// Replaces the third element of the array
```

- Get the number of elements in the array using the `length` property:

```
myArray.length;
```

Iterating Through a One-Dimensional Array

The "***for***" loop is often used with arrays to iterate through the elements.

Example 1: Display All Elements from First to Last

```
for (var index = 0; index < myArray.length; index++) {
    var currentElement = myArray[index];
    console.log(currentElement);
}
```

Example 2: Display All Elements from Last to First

```
for (var index = myArray.length - 1; index >= 0; index--) {
    var currentElement = myArray[index];
    console.log(currentElement);
}
```

Array Methods

`: Add an element to the end of an array

```
var myArray = ["first", "second", "third"];
myArray.push("fourth"); // Adds this element to the end of the array
```

`: Remove the last element of an array

```
var myArray = ["first", "second", "third"];
var lastElement = myArray.pop(); // Removes and retrieves the last element of the arra
```

`: Remove the first element of an array

```
var myArray = ["first", "second", "third"];
var firstElement = myArray.shift(); //Removes and retrieves the first element of the a
```

`: Remove one or more elements from an array at a specific index

Example 1: Remove One Element at Index 2

```
var myArray = ["first", "second", "third", "fourth", "fifth"];
myArray.splice(2, 1); // Removes 1 element starting at index 2
```

Example 2: Remove 3 Elements Starting from Index 0 and Store Them in a New Array

```
var myArray = ["first", "second", "third", "fourth", "fifth"];
// Remove 3 elements starting from index 0 and store them in removedElements
var removedElements = myArray.splice(0, 3);
```