JavaScript I

(Semana 2)

MigraCode

Alexandra Yamaui

Expresiones e instrucciones

Expresión (expression):

- Retorna un valor (*evalúa* a un valor)
- Se puede escribir en cualquier lugar en el que se espere un valor (valor de una variable, paŕametro, …)

example.js

```
1 + 1; // returns 2

"hello"; // returns "hello"

2 * 4; // returns 8

"Hello" + "World"; // returns "HelloWorld"
```

Expresiones e instrucciones

Expresión (expression):

- Retorna un valor (*evalua* a un valor)
- Se puede escribir en cualquier lugar en el que se espere un valor (valor de una variable, paŕametro, …)

example.js

```
function greetingPlanet() {
 const planet = "Earth"; // expression
 return `Hello ${planet}`; // expression
}
console.log(`2 + 4 is ${2 + 4}`); //
expression

function double(num) {
 return num * 2; // expression
}
```

Expresiones e instrucciones

Instrucción (statement):

- Realiza una acción

example.js

```
const sum = 1 + 1; // action: assigns result of `1 + 1` to variable `sum`

const greeting = "hello"; // action: assigns result of the expression "hello" to variable `greeting`

console.log(2 * 4); // action: logs the result of `2 * 4` to the console

sayGreeting(greeting); // action: calls the function `sayGreeting` with the parameter `greeting`
```

Ejercicios

Ejercicio A (5 mins)

En el terminal de VS Code, ejecuta el comando `node` y ejecuta las siguientes expresiones. Cual es el resultado? Hay algo que te parezca inesperado? (Para salir del node REPL, tienes que presionar Ctrl+d o Cmd+D en Mac)

- `1 + 2`
- `"hello"`
- `let favouriteColour = "purple"`
- `favouriteColour`
- `console.log(favouriteColour)`

Tipos de datos

- Number
- String
- Boolean
- Function
- Array
- Object
- Undefined
- Null

Booleanos

- Tipo de dato
- Tiene sólo dos posibles valores `true` (verdadero) o `false` (falso)

example.js

```
let codeYourFutureIsGreat = true;

let thisIsATerribleClass = false;
```

Booleanos

Booleano:

- Todo lo que sea un valor *evalúa* a true
- Todo lo que no tenga un valor *evalúa* a false
- La función Boolean() puede usarse para determinar si una expresión es true o false

example.js

```
Boolean(100); // true
Boolean(3.14); // true
Boolean(-15); // true
Boolean("Hello"); // true
Boolean("false"); // true
Boolean(7 + 1 + 3.14); // true
Boolean(""); // false
Boolean(0); // false
Boolean(undefined); // false
Boolean(null); // false
```

Operadores de comparación

Comparación:

- Expresión que retorna un valor booleano
- Se usa para determinar igualdad o diferencia entre dos valores

Operadores de comparación (Ejercicio: let x = 5;)

==	igualdad en valor	x == 8	false
		x == 5	true
		x == "5"	true !
===	igualdad en valor y tipo	x === 5	true
		x === "5"	false
!=	desigualdad en valor	x != 8	true
		x != "5"	false !
!==	desigualdad en valor o tipo	x !== 5	false
		x !== "5"	true
>	mayor que	8 > x	true
<	menor que	x < 8	true
>=	mayor o igual que	5 >= x	true

Condicionales

- Se usan para ejecutar instrucciones basado en ciertas condiciones
- Alteran el flujo de ejecución
- El más común es el `if`

example.js

```
let isHappy = true;


// The condition is an expression that returns
true or false
if (isHappy) {
 console.log("I am happy");
}
console.log("End of script");

// output ?
```

Condicionales

- Se usan para ejecutar instrucciones basado en ciertas condiciones
- Alteran el flujo de ejecución
- El más común es el `if`

example.js

```
let isHappy = true;


// The condition is an expression that returns
true or false
if (isHappy) {
 console.log("I am happy");
}
console.log("End of script");

// output
I am happy
End of script
```

Condicionales

- Se usan para ejecutar instrucciones basado en ciertas condiciones
- Alteran el flujo de ejecución
- El más común es el `if`

example.js

```
// boolean expression (returns a boolean
value)
if ((1 + 1) === 2) {
 // do something
}
// boolean expression (returns a boolean
value)
if (10 < 5) {
 // do something
}
```

Ejercicios

Ejercicio B (10 mins)

1. En node REPL, cual es el resultado de aplicar el operador `typeof` a `true` o `false`?
2. Corregir la siguiente función para que devuelva el string `"You've given me a bool, thanks!"` al recibir un booleano como parámetro

```
function boolChecker(bool) {
 if (typeof bool === ?) {
   return "You've given me a bool, thanks!";
 }

 return "No bool, not cool.";
}

boolChecker(true);
boolChecker(2);

typeof 2;
typeof var2;
```

Condicionales

- La instrucción if ejecuta un bloque de instrucciones si la condición **se cumple**
- La instrucción else ejecuta un bloque de instrucciones alternativo si la condición **no se cumple**

example.js

```
let isHappy = true;


// The condition is met
if (isHappy) {
 console.log("I am happy :)");
} else { // the condition is not met
 console.log("I am not happy :(");
}
console.log("End of script");

// output?
```

Condicionales

- La instrucción `if` ejecuta un bloque de instrucciones si la condición **se cumple**
- La instrucción `else` ejecuta un bloque de instrucciones alternativo si la condición **no se cumple**

example.js

```
let isHappy = true;


// The condition is met
if (isHappy) {
 console.log("I am happy :)");
} else { // the condition is not met
 console.log("I am not happy :(");
}
console.log("End of script");

// output
I am happy :)
End of script
```

Condicionales

- También hay una instrucción `else if` para manejar múltiple condiciones

example.js

```
let age = 24;

if (age >= 65) {
 console.log("John is an old guy.");
} else if (age < 18) {
 console.log("John is a young boy.");
} else {
 console.log("John is an adult.");
}
```

Ejercicios

Ejercicio C (5 mins)

1. Puedes explicar qué hace esta función línea por línea? Qué ocurre si pasas un string como parámetro?

```
function numberChecker(num) {
 if (num > 20) {
   return `${num} is greater than 20`;
 } else if (num !== 20) {
   return `${num} is equal to 20`;
 } else if (num < 20) {
   return `${num} is less than 20`;
 } else if (num < 0) {
   return `${num} is less than 20`;
 } else {
   return `${num} isn't even a number :(`;
 }
}
```

Ejercicios

Ejercicio D (10 mins)

Crea una función que retorne un mensaje dependiendo de tu estado de ánimo! La función debe:

- Tener un parámetro de entrada
- Retornar "Good job, you're doing great!" si el parámetro es igual a "happy"
- Retornar "Every cloud has a silver lining" si el parámetro es igual a "sad"
- Retornar "Beep beep boop" si el parámetro es un número
- Retornar "I'm sorry, I'm still learning about feelings!" si el parámetro es cualquier otra cosa

Operadores Lógicos

- Se usan usualmente con valores booleanos
- Permiten escribir expresiones que evalúan a un valor booleano

example.js

```
// AND operator
&&

// OR operator
||

// NOT operator
!
```

Expresiones Lógicas

- Combinación de expresiones unidas por un operador lógico
- Se evalúan de izquierda a derecha
- Tienen evaluación *corto-circuito*

example.js

```
// AND operator
expr1 && expr2

// OR operator
expr1 || expr2

// NOT operator
!expr
```

Operadores lógicos

Operadores lógicos:

- OR (||)
- AND (&&)
- NOT (!)

example.js

```
let num = 10;

function satisfiesRequirements(num) {
 if (num > 3 && num < 10) {
   return true;
 } else if (1 || 12) {
   return true;
 }
 return false;
}
satisfiesRequirements(num); // output ?
```

Operadores lógicos

Operadores lógicos:

- OR (||)
- AND (&&)
- NOT (!)

example.js

```
let num = 10;

function satisfiesRequirements(num) {
 if (num > 3 && num < 10) {
   return true;
 } else if (1 || 12) {
   return true;
 }
 return false;
}
satisfiesRequirements(num); // output: true
```

Expresiones Lógicas

Evaluación corto-circuito:

- Para expresiones del estilo `false && expr` se hace corto-circuito y se evalúa a `false`.
- Para expresiones del estilo `true || expr` se hace corto-circuito y se evalúa a `true`.

example.js

```
let num = 10;

function satisfiesRequirements(num) {
 if (n > 3 || (n < 10 && n > 8)) {
   return true;
 }

 return false;
}
satisfiesRequirements(5);
```

Expresiones Lógicas

OR:

- Si al menos una de las expresiones evalúa a `true`, devuelve `true`

(Recuerda que todo lo que tenga un valor evalúa a `true`)

example.js

```
let o1 = true || true; // ?
let o2 = false || true; // ?
let o3 = true || false; // ?
let o4 = false || 3 === 4; // ?
let o5 = "Cat" || "Dog"; // ?
let o6 = false || "Cat"; // ?
let o7 = "Cat" || false; // ?
```

Expresiones Lógicas

OR:

- Si al menos una de las expresiones evalúa a `true`, devuelve `true`

(Recuerda que todo lo que tenga un valor evalúa a `true`)

example.js

```
let o1 = true || true; // t || t returns true
let o2 = false || true; // f || t returns true
let o3 = true || false; // t || f returns true
let o4 = false || 3 === 4; // f || f returns
false
let o5 = "Cat" || "Dog"; // t || t returns Cat
let o6 = false || "Cat"; // f || t returns Cat
let o7 = "Cat" || false; // t || f returns Cat
```

Expresiones Lógicas

AND:

- Si al menos una de las expresiones evalúa a `false`, devuelve `false`

(Recuerda que todo lo **no** que tenga un valor evalúa a `false`)

example.js

```
let a1 = true && true; // ?
let a2 = true && false; // ?
let a3 = false && true; // ?
let a4 = false && 3 === 4; // ?
let a5 = "Cat" && "Dog"; // ?
let a6 = false && "Cat"; // ?
let a7 = "Cat" && false; // ?
```

Expresiones Lógicas

AND:

- Si al menos una de las expresiones evalúa a `false`, devuelve `false`

(Recuerda que todo lo **no** que tenga un valor evalúa a `false`)

example.js

```
let a1 = true && true; // t && t returns true
let a2 = true && false; // t && f returns
false
let a3 = false && true; // f && t returns
false
let a4 = false && 3 === 4; // f && f returns
false
let a5 = "Cat" && "Dog"; // t && t returns Dog
let a6 = false && "Cat"; // f && t returns
false
let a7 = "Cat" && false; // t && f returns
false
```

Expresiones Lógicas

NOT:

- Negación

example.js

```
let n1 = !true; // !t returns false
let n2 = !false; // !f returns true
let n3 = !"Cat"; // !t returns false
```

Ejercicios

Ejercicio E (5 mins)

Escribe las siguientes expresiones en node REPL y ve el resultado. Encuentras algún valor inesperado?

- `let num = 10`
- `num > 5 && num < 15`
- `num < 10 || num === 10`
- `false || true`
- `!true`
- `let greaterThan5 = num > 5`
- `!greaterThan5`
- `!(num === 10)`

Loops

example.js

```
console.log("The count is 1");
console.log("The count is 2");
console.log("The count is 3");
console.log("The count is 4");
console.log("The count is 5");
// ...
console.log("The count is 100");
```

Loops

Loops (iteradores):

Permiten ejecutar el mismo código múltiples veces con valores diferentes

example.js

```
console.log("The count is 1");
console.log("The count is 2");
console.log("The count is 3");
console.log("The count is 4");
console.log("The count is 5");
// ...
console.log("The count is 100");
```

Loops

Loops (iteradores):

Permiten ejecutar el mismo código múltiples veces con valores diferentes.

Cada vuelta se llama *iteración*

example.js

```
console.log("The count is 1");
console.log("The count is 2");
console.log("The count is 3");
console.log("The count is 4");
console.log("The count is 5");
// ...
console.log("The count is 100");
```

Loops

while loops:

- Se evalúa una condición de parada
- Se ejecuta el código dentro del bloque de instrucciones si la condición **se cumple**
- La condición se evalúa en cada iteración

example.js

```
let count = 1; // counter

while (count <= 100) {
 console.log("The count is: " + count);
 count += 1; // This is the same as count =
count + 1
}
….

The count is: 1
The count is: 2
The count is: 3
```

Ejercicios

Ejercicio G (10 mins)

1. Imprime la cuenta regresiva del Apolo 11, usa el mensaje dado como última línea. Comienza desde 8 hasta 0!

```
const apolloCountdownMessage = "all engine
running… LIFT-OFF!";
let countdown = 8;
console.log(apolloCountdownMessage);


// Expected output
8
7
6
5
4
3
2
1
0
all engine running... LIFT-OFF!
```

Loops

for loops:

- Similares a los while loops
- La sintaxis incluye:
  - Inicialización de la variable contador
  - Evaluación de la condición de parada
  - Incremento/decremento del contador en cierta cantidad

example.js

```
for (initialization; condition; final-expression)
{ ... }
```

Checks condition

```
for (let i=0; i < 20; i = i+1) {...}
```

Start counter at 0

On each loop increment the counter by 1

Loops

for loop:

- Inicialización de la variable contador (una vez antes de iniciar las iteraciones)
- Evaluar la condición de parada (cada iteración)
- Incremento/decremento de la variable contador (cada iteración)

example.js

```
for (let i = 0; i < 100; i=i++) {
 console.log("The count is: " + counter);
}

let i = 0;
while (i < 100) {
 …
 i = i + 1;
}

// i++ is a shortcut for i = i + 1
// sames as i += 1
```

Ejercicios

Ejercicio H (10 mins)

Calcular el exponencial de los números pares del 5 al 20 usando un for loop y las funciones provistas.

```
function exponential(number) {
 return number * number;
}
function isEven(number) {
 return number % 2 === 0;
}
// Expected output
The exponential of 6 is 36
The exponential of 8 is 64
The exponential of 10 is 100
The exponential of 12 is 144
The exponential of 14 is 196
The exponential of 16 is 256
The exponential of 18 is 324
```

Arreglos

example.js

```
const mentor1 = "Daniel";
const mentor2 = "Irina";
const mentor3 = "Rares";
```

Arreglos

- Variable para almacenar múltiples valores
- **Estructura de datos** que almacena una lista de valores
- Puede almacenar cualquier tipo de dato
- Pueden almacenar múltiples valores de diferentes tipos de datos a la vez

example.js

```
const mentor1 = "Daniel";
const mentor2 = "Irina";
const mentor3 = "Rares";

const mentors = ["Daniel", "Irina", "Rares"];
```

Arreglos

- Variable para almacenar múltiples valores
- **Estructura de datos** que almacena una lista de valores
- Puede almacenar cualquier tipo de dato
- Pueden almacenar múltiples valores de diferentes tipos de datos a la vez

example.js

```
const mentor1 = "Daniel";
const mentor2 = "Irina";
const mentor3 = "Rares";

const mentors = ["Daniel", "Irina", "Rares"];
```

Arreglos

- Variable para almacenar múltiples valores
- **Estructura de datos** que almacena una lista de valores
- Puede almacenar cualquier tipo de dato
- Pueden almacenar múltiples valores de diferentes tipos de datos a la vez

example.js

```
const mentor1 = "Daniel";
const mentor2 = "Irina";
const mentor3 = "Rares";

const mentors = ["Daniel", "Irina", "Rares"];

const testScores = [16, 49, 85];
const grades = ["F", "D", "A"];
const greetings = ["Hello, how are you?", "Hi! Nice to meet you!"];
const mix = [true, "ABC", 36]; // try to avoid arrays of different data types
```

Arreglos

Cómo acceder a los valores de un arreglo?

- Acceder (**indexar**) un valor: escribir el nombre de la variable seguido por corchetes ([]) y la posición del elemento en el arreglo
- Los arreglos son **indexados** empezando desde el **0** (no desde el 1)

example.js

```
const students = ["Ahmed", "Maria", "Atanas",
"Nahidul", "Jack"];

students[0]; // "Ahmed"
students[2]; // "Atanas"
students[3]; // "Nahidul"
students[1] // “Maria”
```

Arreglos

Cómo modificar elementos en un arreglo?

- Escribir el nombre de la variable seguido de corchetes ([]) con la posición del elemento a modificar, seguido del símbolo de asignación (=) y el nuevo valor

example.js

```
let students = ["Ahmed", "Maria", "Atanas",
"Nahidul", "Jack"];

students[2] = "Bianca";

console.log(students); // ["Ahmed", "Maria",
"Bianca", "Nahidul", "Jack"]

let jaime = "Jaime";

jaime = 23;
```

Ejercicios

Ejercicio I (5 mins)

En el node REPL, ingresa el siguiente arreglo:

```
const fruits = ['banana', 'apple', 'strawberry', 'kiwi', 'fig', 'orange'];
```

Ahora, usando los índices correctos, obtén los siguientes valores:

- strawberry
- kiwi
- orange
- banana

Luego, reemplaza 'apple' con 'raspberry', y reemplaza 'fig' con 'pineapple'.

Ejercicios

Ejercicio J (5 mins)

Completa la siguiente función tal que si la segunda posición del arreglo contiene el nombre "Amy" retorne `"Second index matched!"`

```
function secondMatchesAmy(array) {
 if (?) {
   return "Second index matched!";
 }
 return "Second index not matched";
}
let names = ["Alex", "Amara", "Carlos"];
let names2 = ["Ali", "Amy", "Naresh"];

const result = secondMatchesAmy(names);
console.log(result)
```

Arreglos

Iterando un arreglo

- Iterar sobre los valores de un arreglo usando loops

example.js

```
const daysOfWeek = [
 "Monday",
 "Tuesday",
 "Wednesday",
 "Thursday",
 "Friday",
 "Saturday",
 "Sunday",
];

for (let i = 0; i < daysOfWeek.length; i++) {
 const dayMessage = "day is: " +
daysOfWeek[i];
 const indexMessage = "index is: " + i;
 console.log(indexMessage, dayMessage);
}
```

Ejercicios

Ejercicio K (10 mins)

Escribe una función que reciba un arreglo de estudiantes. En la función, usa un for loop para iterar sobre el arreglo e imprimir el nombre de cada estudiante.

Strings - arreglos

Strings

- Cadenas de caracteres (incluyendo los espacios)
- Los strings comparten algunas propiedades de los arreglos
  - Indexación
  - length
  - Son iterables

example.js

```
const greeting = "Hola, I'm Alexandra";

for (let i = 0; i < greeting.length; i++) {
 console.log(greeting[i]);
}

// output?
```

EJERCICIO

Ejercicio F (modificado) (15 mins)

Escribe una función que revise si un nombre de usuario tiene un formato y un tipo de usuario adecuado.La función debe:

- Recibir dos parámetros: uno para el nombre de usuario y el otro para el tipo de usuario
- Si el nombre de usuario empieza por vocal y tiene una longitud de 5 a 10 caracteres, debe retornar `"Username valid"`; de lo contrario, debe retornar `"Username invalid"`
- Si el tipo de usuario es un `admin` o un `manager`, debe retornar `"Username valid` sin importar el nombre de usuario

EJERCICIO

Ejercicio F (15 mins)

Escribe una función que revise si un nombre de usuario tiene un formato y un tipo de usuario adecuado.La función debe:

- Recibir dos parámetros: uno para el nombre de usuario y el otro para el tipo de usuario
- Si el nombre de usuario empieza por mayúscula y tiene una longitud de 5 a 10 caracteres, debe retornar `"Username valid"`; de lo contrario, debe retornar `"Username invalid"`
- Si el tipo de usuario es un `admin` o un `manager`, debe retornar `"Username valid` sin importar el nombre de usuario

Nota: para transformar una letra a mayúscula o minúscula puedes usar las funciones **toLowerCase**() y **toUpperCase**()