Introducción a la programación - Fundamentos

¿Programa?

Podemos definir **programa** como un conjunto de instrucciones que las computadoras interpretan para resolver un problema. Para escribirlas, usamos **lenguajes formales**: ¡los lenguajes de programación!

Al igual que en un lenguaje natural, como el Español o el Inglés, tienen una determinada sintaxis. La diferencia está en que los lenguajes formales tienen **sintaxis rígida**, ya que van a ser interpretados por una máquina. Si bien en el Español podemos obviar palabras, signos de puntuación, o tener errores de ortografía (y probablemente nuestro interlocutor nos entienda igual), una computadora no será tan inteligente y nos mostrará errores si por ejemplo nos olvidamos de un punto y coma.

Hay muchas formas de construir software, a las que llamamos **paradigmas**. El que vamos a ver en este curso por su simpleza es el de la **programación imperativa**. En él, diseñamos nuestros programas como **algoritmos**, es decir, secuencias de pasos para cumplir una determinada función.

¿Algoritmo?

Eso, una **secuencia de pasos** para cumplir una función. El intérprete del lenguaje que utilicemos ejecutará esos pasos de forma ordenada para dar el resultado que nosotros queremos. Siempre la estrategia es **dividir el problema** en pasos.

Por ejemplo, si quisiéramos hacer un algoritmo para calentar algo en un microondas durante 60 segundos, escribiríamos algo como:

```
comida = agarrarComida();
ponerEnMicroondas(comida);
encender(60);
```

Pero a su vez, la tarea de **ponerEnMicroondas** se divide en otras tareas:

```
function ponerEnMicroondas(comida) {
abrirPuerta();
colocarComida(comida);
cerrarPuerta();}
```

De esta forma podemos crear pequeñas porciones de código (**funciones**) que nos permiten **abstraer** ciertas tareas, para poder reutilizarlas posteriormente si lo necesitáramos.

No te preocupes si no entiendes la sintaxis. Proviene de **JavaScript**, el lenguaje que utilizaremos. Por ahora podemos ver que:

- El ; del final de cada línea separa **sentencias** (pasos).

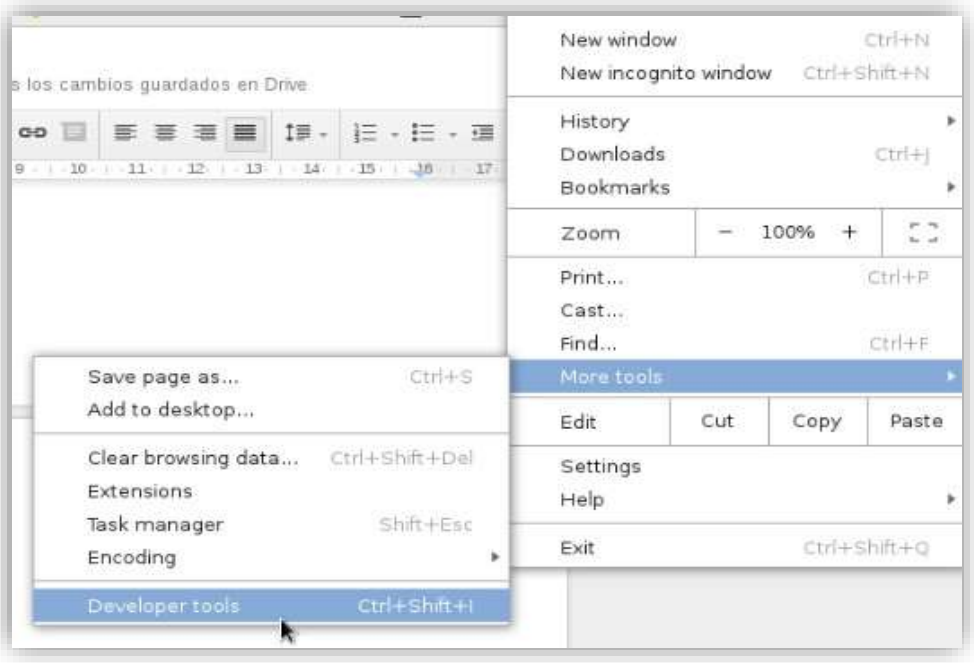
- Se ejecuta una sentencia después de la otra, de arriba hacia abajo.

- Podemos **invocar** funciones escribiendo su nombre, y entre paréntesis algunos valores que necesite para operar.

- Cuando invocamos a funciones, se pasa a ejecutar el código de dicha función y luego se continúa del lugar donde estábamos (ej: en ponerEnMicroondas(comida) primero se ejecutan las 3 sentencias de ponerEnMicroondasy solo después de eso se invoca a encender).

Guía

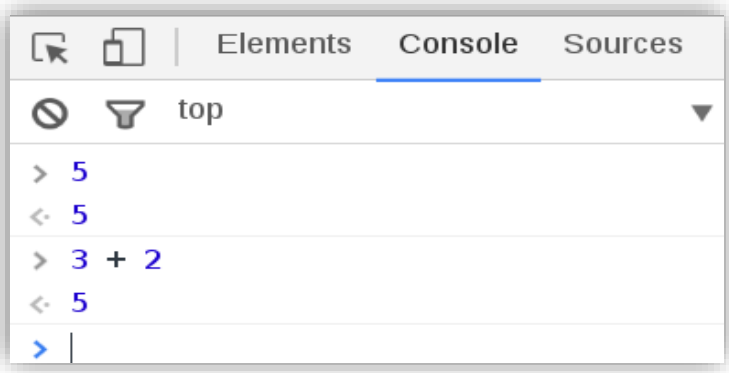
En *Google Chrome*, podemos acceder a la **consola de desarrolladores**:



Ahí, podemos **probar porciones de código** y **ver sus resultados** inmediatamente.

Expresiones

Todos los lenguajes están basados en **expresiones**, o sea, **algo a lo cual le podemos consultar su valor**. Una expresión simple por ejemplo es un número:



Como vemos, cuando escribimos una expresión en la consola y tocamos Enter, **nos devuelve su valor**. Tanto `5`como `3 + 2`son expresiones que el intérprete tuvo que calcular.

Entonces decimos que una expresión puede ser:

- Un valor
- El resultado de aplicar **operaciones** a ciertos valores

Mira el **Anexo 1** para ver algunos operadores comunes que nos da JavaScript.

Valores

Un valor **es de un tipo** determinado. Los valores que vimos hasta ahora son valores numéricos, pero hay varios tipos. Otro muy común son las **cadenas** (o "strings"), con los que podemos representar cualquier porción de texto.

Strings

Para crear un **string**, debemos escribir el texto entre comillas (dobles o simples). Estos tienen varias operaciones incluídas:

```
-> "hola" + " " + "mundo" // el + aplicado a strings los concatena
<- "hola mundo"

-> "hola".length // cantidad de letras
<- 4

-> "hola".charAt(0) // primer caracter
<- "h"

-> "hola".endsWith("ola") // ¿termina con "ola"?
<- true
```

Descubrimientos hasta ahora:

- Los valores tienen funciones para operar con ellos.
- Podemos escribir comentarios (que serán ignorados) con // y un texto.
- Apareció un tipo de dato nuevo, ¡el **booleano**!, que puede valer trueo false.

Mirá el **Anexo 2** para ver los tipos de datos disponibles en JavaScript.

Variables

De nada sirve escribir tantas expresiones si no podemos almacenarlas en ningún lado. Para esto, tenemos las **variables**. Son **contenedores** que pueden **guardar un valor** en memoria.

Se definen como:

```
var cantidadDePersonas = 40;
```

O sea, varpara decir que es una variable, un nombre para identificarla, y un **valor inicial**
seguido del **operador igual** (=).

- Los nombres se suelen escribir en *lowerCamelCase* y pueden tener letras, números, y guiones bajos.
- El valor inicial puede ser cualquier expresión.

Al "operador igual" se lo llama **asignación destructiva**, porque **pisa el valor anterior que tenía la variable**. Es decir, que si luego de crear la variable actual hacemos:

```
cantidadDePersonas = 8;
```

Estamos **destruyendo** el valor anterior (*40*) y asignando uno nuevo (*8*) en su lugar. Noten que esta vez no escribimos `var` porque la variable ya ha sido creada.

Podemos usar variables para almacenar resultados que provienen de fuentes externas, como el usuario. Por ejemplo:

```
var nombreIngresado = prompt("Ingresá tu nombre");
// (la función prompt nos da un texto que ingrese el usuario)
alert("Hola " + nombreIngresado);
// (la función alert muestra una alerta con el texto que nosotros le pasemos)
```

Funciones

Las funciones son **otro tipo de dato más**. Pueden haber valores de tipo función y por lo tanto también son expresiones.

Ésta es una función que calcula el promedio de dos números:

```
var promedioEntre = function(unNumero, otroNumero) {

  return (unNumero + otroNumero) / 2;
}
```

A `unNumero` y `otroNumero` se los llama **parámetros**, y es lo que necesita nuestra función para trabajar.

Como vemos, la función **retorna** el resultado de la operación.

¿Qué valor tendrá esta expresión?

```
promedioEntre(4, 8) * promedioEntre(1, 5)
```

¡Probala!

El valor de retorno

No es obligatorio que una función **retorne un valor**. En caso de no hacerlo, el valor producido por invocarla es undefined. Este valor junto a null se usan para representar la no presencia de valor.

Las funciones que no retornan nada deberían tener **efecto** (es decir que causan cambios en alguna entidad externa). Las funciones que sí retornan un valor, por lo general no tienen efecto. Uno debería nombrar a las funciones de forma tal que se note en cuál de estos grupos entra. Por ejemplo:

- `obtenerDeuda() // función sin efecto`
- `cancelarDeuda() // función con efecto`
- `cancelarDeudaSiExpiró() // función con efecto (condicional)`

Verificando condiciones: el if

El ifnos permite **verificar** si se cumple una **condición**, y en ese caso hacer algo al respecto.

Su sintaxis es más o menos así:

```
if (valorOExpresiónBooleana) {
  // hacer algo
}
```

Por ejemplo, podemos mostrar un cartel si 4 es mayor a 2:

```
if (4 > 2) {
  alert("OMG, ¡4 es mayor a 2!");
}
```

Sí, sabemos que no tiene sentido hacer un if con una condición que su valor es siempre true.
Algo más interesante, por ejemplo, sería ver si el nombre que ingresó el usuario es

"Carlos", y tomar una decisión al respecto.

```
var nombreDelUsuario = prompt("¿Cuál es tu nombre?"); if

(nombreDelUsuario === "Carlos") {
  alert("Hola Carlos, bienvenido al sistema.";
  arrancarTodo();
} else {
  alert("Ehmmm, vos no sos."); salir()
}
```

Como se ve, opcionalmente podemos agregar un bloque `else` para cuando la condición no se cumple.

- El operador **===** ("igual a") determina si dos valores son iguales y devuelve un booleano. No confundir con el de la asignación destructiva (**=**). Es el operador opuesto a **!==** ("distinto a").

Operaciones repetitivas: El while

Para hacer operaciones repetitivas tenemos el ciclo `while`. Este permite ejecutar una operación **mientras se cumpla una condición**.

Este es un código de ejemplo para sumar todos los números que ingrese un usuario;

```
var pedirNumero = function() {
  var input = prompt("Ingrese un número o nada para dejar de sumar");
  return parseInt(input); // retornamos el valor convertido a número
}

var suma = 0;
var numeroIngresado = pedirNumero();
while (numeroIngresado) { // mientras exista un numeroIngresado suma =
  suma + numeroIngresado; // acumulamos en suma el total
  numeroIngresado = pedirNumero(); // pedimos otro número y lo guardamos}

alert("El resultado de la suma es " + suma); // mostramos el resultado
```

Manejando muchos valores: Las colecciones

Las listas / colecciones / arrays son **conjuntos de varios valores**, comúnmente del mismo tipo. Podemos crear listas de la siguiente forma:

```
var numeros = [ 1, 2, 3 ];
```

Es una herramienta muy poderosa y que vamos a usar cada vez que necesitemos mostrar un grupo de cosas (ej: una lista de usuarios en pantalla, de facturas, de lo que sea).

Podemos acceder a los elementos de la lista por posición. Es decir, que si queremos acceder al segundo número debemos escribir la expresión `numeros[1]`.

Al numero que va entre corchetes se lo llama **índice**. Los índices **siempre están basados en cero**. Es decir, que el segundo elemento es el 1, el tercero es el 2, y etc...

A la tarea de **hacer algo con cada elemento** de una lista, se la llama **iterar**. Un ejemplo de cómo hacer algo con cada elemento de una lista:

```
[ "pedro", "juan", "jorge" ].forEach(function(nombre) {
    alert("Iterando con " + nombre);
});
```

Las listas no son fijas, sino que pueden mutar todo el tiempo según se deseen agregar cosas. Por ejemplo, numeros.push(4); agregaría un cuarto número a la lista inicial.

Anexos

Anexo 1:

Operadores

+ Suma	`2 + 3 // 10`
- Resta	`5 - 3 // 2`
-Menos	`-(2 + 5) // -7`
* Multiplicación	`2 * 4 // 8`
/ División	`7 / 2 // 3.5`
% Resto en la división	`40 % 30 // 10`

Lógicos

- === Igual `4 === 4 // true`
- !== Distinto `4 !== 4 // false`
- > Mayor `5 > 2 // true`
- >= **Mayor oigual** `5 >= 5 // true`
- < Menor `8 < 2 // false`
- <= **Menor oigual** `8 <= 1 // false`
- ! Negación*(unario)* `!(2 > 5) // true`
- && AND `(2 > 1) && (2 === 3) // false`
- || OR `(2 > 1) || (2 === 3) // true`

Anexo 2: Tipos de datos

Número (`Number`): Números con o sin decimales. *Ejemplos:*

```
4
-2.4
0
```

Texto (`String`): Cadenas de texto o caracteres. *Ejemplos:*

```
"hola"
"a"
```

Booleano (`Boolean`): Verdadero o falso. *Ejemplos:*

```
true false
```

Comportamiento (`Function`): Porción de código ejecutable. *Ejemplos:*

```
function saludar() {
    alert("Hola mundo");
}
function masDos(valor) {
    return valor + 2;
}
```

Lista (`Array`): Conjunto ordenado de varios valores. *Ejemplos:*

```
[1, 2, 3]
["hola", true, 3.4, function() {}]
// ^- aunque en general no tiene sentido agrupar valores de tipos tan distintos
```

Objeto (`Object`): Agrupación lógica de cualquiera de los valores anteriores. *Ejemplos:*

```
{
    edad: 24,
    nombre: "Rodri",
  tieneRulos: true,

  hijos: [],
  saludar: function() {
      alert("Hola! ¿cómo va?");

  },
  direccion: {
      calle: "Calle falsa",
        altura: 123,
      ciudad, "Sarasa",
      provincia: "..."
  }
}
```

Dos buenas prácticas que deberían seguirse.

Para finalizar, me gustaría agregar dos cosas que me parecen fundamentales a la hora de programar:

- Siempre poner **nombres descriptivos** a las variables. La persona que lo lee tiene que entender de qué tipo es y para qué se usa. Evitar las abreviaturas, que solo confunden más y generan ambigüedades.

- **No duplicar lógica**. Escribir la lógica de lo que se desea hacer **una sola vez** y usar funciones para abstraer ese comportamiento y poder **reutilizarlo** en otros lugares.