

Plan 2019



Material del Curso



Programación 1

Cátedra de Programación

Universidad ORT Uruguay

V. 2024 – Abril 2024 V19

Índice

1	Introducción	5
1.1	Prólogo de este material	5
1.2	Objetivos del curso	5
1.3	Logros a obtener	5
1.4	Resumen de temas	5
1.5	Ingeniero, Licenciado, Programador	6
1.5.1	Perfil del Licenciado en Sistemas	6
1.5.2	Perfil del Ingeniero en Sistemas	7
1.5.3	Perfil de Programador	7
1.6	Definiciones generales	8
1.6.1	Sistema	8
1.6.2	Hardware	8
1.6.3	Software	8
1.6.4	Programar	8
1.6.5	Diseñar	8
1.6.6	Sistema Operativo	9
1.7	Ciclo de vida del Software	9
1.8	Lenguajes de programación	9
1.9	Nociones de Pensamiento Computacional	10
1.9.1	Ejemplos de Abstracción	12
1.9.2	Ejemplos de Descomposición	12
1.9.3	Ejemplos de Reconocimiento de Patrones	12
1.9.4	Ejemplos de Algoritmo	12
1.10	Algoritmo y propiedades	12
1.11	Estructuras de control	13
2	Variables e Introducción a JS	17
2.1	Introducción a variables	17
2.2	Seudocódigo y Ejemplos	17
2.2.1	Cálculo del IVA	17
2.2.2	Suma de 10 números	18
2.3	Asignación	19
2.4	Corrida a mano	20
2.5	Más ejemplos	21
2.5.1	Producto de 20 números	21
2.5.2	Loop	21
2.5.3	Valores repetidos	21
2.5.4	Promedio	22
2.6	Intercambio de 2 variables	22
2.7	Programa	23
2.8	Introducción a JavaScript	23
2.9	Expresiones aritméticas	24
2.10	Variables booleanas y expresiones lógicas	24
2.10.1	Variables booleanas	24
2.10.2	Operadores relacionales	24
2.10.3	Operadores lógicos	24
2.10.4	Expresiones lógicas	25
2.11	JS: Números y operadores	26
2.11.1	Números en JS	26
2.11.2	Operadores en JS	27
2.12	Uso de variables booleanas	28

3	Codificación	30
3.1	Strings en JS	30
3.2	Sobrecarga de operadores	30
3.3	Variables en JS	31
3.4	Console	32
3.5	Sentencias en JS	33
3.5.1	Ingreso de datos y mostrar datos	33
3.5.2	Suma de 10 números en JS: for	35
3.5.3	Suma de 10 números: while	35
3.5.4	Valores repetidos: if	37
3.5.5	Switch	38
3.6	Ejemplos	38
3.6.1	Máximo	38
3.6.2	Ecuación de 2do grado	41
4	Funciones	42
4.1	Introducción a funciones	42
4.2	Ejemplos de funciones	43
4.2.1	Cálculo del área	43
4.2.2	Perímetro	43
4.2.3	Número par	43
4.2.4	Puede votar	44
4.2.5	Likes	44
4.2.6	Números ordenados	44
4.2.7	Signo	45
5	Más de String	46
5.1	Profundización de String	46
5.2	Ejercicios básicos	47
5.2.1	Armar cartel	47
5.2.2	Armar cartel largo máximo	47
5.2.3	Contar cantidad de una letra dada de una frase	47
5.3	Otros métodos útiles de String: slice, substring, substr	48
5.4	Más ejercicios de String	48
5.4.1	Dar vuelta palabra	48
5.4.2	Indicar si es fin de semana	49
5.4.3	Palíndromo	49
5.4.4	Ocurrencias de primera letra	49
6	Arrays	50
6.1	Introducción a Array	50
6.2	Carga y recorrida de array	52
6.2.1	Carga	52
6.2.2	Recorrida de array	52
6.2.2.1	Recorrida con for "tradicional"	52
6.2.2.2	Recorrida con for in	52
6.2.2.3	Recorrida con for of	53
6.3	Ejercicios básicos de array	53
6.3.1	Suma	53
6.3.2	Promedio	53
6.3.3	Máximo	54
6.3.4	Desviación estándar	54
6.4	Búsqueda en array	55
6.4.1	Búsqueda manual	55
6.4.2	Otras formas de buscar	56

6.5	Ordenación de array	56
6.5.1	Ordenación por selección	56
6.5.2	Ordenación usando método Sort	57
6.6	Más ejercicios de array	58
6.6.1	Largo de palabras	58
6.6.2	Intercalar	58
6.6.3	Eliminar	58
6.6.4	Ejercicio: Comparación	58
6.6.5	Más cercano	59
7	Introducción a Web, HTML y CSS	60
7.1	Introducción a Web	60
7.1.1	Resumen de definiciones	60
7.1.2	Arquitectura Web	61
7.1.3	Otros conceptos relevantes: Accesibilidad e Internacionalización.	62
7.2	Introducción a HTML	63
7.2.1	Primera página de ejemplo	63
7.2.2	Análisis detallado de la página	64
7.2.3	Encabezados y párrafos	64
7.2.4	Listas	65
7.2.5	Tablas	66
7.2.6	Links	68
7.2.7	Otros elementos: caja de texto, combo, botón	69
7.2.8	Tags semánticos	70
7.3	CSS básico y vinculación con HTML	71
7.3.1	¿Cómo funciona CSS?	71
7.3.2	¿Dónde ubicarlo?	72
7.3.3	Ejemplo Introductorio de HTML y CSS	72
8	HTML y JS	75
8.1	Nociones DOM	75
8.2	Incorporando la interfaz: vínculo HTML y JS	76
8.3	Ejemplo: Ingresar una frase y pasarla a mayúsculas	76
8.3.1	Parte interfaz	76
8.3.2	Parte lógica	77
8.3.3	Ampliación: mostrar los datos en lista	79
8.3.4	Ampliación: mostrar los datos en una tabla	79
8.4	Ejemplos básicos de vinculación HTML-JS	80
8.4.1	Votar. Ingreso de dato y presentación en párrafo.	80
8.4.2	Múltiplos	81
8.4.3	Extensión: múltiplos (uso de radio button)	83
9	Depuración	86
9.1	Introducción a Prueba de Programas	86
9.2	Estrategias de Prueba de Programas	86
9.2.1	Caja Negra	86
9.2.2	Caja Blanca	87
9.3	Desarrollo y Prueba de métodos	87
9.4	Manejo de errores	87
9.4.1	¿Cómo se buscan?	88
9.5	Depuración en HTML	88
9.6	Errores comunes y consideraciones de JS	89
10	Más de HTML	92
10.1	HTML - Profundización	92
10.1.1	Caracteres especiales	92

10.2	Otras etiquetas en META	93
10.3	Comentarios y listas	93
10.3.1	Comentarios	93
10.3.2	Listas	93
10.4	Hipervínculos y énfasis	94
10.4.1	Hipervínculos	94
10.4.2	Énfasis	95
10.5	Abreviaturas, subíndices y superíndices	96
10.6	Imágenes	97
10.7	Tablas	97
10.8	Formularios (form, label, input, textarea)	98
11	Más de CSS	100
11.1	Profundización en Reglas y selectores	100
11.1.1	Reglas	100
11.1.2	Selectores simples	101
11.1.3	Selectores simples: class	101
11.1.4	Selectores simples: id	101
11.1.5	Seudoclases	102
11.1.6	Seudoelementos	102
11.2	Modelo de Caja	103
11.3	Ejemplo de Tags semánticos y estilos	104
12	Objetos en JS	107
12.1	Introducción	107
12.2	Elementos, propiedades, métodos y this	109
12.2.1	Elementos, propiedades y métodos	109
12.2.2	this	109
12.3	Introducción al concepto de clase	110
12.4	Método toString	111
12.5	Ejercicio: Contactos	111
12.5.1	Extensión: mayores	116
12.5.2	Extensión: ordenar	117
12.5.3	Extensión: apellido repetido	118
12.5.4	Extensión: eliminar un contacto	118
13	Clases y objetos	119
13.1	Asociación	119
13.2	Ejemplo integrador: Consultora	120
13.3	Referencias y objetos	123
13.4	Pasaje de parámetros	124
13.5	Comparación de objetos	124
13.6	Objeto window	125
13.7	Clase Object	126
13.8	Clase Number	126
13.9	Clase Date	127
14	JSON	129
14.1	Introducción a JSON	129
14.2	Local Storage	130

1 Introducción

1.1 Prólogo de este material

Este material de apoyo al curso de Programación I fue desarrollado por los docentes Dra. Ing. Inés Kereki y Lic. Sebastián Pesce. Se agradecen los comentarios y aportes de los docentes Diego Acosta y Lara, MBA, Alejandro Adorjan, MSc, Lic. Juan Pablo Calvo, Ing. Andrés de Sosa, Ing. Matías Núñez, Ing. Mauricio Repetto e Ing. Marcelo Rubino.

La notación utilizada es:

para pseudocódigo:

pseudocódigo

para código JS:

código JS

para código HTML:

código HTML

para CSS:

css

1.2 Objetivos del curso

El curso de Programación I tiene como objetivos:

- Desarrollar el pensamiento computacional (PC): abstracción, resolución de problemas, reconocimiento de patrones y algoritmia.
- Desarrollar las habilidades básicas de programación utilizando un lenguaje de amplio uso.

Implica:

- Conocer y aplicar las estrategias del PC
- Conocer, entender y aplicar los conceptos básicos de programación (variables, expresiones, estructuras de control, estructuras de datos como listas y arrays, funciones y procedimientos)
- Poder diseñar e implementar soluciones de problemas no triviales en un lenguaje de programación

1.3 Logros a obtener

- Desarrollar soluciones algorítmicas a problemas y representarlas como programas de computador
- Aplicar estrategias de implementación descendente y con diseño modular
- Experimentar con un lenguaje de alto nivel de amplio uso

1.4 Resumen de temas

Los principales temas del curso son:

- Noción de variable
- Estructuras de control

- Expresiones aritméticas y lógicas
- Ej. básicos: Máximo, Intercambio, Cálculo de raíces, manejo de String, etc.
- Procedimientos y funciones
- Estructuras: Arrays (ej: promedio, máximo, ordenación, búsqueda, recorridas, ...)
- Noción de objeto
- Nociones de estilo de programación
- Depuración
- Nociones web, HTML y CSS

Se implementarán estos temas en JavaScript.

1.5 Ingeniero, Licenciado, Programador

Ingeniero de Sistemas, Licenciado en Sistemas, Programador... muchas veces no está claro el rol de cada uno de estos profesionales.

1.5.1 Perfil del Licenciado en Sistemas

Los egresados de la Licenciatura en Sistemas cumplen un rol clave en las empresas y organizaciones ya que combinan una fuerte formación en tecnología con un conocimiento profundo de la gestión y el funcionamiento de las empresas. Suelen ocupar puestos de dirección de equipos, consultoría o toma de decisiones relacionada con la tecnología. Pueden tomar distintos roles en áreas de:

- Tecnología

Ejemplos:

Analizar, diseñar e implementar aplicaciones de software para móviles, internet y computación en la nube.

Seleccionar e integrar diferentes tecnologías tanto de hardware como de software.

Desempeñarse en proyectos de investigación e innovación tecnológica.

Implementar tecnologías para la adecuada y segura gestión de los datos, y la información organizacional.

- Administración y Gerencia:

Ejemplos:

Dirigir el área de tecnología informática.

Liderar equipos, gerenciar y valorar económicamente proyectos de sistemas de información.

Comprender los procesos empresariales y de toma de decisión en las organizaciones.

Contribuir a la creación de emprendimientos tecnológicos.

Aplicar habilidades de trato interpersonal, resolución de conflictos y pensamiento creativo.

- Sistemas de Información:

Ejemplos:

Participar en el diseño de procesos de negocio y su posterior informatización.

Identificar nuevas oportunidades y modelos de negocio facilitados por la tecnología.

Actuar como nexo entre el área de tecnología y los demás sectores de la organización.

Diseñar e implementar sistemas de información para apoyar la operativa y la estrategia empresarial.

Aplicar los principios de gestión de calidad y la seguridad de los sistemas de información.

Gestionar, integrar y asegurar los datos empresariales de acuerdo a su valor estratégico para la toma de decisiones

1.5.2 Perfil del Ingeniero en Sistemas

Son profesionales capaces de diseñar y desarrollar sistemas informáticos de gran porte y complejidad, que pueden desempeñarse como:

- creadores de software
- líderes de proyectos
- gerentes de sistemas
- consultores independientes

A través de la selección de materias electivas y de su proyecto final, los estudiantes pueden especializarse en áreas como:

- desarrollo y arquitectura de sistemas
- gestión del software
- seguridad informática
- desarrollo de videojuegos
- gerencia y negocios
- telemática
- sistemas de información

1.5.3 Perfil de Programador

Analicemos un posible caso de la vida real. Una empresa le plantea a un licenciado en Sistemas que desea “instalar computadoras para facturar”. El licenciado actúa de intermediario entre la empresa y el software, realiza entrevistas con las distintas personas relacionadas con este proceso, analiza y modela la realidad. Tiene como reglas ser objetivo, diplomático, verificar toda la información y asegurarse de tener la mayor cantidad de detalle. En función de este análisis realiza un diseño preliminar y elabora un informe o propuesta.

Si la empresa está de acuerdo con la propuesta, el licenciado describe y asigna los programas a los **programadores**, quienes escribirán los programas. Una vez que están prontos esos programas, se pone en marcha el sistema. Esto se puede hacer de varias formas, por ejemplo en paralelo, de golpe, o en etapas. En *paralelo* quiere decir que se pondrá en funcionamiento el nuevo sistema pero también se realizará simultáneamente el trabajo en la forma anterior. De *golpe* refiere a que se sustituye completamente el sistema manual por el nuevo sistema. En *etapas* quiere decir que, por ejemplo, en primera instancia se pondrá en marcha la facturación contado, en otra etapa posterior la facturación crédito.

1.6 Definiciones generales

1.6.1 Sistema

Hay varias definiciones posibles de sistema. Algunas de ellas tomadas de diccionarios indican:

- Conjunto u ordenación de cosas relacionadas de tal manera que forman una unidad o un todo.
- Conjunto de hechos, principios, reglas, etc. clasificadas y ordenadas de tal manera que muestran un plan lógico.
- Una forma establecida de hacer algo.

En computación tomaremos como definición de **sistema**: conjunto u ordenación de elementos organizados para llevar a cabo algún método, procedimiento o control mediante procesamiento de la información.

1.6.2 Hardware

Hardware refiere a todos los elementos físicos de una computadora. Se incluyen los discos, los circuitos, cables, tarjetas, monitor, teclado, mouse, etc.

Se puede clasificar en hardware fundamental y accesorio. El fundamental es el imprescindible para que funcione la computadora, en general se refiere a monitor, unidad de proceso y teclado. Dentro del accesorio se podría incluir por ejemplo una cámara de video o un escáner.

También se pueden clasificar como dispositivos de entrada (mouse, teclado, micrófono), de salida (impresora, monitor) o de entrada salida (pendrives, disco duro, etc.).

1.6.3 Software

Comprende los programas. Un programa puede ser de uso general, como un procesador de texto o una planilla electrónica o de uso específico, por ejemplo un sistema de cálculo de sueldos. Otra clasificación lo divide en software de base (incluye lo necesario para que funcione el hardware) y software de aplicación.

1.6.4 Programar

Es el arte de dar comandos a algo o alguien que pueden ser ejecutados después. Se puede especificar de distintas formas: botones, mapa, lista. Un comando es una orden para algo o alguien para realizar determinada acción. Puede ser, por ejemplo, oral ("levántate", "acomoda") o manual (presionar un botón).

1.6.5 Diseñar

Es el acto de organizar los comandos. Así como un novelista probablemente antes de escribir un libro estructura sobre qué temas tratará y, o, elabore un índice, en forma similar antes de programar se diseña, "se hacen los planos" del software.

1.6.6 Sistema Operativo

Es lo que vincula el hardware con el software. Ejemplos: Windows, Linux, MacOS, Android.

1.7 Ciclo de vida del Software

Si bien no es estricto, habitualmente el proceso que se sigue para desarrollar software es: análisis o relevamiento de requisitos, diseño, implementación, prueba y mantenimiento. Se presenta gráficamente en la Ilustración:



Durante el *análisis o especificación de requisitos* se trata de determinar claramente cuáles son los requisitos que debe satisfacer el sistema, qué características debe tener.

En el *diseño* se realizan los “planos” del sistema.

En la *implementación* se desarrolla efectivamente el software, se escriben los programas.

Durante la *prueba* se trata de eliminar los errores. Existen dos tipos de pruebas: alfa y beta. La prueba alfa habitualmente se hace dentro del equipo de desarrollo. Otro programador verifica los programas. En la prueba denominada beta se les pide colaboración a los clientes. Se les entrega en forma adelantada el software a cambio de que colaboren en su verificación.

Durante el *mantenimiento* se hacen los ajustes del software debido a errores no detectados antes, cambios en la realidad (por ejemplo, creación de un nuevo impuesto sobre la facturación) y, o ampliaciones al sistema original.

1.8 Lenguajes de programación

Nos detendremos en la escritura de programas. Inicialmente los programas se escribían en *lenguaje de máquina*. Este lenguaje es una combinación de 0 y 1 (el 0 representa que no hay corriente, el 1 que sí hay corriente) y sirvió para mostrar que era posible escribir programas, pero el nivel de abstracción de cada instrucción es muy bajo.

Por esta razón surgieron lenguajes del tipo “Assembler”, en el cual por ejemplo se escribe “LOAD X”, que correspondería a la instrucción de máquina 001100101011. Debido a que las máquinas únicamente entienden el lenguaje de 0 y 1, es necesario hacer la traducción a ese lenguaje. Este

proceso de traducción de lenguajes tipo “Assembler” a lenguaje de máquina se realiza por un programa llamado Ensamblador. En general, la correspondencia de sentencias es 1 a 1, una de lenguaje Assembler corresponde a una de máquina. Tanto el lenguaje de máquina como el lenguaje “Assembler” se denominan de *bajo nivel*, refiriéndose a bajo nivel de abstracción. Cada sentencia representa “poco”.

Posteriormente surgieron los lenguajes de *alto nivel* de abstracción, en los cuales una sentencia representa muchas de máquina. Sigue siendo necesario realizar la traducción de ese lenguaje al de máquina. Este proceso se puede hacer de dos formas: compilar e interpretar. En el caso de los lenguajes compilados, el compilador (programa que realiza esa traducción), toma todo el programa y lo traduce al lenguaje de máquina. Posteriormente se ejecutará. El intérprete toma cada línea del programa, la traduce y ejecuta, una a una. Es decisión de diseño del lenguaje si será interpretado o compilado.

Ejemplo:

(sentencia en alto nivel)		(en lenguaje de máquina)
SUM 1 TO TOTAL -----	compilador/intérprete ----	010001.....
11110101...		

Hay gran cantidad de lenguajes de programación: Java, Smalltalk, Pascal, Delphi, C, C++, C#, Modula, Ada, COBOL, Fortran, Basic, Visual Basic.NET (VB.NET), JavaScript, Python, etc. Cada uno de ellos está orientado a distintos fines. La elección de un lenguaje de programación depende de muchos factores, como por ejemplo disponibilidad, finalidad, costo y, o, facilidad.

JavaScript es un muy buen lenguaje para comenzar el aprendizaje de la programación. Así mismo, Java (que se utiliza en Programación II) es un excelente lenguaje para aplicar los conceptos de programación orientada a objetos.

1.9 Nociones de Pensamiento Computacional

Un concepto muy nombrado en estos días es el de “Pensamiento Computacional”. ¿De qué trata?

Veamos un par de ejemplos:

- Queremos salir con amigos. Cuando preparamos la salida, pensamos: ¿qué podemos hacer?, ¿qué opciones hay? ¿dónde podemos ir?, ¿qué hicimos otras veces?, ¿cuánto dinero tenemos disponible? y ¿cuánto nos cuestan las diferentes alternativas? Seguramente consideraremos también otros factores tales como si hará frío o calor y hasta cuánto tiempo tenemos. Luego organizamos efectivamente la salida, combinando la hora, lugar y demás detalles.
- Nuestra lámpara de mesa no funciona. ¿Qué hacemos? Analizamos las posibles causas: verificamos si está enchufada, si la lámpara se quemó, si el enchufe está bien, si hay corriente, entre otras. En función de la situación, vemos qué curso de acción tomar. Por ejemplo, si está quemada, buscamos en casa si hay un repuesto y si no lo hay, vamos al supermercado a comprar uno.

¿Qué tienen en común estos dos casos? En ambos:

- descompusimos el problema en problemas más pequeños (**DESCOMPOSICIÓN**): “Dónde iremos” y “cuánto cuesta” en el primer ejemplo, y “cuál es la causa de que no prenda” en el segundo,
- tomamos sólo lo importante, por ejemplo, no nos preocupamos en este momento si será necesario tomar un ómnibus para la salida y si lo pagaremos con billetes, monedas o tarjeta. (**ABSTRACCIÓN**),
- tuvimos en cuenta el conocimiento de situaciones anteriores similares: por ejemplo recordamos otras salidas previas y tomamos ideas de ahí (**RECONOCIMIENTO DE PATRONES**) y
- organizamos un plan paso a paso. Por ejemplo, para cambiar la lámpara, si me falta un repuesto, ir al supermercado a comprar uno (**ALGORITMO**). Un **algoritmo** es una secuencia ordenada para resolver un tipo específico de problema.

O sea, el pensamiento computacional refiere a tomar un problema complejo y dividirlo en problemas más pequeños y manejables, es lo que se llama descomposición. Cada uno de ellos puede ser analizado en forma individual, considerando cómo se han resuelto problemas similares anteriormente. Esto se denomina reconocimiento de patrones. Se trata también de concentrarse en los hechos y datos importantes, ignorando lo irrelevante, esta es la noción de abstracción. Luego, se puede diseñar un conjunto de pasos para resolver cada uno de esos problemas más pequeños, o sea, diseñar el o los algoritmos.

Si bien el nombre “Pensamiento Computacional” podría sugerir que es una temática exclusiva para personas del ámbito de la computación, es completamente general, como vimos en los ejemplos.

Hay muchas definiciones “formales” de pensamiento computacional. Una lo define como “un proceso de resolución de problemas que tiene, entre otras, las características de formular problemas de forma que permitan el uso de una computadora y otras herramientas para ayudar en su resolución, organizar y analizar lógicamente datos, automatizar soluciones a través del pensamiento algorítmico (como una serie ordenada de pasos), representar datos a través de abstracciones como modelos o simulaciones, identificar, analizar e implementar posibles soluciones para alcanzar la más efectiva y eficiente combinación de pasos y recursos, y generalizar y transferir este proceso de resolución a una amplia variedad de problemas.

El pensamiento computacional también está relacionado con habilidades o competencias tales como:

- la confianza en el manejo de la complejidad,
- la persistencia en el trabajo con problemas difíciles,
- la tolerancia a gestionar problemas abiertos y
- la habilidad para comunicar y trabajar con otras personas para alcanzar una solución o meta.

El pensamiento crítico y las habilidades de resolución de problemas son competencias esenciales que los estudiantes requieren para desempeñarse en ambientes complejos como los actuales.

En resumen, PC trata de:

- Abstracción: concentrarse en lo fundamental, ignorando lo irrelevante
- Descomposición: partir los datos, procesos o problemas en partes más pequeñas y manejables

- Reconocimiento de patrones: encontrar patrones o regularidades y
- Diseño de algoritmos: seguir paso a paso instrucciones para resolver problemas

Estas mismas ideas se aplican cuando resolvemos problemas. Veamos varios ejemplos concretos.

1.9.1 Ejemplos de Abstracción

Ejemplos de Abstracción en la vida real:

- Horario de clases
- Mapa de geografía

1.9.2 Ejemplos de Descomposición

- Ordenar habitación de adolescente: ¿por dónde empezar?
- Organizar recorrido turístico
- Hacer el desayuno

1.9.3 Ejemplos de Reconocimiento de Patrones

- Sumar los números de 1 a 100
- Escribir las palabras “tambo”, “cambio”, “bombero”, “costumbre”: la regla es m antes de b

1.9.4 Ejemplos de Algoritmo

- Poner la mesa: implica por ejemplo: poner el mantel, poner el plato, poner el cuchillo a la derecha, poner el tenedor a la izquierda, poner la servilleta sobre el plato.
- Hacer un licuado: pelar la fruta, ponerla en la licuadora, agregar leche y azúcar, poner la tapa, prender 2 minutos, apagar, servir

1.10 Algoritmo y propiedades

Como vimos anteriormente, un algoritmo es una secuencia ordenada de pasos para resolver un tipo específico de problema. Sus propiedades son:

- finito: como opuesto a infinito. Si es infinito, se denomina proceso computacional.
- entrada: el algoritmo tiene datos de entrada. Por ejemplo, en el caso del algoritmo de Euclides (procedimiento para calcular el máximo común divisor de dos números enteros), los datos de entrada son los dos números.
- salida: el algoritmo tiene datos de salida. En el mismo ejemplo anterior, la salida es el MCD (máximo común divisor).
- efectivo: todas las operaciones son lo suficientemente básicas para que puedan ser realizadas por una persona en tiempo finito. Por ejemplo, si se debe dividir, indicar la cantidad de cifras a tomar.
- preciso: todos los pasos están bien detallados. En el caso de una receta de cocina, por ejemplo si se indica “dar un golpe de horno” no se cumpliría con la precisión, pues no se detalla si un golpe de horno es un breve momento, minutos u horas

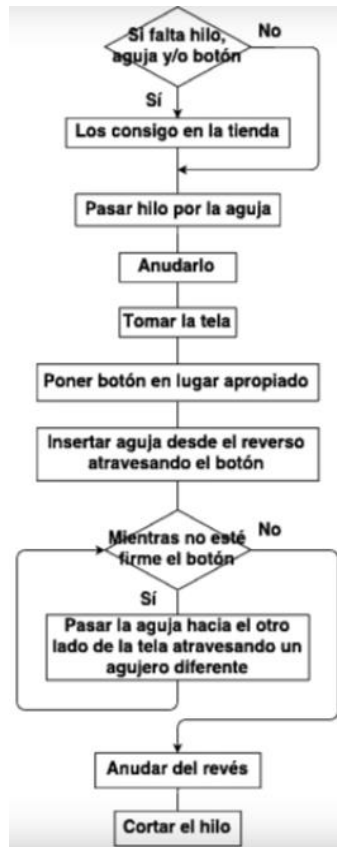
1.11 Estructuras de control

Veamos otro ejemplo de algoritmo bien sencillo de la vida real: podría ser el de describir el proceso de coser un botón. Una versión sería:

```
si me falta hilo, aguja y, o botón
    los consigo en la tienda
fin si
pasar un trozo de hilo por la aguja
anudarlo
tomar la tela
poner el botón en el lugar apropiado
insertar la aguja desde el reverso de la tela atravesando un agujero del botón
mientras no esté firme
    pasar la aguja con el hilo hacia el otro lado de la tela atravesando un agujero diferente del
    botón
fin mientras
anudar el hilo del lado del revés
cortar el hilo
```

Esta es una **secuencia** de pasos en orden, y también incluye una **decisión** (“si me falta hilo, aguja ...”) y una **iteración**: “mientras no esté firme”. La secuencia, la decisión y la iteración son lo que llamamos “estructuras de control” y con ellas se construyen los programas.

El algoritmo es generalmente el punto de partida para crear efectivamente un programa, y se puede representar en **seudocódigo**, en una forma similar como vimos recién o también con un **diagrama de flujo**, como se muestra en la siguiente imagen:



En resumen, las estructuras de control son:

- Secuencia
- Decisión
- Iteración

Analicemos estos ejemplos:

a) Viajar:

ir al Aeropuerto Internacional de Carrasco el 7 de junio hora 8
tomar el avión de 9 hs a Río
llegar a Río 12 hs.
ir al hotel Columbia

Es una lista de pasos, uno tras otro. Es una secuencia. El formato es:

.....
....
....

b) Situación en casa:

si tengo frío
me pongo un abrigo

Situación en un café:

```
si tengo tiempo y tengo mucha hambre
    pedir jugo, torta y waffles
en otro caso
    pedir sólo café
```

Aquí hay decisiones. El flujo de la ejecución se altera en función de una decisión. Los formatos posibles son:

```
si (condición)
    ....
```

y

```
si (condición)
    ....
en otro caso
    .....
```

Puede indicarse “fin si” al final de la estructura. En inglés es “*if* (si) ...” o “*if* (si)*else* (en otro caso)...”

c) Preparar sandwiches:

```
repetir 5 veces
    tomar un pan
    ponerle queso
    ponerle jamón
    tomar otro pan
```

El formato es:

```
repetir x veces
    ....
```

Esta estructura se llama en inglés “*for*”. Se utiliza cuando se conoce la cantidad de veces que se desea repetir. Puede indicarse al final “fin repetir”.

d) Regar

```
mientras esté seca la tierra
    agregar 15 gotas de agua en la maceta
```

El formato es:

```
mientras (condición)
    ....
```

En inglés se denomina “*while* (mientras)”. Se repite el proceso hasta que se de la condición. Puede indicarse al final “fin mientras”.

e) Cocinar panqueques (tengo la masa pronta).

```
repetir
    poner masa en la sartén
    esperar 2 minutos
    dar vuelta
    esperar un minuto
    sacar el panqueque
hasta que se acabe la masa
```

El formato es:

```
repetir
    .....
hasta (condición)
```

El nombre en inglés de esta última estructura es “*repeat.... until*” (repetir hasta). La diferencia entre “repetir” y “mientras” es que el bloque dentro del mientras se ejecutará 0, 1, o más veces; en el repetir se ejecutará como mínimo 1 vez.

En los “....” puede ir cualquier otra sentencia, inclusive otras estructuras repetitivas, condiciones o secuencias.

No es necesario identificar explícitamente cada estructura de control cuando se realiza el programa, simplemente se utilizan.

2 Variables e Introducción a JS

2.1 Introducción a variables

A efectos de introducir el concepto de variable, pensemos en estos casos. Tenemos un juego y queremos saber el puntaje que vamos haciendo. Debemos guardarlo en algún lugar. Lo mismo pasa con el “high score”, tengo que “llevarlo anotado”. Otro caso podría ser que se necesita calcular el precio con IVA, a partir del precio.

En computación, esa idea de “guardar el dato en algún lado” se asocia al concepto de variable. Así tendría una variable para llevar el puntaje actual, otra para el máximo score. En el caso del negocio, tendría que tener una variable con el precio.

Formalmente, una **variable** es un lugar de la memoria de la computadora para almacenar un dato. Tiene un nombre y un tipo de dato (ejemplo: un número, un texto, etc.). El nombre se recomienda que sea nemotécnico, o sea, que refleje su contenido. Así, si en una variable acumularemos el puntaje, es más claro ponerle como nombre “puntaje” que “XR86”.

2.2 Seudocódigo y Ejemplos

2.2.1 Cálculo del IVA

Retomemos el ejemplo del precio. Queremos un programa que solicite un precio y lo muestre con IVA. ¿Qué necesitaremos? Tendría que tener una variable donde guardar el precio y mostrar luego el cálculo.

El pseudocódigo podría ser:

```
mostrar "ingrese precio"
leer precio
mostrar "el precio con iva es "
mostrar precio * 1.22
```

El bloque “leer” permite que se ingrese un dato y quede guardado en la variable que se indicó. El bloque mostrar permite desplegar un dato o un cálculo.

Podría haberlo escrito así:

```
mostrar "ingrese precio"
leer precio
mostrar "el precio total es ", precio * 1.22
```


2.2.2 Suma de 10 números

Supongamos que queremos sumar 10 números que dirá el usuario. Necesito una variable para guardar el total y otra para leer el dato.

```
suma <- 0;
para i desde 1 hasta 10 con paso 1 hacer
    mostrar "ingrese dato";
    leer dato;
    suma <- suma + dato;
fin para
mostrar "la suma es ", suma;
```

La sentencia “suma <- 0” es una **asignación**. Lo de la derecha se “asigna” a la variable de la izquierda. En los lenguajes de programación usualmente se sustituye la flecha por el signo de “=”.

En algunos lenguajes de programación, la sentencia “para” (o “for” en inglés) se escribe:

```
para (i= 1; i <= X; i = i + 1)
```

i representa la variable que controla la estructura. $i=1$ indica que el valor inicial de la variable i es 1; $i \leq X$ es la condición de trabajo, mientras se cumpla esta condición se realizará el proceso; $i = i + 1$ es el incremento que se realizará con cada nueva repetición. En este caso X corresponde al valor 10.

El mismo ejercicio en vez de con la estructura de control “for” (para, repetir), hecho ahora con “while” quedaría:

```
acum = 0
van = 0
mientras van < 10
    leo dato
    acum = acum + dato
    van = van + 1
fin mientras
mostrar acum
```

Fue necesario agregar una variable donde se lleva la cuenta de cuántos números se han ingresado (variable van).

Si se desea con “repeat” (repetir), la implementación sería:

```
acum = 0
van = 0
repetir
    leo dato
    acum = acum + dato
    van = van + 1
hasta (van=10)
mostrar acum
```

2.3 Asignación

Como se vio en el ejemplo anterior, la sentencia “suma <- 0” es una **asignación**. Lo de la derecha se “asigna” a la variable de la izquierda. Visualmente se puede representar con la siguiente imagen:



En general, no se puede asignar cualquier valor a cualquier variable:



2.4 Corrida a mano

Hacer la corrida a mano de un programa implica simular la ejecución del programa por parte de la máquina. Permite detectar posibles errores.

Tomemos por ejemplo el código anterior. ¿Pedirá 10 números? ¿Mostrará bien el total? Se hará la corrida a mano. Para simplificar, en vez de hacerlo con 10 números, se supondrá que hay solamente 2 números. Así, el programa sería:

```
acum = 0
van = 0
repetir
    leo dato
    acum = acum + dato
    van = van + 1
hasta (van=2)
mostrar acum
```

Para realizar la corrida se anotan las variables:

acum = 0, empieza en 0

van = 0, empieza en 0

Luego viene la sentencia "leo dato". La variable dato toma el valor que ingrese el usuario, por ejemplo 5. Se anota:

dato = 5

Después aparece la sentencia "*acum* = *acum* + *dato*". La variable *acum* quedaría entonces en 5. Se representa así:

acum = 0 / 5

La barra vertical indica el nuevo valor.

La siguiente instrucción es "*van* = *van* + 1". Así:

van = 0 / 1

Se verifica la condición de si *van* es 2, lo cual no es cierto, por lo cual sigue en la estructura repetitiva.

Se solicita un nuevo dato, el usuario ingresa el valor 8:

dato = 5 / 8

Nuevamente, la siguiente instrucción es: "*acum* = *acum* + *dato*". Así:

acum = 0 / 5 / 13

La próxima es "*van* = *van* + 1".

van = 0 / 1 / 2

Al chequearse la condición, se cumple. La siguiente línea es "mostrar *acum*", por lo que en pantalla aparecerá el valor 13.

Revisando el proceso, el programa debía pedir 2 datos y mostrar su suma; pidió dos valores (que fueron el 5 y 8) y mostró correctamente su suma (13). Se puede inferir que para el caso de 10 valores (donde se utilizará el valor 10 en lugar del modificado 2), el programa funcionará bien: pedirá 10 valores y mostrará su suma.

Es posible también hacer demostraciones matemáticas para determinar la corrección del programa. Se ve en cursos más avanzados.

2.5 Más ejemplos

2.5.1 Producto de 20 números

Leer 20 datos y mostrar el producto de los 20 números. Una posible solución en pseudocódigo es:

```
van <- 0;
prod <- 1;
mientras van < 20
    mostrar "ingrese dato";
    leer dato;
    van <- van + 1;
    prod <- prod * dato;
fin mientras
mostrar "producto es ", prod;
```

Es similar al ejercicio de la suma visto antes, en este caso los cambios son: se utiliza multiplicación en vez de suma y se inicializa la variable *prod* en 1 (neutro del producto). Observar que también se ha modificado el nombre de la variable para reflejar que en este caso se realiza un producto, no una suma.

2.5.2 Loop

Reconsideremos el ejercicio de leer 20 datos y mostrar el producto. ¿Qué pasaría si por error omitimos la sentencia “*van* = *van* + 1”? El programa quedaría en “*loop*”, en circuito infinito. No terminaría. Es ciertamente recomendable asegurarse que los ciclos iterativos tengan terminación.

2.5.3 Valores repetidos

Leer 3 datos, se sabe que 2 son iguales y el tercero es diferente. Mostrar el valor repetido. En este caso, una posible solución en pseudocódigo es:

```
mostrar "ingrese los 3 datos";
leer a;
leer b;
leer c;
si (a=b) entonces
    mostrar "el valor es ", a;
sino
    mostrar "el valor es ", c;
fin si
```

Observar que lo que se pone entre “” sale igual en la pantalla, es lo que denominamos un “**string**” o secuencia de caracteres.

2.5.4 Promedio

Leer datos hasta que se ingrese el número 99 e imprimir el promedio. El número 99 NO integra el promedio. En este caso, una posible solución en pseudocódigo es:

```
total = 0
van = 0
ingresar dato
mientras (dato <> 99)
    total = total + dato
    van = van + 1
    ingresar dato
fin mientras
si van > 0
    mostrar (total / van)
en otro caso
    mostrar "sin datos"
fin si
```

En este caso, es necesario llevar una variable donde se cuenta la cantidad de datos (van) y otra donde se acumulan todos los datos (total).

2.6 Intercambio de 2 variables

Un problema frecuente es el de intercambiar dos variables. Por ejemplo, se tiene una variable A que vale 20 y una variable B que vale 10. Se pide intercambiar el contenido de ambas variables (resolverlo en forma genérica, no en particular para los valores 10 y 20).

Así, la solución sería:

```
AUX = A
A = B
B = AUX
```

En detalle, en pseudocódigo completando el ingreso de datos podría ser:


```
mostrar "ingrese el valor de a";  
leer a;  
mostrar "ingrese el valor de b";  
leer b;  
mostrar "el valor de a ", a;  
mostrar "el valor de b ", b;  
c <- b;  
b <- a;  
a <- c;  
mostrar "el valor de a es ", a;  
mostrar "el valor de b es ", b;
```

2.7 Programa

Hemos hecho varios ejemplos de programa. Una programa es un conjunto de instrucciones en orden lógico que describen un algoritmo determinado.

Las características esperables de un programa son:

- eficaz: hace lo pedido;
- entendible: el código es claro;
- modificable: el código puede ser fácilmente modificado, para posibles ajustes o ampliaciones; y
- eficiente: ejecuta en el menor tiempo posible. Notar que la eficiencia va después de la eficacia.

2.8 Introducción a JavaScript

JavaScript es un lenguaje de programación diseñado en 1995. Hoy se ha convertido en el lenguaje de la web. ECMAScript es la especificación del lenguaje utilizada para su creación. Con JS se escriben los programas o “scripts”. JS es muy versátil y puede ser ejecutado en un browser, un servidor o cualquier dispositivo que tenga un motor de JS (JS Engine). Se le llama también intérprete o motor.

Importante: en los próximos ejemplos nos familiarizaremos con la sintaxis y características de JS y luego más adelante veremos cómo integrarlo en nuestras páginas. El enfoque es “conocer los elementos que tiene”, así como conocer las piezas de un juego de construcción, para luego efectivamente construir.

Haremos varias pruebas. En todos los casos utilizaremos Chrome desde Windows. Abriremos el navegador en una página en blanco (escribiendo about:blank en la barra de direcciones) y presionando F12 aparece la “consola”, donde podremos interactuar y hacer las primeras pruebas.

La consola de un navegador permite ejecutar sentencias o expresiones JS en una forma interactiva. También permitirá depurar un programa. El intérprete analiza y ejecuta el texto introducido. Si es una expresión, la evalúa y muestra el resultado.

2.9 Expresiones aritméticas

Una expresión aritmética es una constante, o variable, o combinación de constantes y variables vinculadas por los operadores de + - * / y otros operadores. Tiene resultado numérico. Puede tener paréntesis.

Ejemplos.

4 + 3

50 % 3 nota: el operador % representa el “módulo”, el resto de la división entera. 50%3 es el resto de la división de 50 entre 3, que da 2. 50 dividido 3 da cociente 16.

(total * 1.23+5)+ (precio*0.20-0.4)

Utilizaremos la consola de JS para probar las expresiones aritméticas. Los navegadores ya proveen una consola, que permite dar comandos de JS. Para abrirla en Chrome, utilizar ctrl+shift+i o F12.

2.10 Variables booleanas y expresiones lógicas

2.10.1 Variables booleanas

Una variable booleana (*boolean*) o variable lógica es una variable que solamente vale verdadero (*true*) o falso (*false*).

2.10.2 Operadores relacionales

Los operadores relacionales son:

menor: <

mayor: >

mayor o igual: >=

menor o igual: <=

igualdad: == y === (en JS)

diferentes: !=

Notar que el operador para preguntar por igualdad consta de "===" o "===", la asignación es "=". La comparación estricta (===) sólo es verdadera si los operandos son del mismo tipo y los contenidos coinciden. La comparación con == convierte los operandos al mismo tipo antes de hacer la comparación.

2.10.3 Operadores lógicos

Los operadores lógicos son: *and* (y), *or* (o), y *not* (no). Considerando la sintaxis de JavaScript, el *and* se representa por &&, el *or* por || y el *not* por !. En las tablas siguientes se explica cada uno.

El operador *not* invierte el valor verdadero en falso y viceversa:

	not
verdadero	falso
falso	verdadero

El operador *and* (y) tiene resultado cierto cuando las condiciones sobre las que se aplica son ciertas. Por ejemplo, si para ir al cine las condiciones son a) tener tiempo y b) tener dinero, sólo se irá al cine cuando ocurran las 2 condiciones simultáneamente.

ejemplo: tener tiempo	ejemplo: tener dinero	and ejemplo: tener tiempo y tener dinero
falso	falso	falso
falso	verdadero	falso
verdadero	falso	falso
verdadero	verdadero	verdadero

El operador *or* (o) tiene resultado cierto cuando alguna de las condiciones son ciertas. Por ejemplo, si para enterarnos de una noticia se puede hacer a través de a) la radio b) la TV, basta con escuchar la radio, ver la TV o ambas cosas para enterarnos.

ejemplo: escuchar radio	ejemplo: ver TV	or ejemplo: escuchar radio o ver TV
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	verdadero

Las constantes lógicas son: verdadero (*true*) y falso (*false*).

2.10.4 Expresiones lógicas

Una expresión lógica es una constante lógica, o variable lógica, o 2 expresiones aritméticas separadas por operadores relacionales, u operadores lógicos actuando sobre constantes lógicas, variables lógicas, expresiones relacionales u otras expresiones lógicas. El resultado es verdadero o falso.

Así como en matemáticas se sabe que las operaciones de multiplicar y dividir se ejecutan antes que la suma y la resta, la **precedencia** de los operadores lógicos es: *not*, *and*, *or*. Primero se evaluará el *not*, luego el *and* y finalmente el *or*. Se recomienda, a los efectos de clarificar, utilizar paréntesis.

Ejemplos de expresiones lógicas. Probar en la consola de JS. Sean:

```
x = 3;  
y = 4;  
z = 2;  
f = false;
```

¿Cuánto vale cada una de las siguientes expresiones?

- a) $(x > z) \ \&\& \ (y > z)$
- b) $(x + y / 2) \leq 3.5$
- c) $! f$
- d) $! f \ || \ ((x < z) \ \&\& \ (z \geq 3 * y))$
- e) $f \ \&\& \ (x > z + y)$

Respuestas:

- a) verdadero
- b) falso, 5 no es menor que 3.5
- c) verdadero
- d) verdadero. Ya al evaluar "not f" se sabe que la expresión entera es verdadera pues es un "or".
- e) falso. El primer término es falso, por lo cual, dado que es un "and", la expresión entera es falsa. Se dice que se hace evaluación de "corto circuito" cuando apenas se detecta el resultado de la expresión se deja de hacer la evaluación. Tiene como ventaja el ahorro de tiempo.

2.11 JS: Números y operadores

2.11.1 Números en JS

Puedo usar números con o sin punto decimal. Se guardan en punto flotante.

```
> 3.14  
< 3.14
```

Puedo usar notación científica:

```
> 12e5  
< 1200000
```

El rango de los números es $-(2^{53} - 1)$ y $2^{53} - 1$. ¿Qué pasaría si divido 1/0?. Da **Infinity**. Si divido: $-3/0$ da **-Infinity**.

```
> 1/0
< Infinity
> -3/0
< -Infinity
```

El número más grande es Number.MAX_VALUE y el más chico es Number.MIN_VALUE:

```
> Number.MAX_VALUE
< 1.7976931348623157e+308
> Number.MIN_VALUE
< 5e-324
```

Si quiero mostrarlo con cierta cantidad de decimales redondeado, puedo usar "toFixed(n)". El número ponerlo entre paréntesis.

```
> (9.43).toFixed(1)
< "9.4"
> (9.48).toFixed(1)
< "9.5"
```

También hay valores predefinidos de Pi y e, así como otras funciones: sqrt (raíz) y pow (potencia):

```
> Math.PI
< 3.141592653589793
> Math.E
< 2.718281828459045
> Math.sqrt(100)
< 10
> Math.pow(2,3)
< 8
```

Las operaciones en punto flotante no son exactas:

```
> 0.1 + 0.2
< 0.30000000000000004
```

2.11.2 Operadores en JS

Los operadores aritméticos en JS son + - * / . Se puede sumar, restar, multiplicar, dividir y usar paréntesis. Si ponemos una expresión errónea avisa. Respeta la precedencia: primero paréntesis, luego * / y luego + -.

```
> 13 + 7
< 20
> 13 - 2.5
```



```
< 10.5
> (8*3-4)/5
< 4
> 8/3
VM175:1 Uncaught SyntaxError: Invalid or unexpected token
> 8 3
VM177:1 Uncaught SyntaxError: Unexpected number
```

El operador + y el - se pueden usar en forma “unaria”, por ejemplo:

```
> -3+8
< 5
```

Hay otros operadores, como módulo, que, como se indicó, es el resto de la división:

```
> 23%4
< 3
```

2.12 Uso de variables booleanas

Se desea un programa que lea datos hasta que se ingrese el valor 99 e indicar *al final* si pasó el valor 37 entre los datos ingresados. Analizar la siguiente solución:

```
paso = false
leo dato
mientras (dato != 99)
    si (dato == 37)
        paso = true
    en otro caso
        paso = false
    fin si
    leo dato
si (paso)
    mostrar "SI"
en otro caso
    mostrar "NO"
fin si
```

Realizar la corrida a mano. ¿Qué sucede? En realidad, el programa mostrará si el último dato antes del 99 era o no el 37.

Una solución correcta es:

```
paso = false
leo dato
mientras (dato != 99)
    si (dato == 37)
        paso = true
    fin si
    leo dato
si (paso)
    mostrar "SI"
en otro caso
    mostrar "NO"
fin si
```

Observar que no es necesario preguntar: "si (paso == true)", con poner "si (paso)" alcanza, pues dado que es una variable booleana sus únicas opciones son verdadero o falso. Si nos interesara preguntar por el valor falso, en vez de poner si (paso== false) poner si (!paso).

En la línea "mostrar 'SI'", el texto 'SI' es un *String*, una secuencia de caracteres. Por pantalla aparecerá la palabra SI.

3 Codificación

3.1 Strings en JS

Un string es una secuencia de caracteres entre “”.

Haremos pruebas. Puedo poner una frase:

```
> "hola que tal"
< "hola que tal"
```

Si necesito poner comillas dentro del texto puedo usar comillas simples.

Puedo “concatenar” textos, esto es, unirlos:

```
> "hola " + "Ana"
< "hola Ana"
```

Para saber el largo de un string tenemos la propiedad “length”. Para acceder a una propiedad se utiliza el punto y luego el nombre de la propiedad sin paréntesis:

```
> "esta frase es larga".length
< 19
```

3.2 Sobrecarga de operadores

Es interesante observar que el operador + tiene varias funciones, es la noción de “sobrecarga” del operador. Cuando lo usamos con números, suma. Cuando lo usamos con textos, los concatena. También se puede usar en forma “unaria”.

```
> 13+7
< 20
> "hola" + " que tal"
< "hola que tal"
```

¿Qué pasaría si se combinan string y números? JS hace la conversión automática. Cuando hay alguna ambigüedad, utiliza reglas de prioridad predefinidas para resolverlas.

Veamos estos ejemplos:

- a) $13 + 7$ da 20
- b) “13”+”7” da “137”
- c) “13”+7 da “137”, porque da prioridad al operador + de Strings, convirtiendo el 7 a string
- d) $13+7$ da “137”

Si pongo el operador unario antes del “13”, lo pasa a número.

```
> + "13" + 7  
< 20  
> "13" + 7  
< "137"
```

Para convertir un String a número usar `parseInt` o `parseFloat`:

```
> parseInt("25")  
< 25  
> parseFloat(25.4)  
< 25.4  
> parseInt("25.4")  
< 25  
> parseInt("casa")  
< NaN
```

NaN quiere decir “not a number” (no es un número).

Nota: `parseInt` trata de reconocer el entero, si vienen otros datos después los ignora. Ej:

```
> parseInt("123ab")  
< 123  
> parseInt("ab1234")  
< NaN
```

Si se requiere validar que efectivamente sea un entero usar `Number()`. La función `Number()` convierte un valor en su representación numérica, pero espera que el valor sea completamente numérico, y si contiene cualquier caracter no numérico, incluso si está precedido por dígitos, resultará en NaN.

```
> Number("123ab")  
< NaN
```

3.3 Variables en JS

Para definir una variable se usa la palabra **var** o **let**. Más adelante se verán sus diferencias. En general, usaremos **let**. Es recomendable definir siempre las variables previamente a su uso.

El nombre de una variable puede contener letras, números, \$ y underscore (guión bajo). Debe empezar con letra. Es case sensitive, o sea que diferencia mayúsculas y minúsculas. No se pueden usar palabras reservadas.

El operador de asignación es el `=`. Se puede declarar y asignar a la vez o hacerlo por separado. Cada sentencia debería terminar en “;” (podría omitirse si es una sola sentencia).

```
> let x = 3;
< undefined
> let y = 5+x;
< undefined
> y = y-2
< 6
> x = y+z;
VM523:1 Uncaught ReferenceError: z is not defined
  at <anonymous>:1:7
```

3.4 Console

Para desplegar una variable o información, también se puede utilizar console.log:

```
> let aux = 3;
< undefined
> console.log(aux);
```

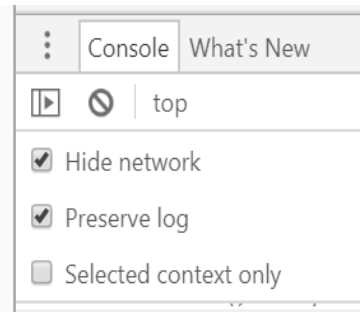
Se verá en la consola el valor. Puedo poner solo aux y también se ve el valor:

```
> aux
< 3
```

Hay otros mensajes para console: por ej. warn, error

```
> console.warn("hi")
⚠ ▶hi VM225:1
< undefined
> console.error("este es un error")
✖ ▶este es un error VM283:1
```

Se puede limpiar la consola con console.clear(), aunque depende también de la opción “preserve log”.



3.5 Sentencias en JS

Un programa es una secuencia de sentencias que se ejecutan en el orden en que fueron definidas.

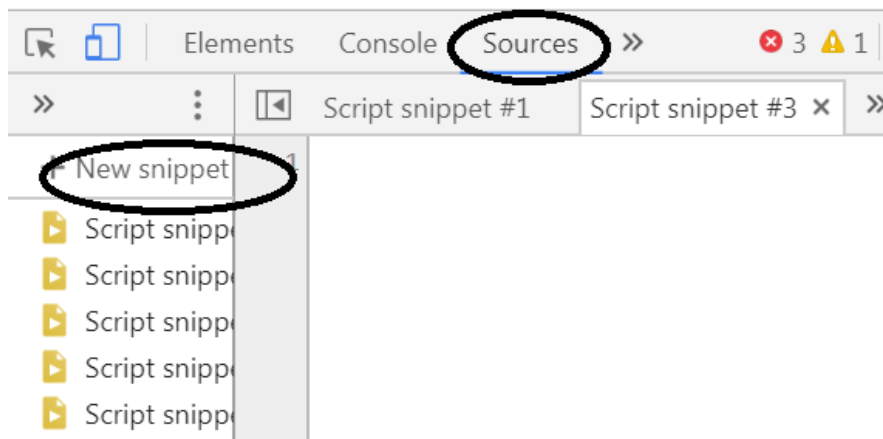
En JS se recomienda delimitar las sentencias siempre con “;”. Se puede omitir si la sentencia termina con una nueva línea, pero se recomienda ponerlo siempre. Además, se recomienda poner una sentencia por línea por legibilidad. Puede ocupar más de una si fuera muy larga.

En el código también se pueden poner comentarios, que ayudan al programador a entender lo que se hace. Se ponen:

si es de una línea: //.

Si fuera de varias: /**/.

En las pruebas que se hicieron hasta ahora se probaron sentencias individuales. Vamos a utilizar ahora en la consola, dentro de Sources, la opción de “snippet”, fragmento o retazo para poner varias líneas en un mismo programa. Luego veremos cómo incluir JS para ser utilizado desde una página Web.



Se crea un nuevo snippet y se le puede cambiar el nombre, por ejemplo: “Ejemplos básicos”.

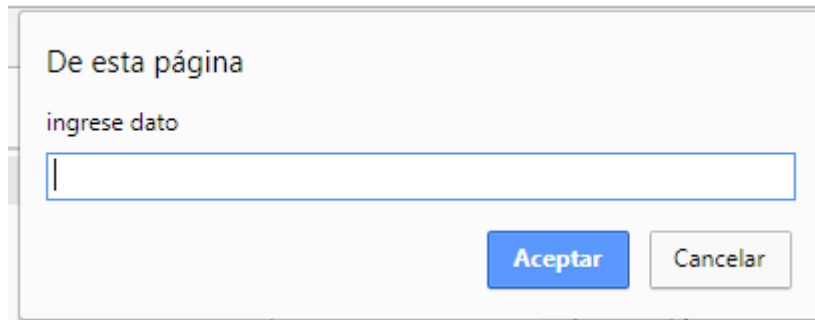
3.5.1 Ingreso de datos y mostrar datos

Codificaremos el ejemplo de leer un precio y mostrarlo con IVA:

```
mostrar "ingrese precio"
leer precio
mostrar "el precio total es ", precio * 1.22
```

Tenemos un mensaje de mostrar, que lo podemos representar con “console.log”. Nos falta la sentencia para leer un dato. Se utiliza “prompt”. Por ejemplo, se desea ingresar un dato, dentro de la sentencia “prompt” se pone el texto a desplegar. Al ejecutarse, se muestra una ventana que muestra el texto que se puso.

```
> let x = prompt("ingrese dato");
< undefined
```



(Nota: La ventana donde se está probando se abrió con “about: blank” en la barra de direcciones. Si estuviera trabajando con otra ventana, saldría la dirección de la página en la ventanita).

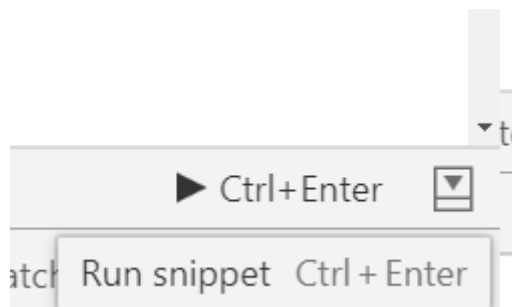
Sugerencia importante: para evitar problemas de operaciones no deseadas, asegurarse de que cuando se toma un dato se lo trate como número utilizando “parseInt”.

Para mostrar el resultado se puede usar console.log o “alert”, que despliega una ventana similar a la de prompt. En el código se muestran ambas opciones.

El código JS quedaría:

```
// calcula el iva
let x = parseInt(prompt("ingrese el precio"));
x = x * 1.22;
console.log("el total es " + x);
alert("el total es "+x);
```

Lo ubicamos dentro del snippet. Para ejecutar este código, presionar “Ctrl+enter” o presionar la opción de “Run Snippet”:



Si se desea guardar el código, se indica sobre el nombre del archivo, presionando botón derecho, “save as” y se indica lugar. Guarda un archivo de texto con el código.

3.5.2 Suma de 10 números en JS: for

Retomamos el pseudocódigo ya visto:

```
suma <- 0;
para i desde 1 hasta 10 con paso 1 hacer
    mostrar "ingrese dato";
    leer dato;
    suma <- suma + dato;
fin para
mostrar "la suma es ", suma;
```

Tiene una sentencia “para”, en JS se codifica “for”. El formato es:

```
for (inicio; condición; incremento) {
.....
}
```

inicio corresponde al valor inicial para la repetición.

condición está vinculado a la terminación, por ejemplo $i \leq 10$

incremento refiere a la variación que se realizará en cada bucle, por ejemplo $i = i + 1$

El código en JS sería:

```
let suma = 0;
let i;
let dato;
for (i=1; i<=10; i++){
    dato= parseInt(prompt("Ingrese dato"));
    suma = suma + dato;
}
alert("la suma es "+suma);
```

Es importante resaltar que hay más de una forma posible de codificar este mismo programa.

3.5.3 Suma de 10 números: while

Codificaremos ahora el mismo ejercicio usando mientras (“while”). El pseudocódigo es similar a:

```
acum = 0
van = 0
mientras van <10
    leo dato
    acum = acum + dato
    van = van + 1
fin mientras
mostrar acum
```


La estructura mientras se codifica:

```
while (condición) {  
    Código a ejecutar  
}
```

Así, el código JS sería:

```
let suma = 0;  
let i=0;  
let dato;  
while (i <=10){  
    i=i+1;  
    dato= parseInt(prompt("Ingrese dato"));  
    suma = suma + dato;  
}  
alert("la suma es "+suma);
```

En JS se dispone también de la estructura “repetir mientras”, que ejecuta un bloque de código al menos una vez y luego repite la ejecución mientras una condición especificada sea verdadera. La estructura “repetir mientras” se codifica:

```
do {  
    código a ejecutar  
} while (condición);
```

3.5.4 Valores repetidos: if

Codificaremos el ejemplo de leer 3 datos (asumiendo 2 iguales y uno diferente), para indicar el repetido. El pseudocódigo que vimos es:

```
mostrar "ingrese los 3 datos";
leer a;
leer b;
leer c;
si (a==b) entonces
    mostrar "el valor es ",a;
sino
    mostrar "el valor es ", c;
fin Si
```

Las decisiones se codifican:

```
if (condición) {
    sentencias
}
else {
    sentencias
}
```

```
if (condición) {
    sentencias
}
```

El código en JS sería:

```
// se leen 3 datos, 2 iguales y uno diferente, decir el repetido
alert("ingrese los 3 datos");
let a = parseInt(prompt("Primer dato "));
let b = parseInt(prompt("Segundo dato "));
let c = parseInt(prompt("Tercer dato "));
if (a===b){
    alert("El repetido es "+a);
}
else{
    alert("El repetido es "+ c);
}
```

3.5.5 Switch

La sentencia switch evalúa una expresión, la compara con un conjunto de valores predefinidos, y ejecuta comandos según el caso. Las sentencias indicadas en la opción default se ejecutan si no se cumplió ninguna de las condiciones anteriores.

En cada caso, se recomienda utilizar una sentencia break para asegurarse que el programa salga del switch, si se omite continúa con la siguiente instrucción dentro del switch. Veamos un ejemplo donde la variable result queda cargada con distintos textos según la variable valor:

```
let result = "";
switch (valor) {
  case 0:
    result = "opción 0";
    break;
  case 1:
    result = "opción 1";
    break;
  default:
    result = "otro valor";
}
```

3.6 Ejemplos

3.6.1 Máximo

Se desea leer números hasta que se ingrese el valor 0. Al final, mostrar el mayor dato ingresado (excluyendo el 0). Analizaremos varias posibles soluciones. Una primera opción de pseudocódigo es:

```
max = 0
leo dato
mientras (dato !=0)
  si (dato > max)
    max =dato
  fin si
  leo dato
fin mientras
mostrar max
```

¿Qué pasa si como primer dato se ingresa el valor 0? Debería aparecer un texto indicando esta situación. La versión corregida sería:

```

max = 0
huboDatos = false
leo dato
mientras (dato !=0)
    huboDatos = true
    si (dato > max)
        max= dato
    fin si
    leo dato
fin mientras
si (huboDatos)
    mostrar max
en otro caso
    mostrar "no hubo datos"
fin si

```

Consideremos otro caso más. ¿Qué pasa si los datos son todos negativos? El programa indicaría que el máximo es 0, lo cual no es correcto. Una alternativa es inicializar la variable *max* con el menor valor posible: *LowValue*. Este valor dependerá del tipo de datos que se considere, pero está perfectamente definido para cada uno de ellos.

```

max = LowValue
huboDatos = false
leo dato
mientras (dato !=0)
    huboDatos = true
    si (dato > max)
        max = dato
    fin si
    leo dato
fin mientras
si (huboDatos)
    mostrar max
en otro caso
    mostrar "no hubo datos"
fin si

```

Otra alternativa sería inicializar *max* con el primer dato válido. Notar que la sentencia "mostrar *max*" despliega el valor de la variable *max*, o sea, su contenido.

Modificaremos el ejercicio del máximo, para incorporar que se pueda incluir cualquier valor, inclusive el 0. Así, el pseudocódigo sería:

```

max = LowValue
huboDatos = false
mostrar "quiere ingresar datos?"
leo resp
mientras (resp == "S")
    mostrar "ingrese dato"
    leo dato
    huboDatos = true
    if (dato > max)
        max = dato
    fin si
    mostrar "mas datos?"
    leo resp
fin mientras
si (huboDatos)
    mostrar "Máximo vale" + max
en otro caso
    mostrar "no hubo datos"
fin si

```

La codificación en JS es:

```

let max = Number.MIN_SAFE_INTEGER;
let huboDatos = false;
let resp = prompt("¿quiere ingresar datos? S/N");
while (resp === "S"){
    let dato = parseInt(prompt("ingrese dato"));
    huboDatos = true;
    if (dato > max){
        max= dato;
    }
    resp = prompt("¿más datos?");
}
if (huboDatos){
    alert("El máximo vale" + max);
}
else {
    alert("No hubo datos");
}

```

3.6.2 Ecuación de 2do grado

Se tienen los 3 coeficientes de una ecuación de 2do. grado. Indicar las raíces (ej. raíces 1 y 3, raíz doble 5, sin raíces reales).

```
let a= parseInt(prompt("a"));
let b= parseInt(prompt("b"));
let c= parseInt(prompt("c"));
let res = "";
let disc = b*b - 4*a*c;
if (disc==0){
    res = "raíz doble" + (-b/(2*a));
}
if (disc <0){
    res = "sin raíces reales";
}
if (disc >0){
    let x1 = (-b + Math.sqrt(disc))/(2*a);
    let x2 = (-b - Math.sqrt(disc))/(2*a);
    res = "Raíces " + x1 + " " + x2;
}
console.log(res);
```

4 Funciones

4.1 Introducción a funciones

Veremos ahora el concepto de función. Pensemos en el siguiente ejemplo: se debe ingresar una cantidad en pesos y se quiere convertir a dólares. Podría pensarlo:

```
leer cotizacionDolar
leer cantidadPesos
monto = convertirADolares(cotizacionDolar, cantidadPesos)
mostrar monto
```

¿Cómo sería el proceso de convertirADolares?

```
proceso convertirADolares(cotizacion, cantidad){
    return cantidad/cotizacion
}
```

Estamos definiendo una función. Una función es un bloque de código (con parámetros o no) asociado a un nombre. Es un conjunto de instrucciones que realizan un tarea o calculan un valor. El uso de funciones hace que se amplíe el lenguaje, por ejemplo ahora sabe “convertirADolares”.

¿Cómo codificamos este ejemplo en JS?

```
let cotizacion = parseInt(prompt("Ingrese cotizacion del dólar"));
let cantidad = parseInt(prompt("Ingrese cantidad"));
alert("total " + convertirADolares(cotizacion, cantidad));

function convertirADolares(cotiz, cant){
    return cant/cotiz;
}
```

Para definir la función se usa la palabra “function”, el nombre y la lista de parámetros. Los parámetros son la información que recibe para trabajar. Son variables que se asignan en la invocación. Luego viene el código delimitado por {}. Para finalizar, la sentencia return refiere el valor a retornar. Si no tiene un valor asociado, devuelve undefined.

O sea, la declaración de una función en JS consiste de:

- el nombre de la función
- la lista de parámetros, entre paréntesis y separados por comas
- las sentencias JS que definen la función entre { }.

La sentencia return especifica el valor devuelto por la función. Los tipos primitivos como los números, son pasados por valor, esto quiere decir que se pasa el valor directamente y si la función cambia el valor del parámetro, este cambio no se refleja globalmente ni en la función que la llamó. Si los parámetros son de tipo no primitivo (como Arrays que veremos más adelante), si la función cambia sus propiedades, los cambios son visibles desde fuera.

4.2 Ejemplos de funciones

4.2.1 Cálculo del área

Veremos cómo aplicar este concepto de función. Queremos hacer un programa que lea la base y la altura de un triángulo y calcule su área.

En pseudocódigo sería:

```
leer baseTriangulo
leer alturaTriangulo
area = calcular área(baseTriangulo, alturaTriangulo)
mostrar area
```

La función área recibe dos parámetros: la base y la altura. Su código podría ser:

```
function area (base, altura)
    return base * altura / 2;
}
```

El código completo en JS sería:

```
let baseTriangulo = parseInt(prompt("Ingrese base"));
let alturaTriangulo = parseInt(prompt("altura"));
alert("area " + area(baseTriangulo, alturaTriangulo));

function area(base,altura){
    return base*altura/2;
}
```

Los parámetros se asignan en el mismo orden de la invocación de la función. Si a la función se la llama con menos parámetros que los que tiene definidos, los demás son “undefined”. Si le paso más parámetros de los que tiene, no los usa.

4.2.2 Perímetro

Hacer una función que reciba los 3 lados de un triángulo y calcule su perímetro.

```
function perimetro(a,b,c){
    return a+b+c;
}
```

4.2.3 Número par

Hacer función que reciba un número y retorne verdadero si es par.

```
function esPar(num){
    return num%2==0;
}
```


4.2.4 Puede votar

Hacer función que reciba un edad y retorne verdadero si la persona puede votar, esto es, tiene 18 o más.

```
function puedeVotar(edad){  
  return edad >= 18;  
}
```

4.2.5 Likes

Ana está contenta cuando tiene más de 50 “me gusta” en una foto de Facebook, y además la cantidad de “no me gusta” es menor a 10.

Hacer función que reciba la cantidad de “me gusta” y “no me gusta” y retorne verdadero si está contenta o falso en otro caso.

```
function contenta(meGusta, noMeGusta){  
  return (meGusta > 50 && noMeGusta < 10);  
}
```

4.2.6 Números ordenados

Dados 3 números, indicar si están ordenados. Implementar una función que retorne true si se cumple que están ordenados.

```
let i = parseInt(prompt("a "));  
let j = parseInt(prompt("b "));  
let k = parseInt(prompt("c "));  
if (estanOrdenados(i,j,k)){  
  alert("están ordenados!");  
}  
else {  
  alert("No están ordenados");  
}  
function estanOrdenados(a,b,c){  
  return a <= b && b <= c;  
}
```

4.2.7 Signo

Dado un número, devolver 1 si es positivo, 0 si es 0 o -1 si es negativo.

```
function mostrarSigno(a){  
  let signo = 0;  
  if (a>0){  
    signo = 1;  
  }  
  if (a<0){  
    signo = -1;  
  }  
  return signo;  
}
```

Para probarla: `alert(signo(-3));`

Observar que si la función necesita otras variables auxiliares, puede definir las.

Al definir una variable se puede usar `let` o `var`, como se indicó antes. La diferencia es el alcance de las variables. **let** permite declarar variables limitando su alcance al bloque, declaración, o expresión donde se está usando y **var** define una variable global o local en una función sin importar el ámbito del bloque.

5 Más de String

5.1 Profundización de String

Como vimos anteriormente, los strings son utilizados para almacenar y manipular textos. Un string se define como una secuencia de caracteres entre comillas simples o dobles.

Para saber el largo, tenemos la propiedad **length**.

```
> let texto = "hola";  
< undefined  
> texto.length;  
< 4
```

Para poner caracteres especiales en un string, se pone \ antes.

```
> texto = "este es un \"texto\" con comillas"  
< "este es un "texto" con comillas"
```

En general, para utilizar un método (función vinculada a un objeto, se amplía más adelante este concepto) se coloca punto luego de la variable y el nombre del método seguido de paréntesis. Dentro de los paréntesis pondremos información adicional si es requerida por el método.

Hay muchos métodos interesantes. Ejemplo: `indexOf`. Este método devuelve -1 si no está lo buscado o si está, la posición en que se encuentra (comenzando desde 0).

```
> texto = "esta es una frase muy larga";  
< 'esta es una frase muy larga'  
> texto.indexOf("una");  
< 8  
> texto.indexOf("unas");  
< -1
```

Se puede indicar el lugar de comienzo de la búsqueda.

```
> texto.indexOf("a",20);  
< 23  
> texto.indexOf("a",15);  
< 23  
> texto.indexOf("a",5);  
< 10  
> texto.indexOf("a",0);  
< 3
```

También está el método `search`, que es similar, pero no se puede indicar lugar de comienzo.

Puedo pasar a mayúsculas, por ejemplo, si texto vale "hola":

```
> texto.toUpperCase();  
< "HOLA"
```

También está `toLowerCase()`, que pasa a minúsculas y `charAt()` que devuelve el carácter de una posición.

```
> texto.charAt(3);  
< "a"
```

5.2 Ejercicios básicos

5.2.1 Armar cartel

Hacer función que recibe el cargo, nombre y apellido y arme su identificación con el cargo primero, nombre y el apellido en mayúscula.

```
function identificacion(cargo, nombre, ape){  
  return cargo+ " " + nombre + " " + ape.toUpperCase()  
}
```

5.2.2 Armar cartel largo máximo

Actualizar la función del cartel, para que si el largo del nombre y apellido supera 20 caracteres, omita el nombre.

```
function identificacion(cargo, nombre, ape){  
  let cartel = cargo + " " + nombre + " " + ape;  
  if((nombre + " " + ape).length > 20){  
    cartel = cargo+ " " + ape;  
  }  
  return cartel;  
}
```

5.2.3 Contar cantidad de una letra dada de una frase

Hacer una función que reciba una frase y una letra, y cuente la cantidad de veces que aparece dicha letra.

```
function cantidadLetra(frase, letra){
  let cant = 0;
  let i = 0;
  for (i = 0; i < frase.length; i++){
    if (frase.charAt(i)==letra){
      cant++;
    }
  }
  return cant;
}
```

5.3 Otros métodos útiles de String: slice, substring, substr

Para cortar un string tenemos los métodos:

- slice(start, end) Toma desde las posiciones start (comienzo) inclusive a end (final) exclusive. Si son negativos, arranca de atrás.
- substring(start, end). Es igual al slice, pero no permite negativos
- substr(start, length). Es igual al slice, pero el 2do. parámetro da el largo.

Veamos ejemplos:

```
> let texto = "esta es una frase larga";
< undefined
> texto.substring(1,5);
< "sta "
> texto.substr(1,5);
< "sta e"
> texto.slice(1,5);
< "sta "
> texto.slice(-3,-1);
< "rg"
```

5.4 Más ejercicios de String

5.4.1 Dar vuelta palabra

Hacer función que de vuelta una palabra:

```
function darVuelta(frase){
  let nueva = "";
  let prim = frase.charAt(0);
  for (let i = frase.length-1; i>=0; i--){
    nueva = nueva + frase.charAt(i);
  }
  return nueva;
}
```

5.4.2 Indicar si es fin de semana

Implementar una función que reciba el nombre de un día y retorne si es fin de semana.

```
function esFinDeSemana(dia){
  return (dia==="sabado"|| dia==="domingo");
}
```

5.4.3 Palíndromo

Implementar una función que reciba una palabra e indique si la misma es palíndromo, o sea, si se lee igual de izquierda a derecha que de derecha a izquierda. Asumir que viene todo en minúsculas y sin espacios.

```
function esPal(frase){
  let esP = true;
  for (let i = 0; i < frase.length/2; i++){
    if (frase.charAt(i) != frase.charAt(frase.length-i-1)){
      esP = false;
    }
  }
  return esP;
}
```

Podemos probarlo:

```
let frasesita ="anitalavalatina";
console.log(esPal(frasesita));
```

5.4.4 Ocurrencias de primera letra

Dada palabra, decir cuántas veces aparece su primera letra.

```
function cantVeces(frase){
  let veces = 0;
  let prim = frase.charAt(0);
  for (i = 0; i < frase.length; i++){
    if (frase.charAt(i) === prim){
      veces++;
    }
  }
  return veces;
}
```

6 Arrays

6.1 Introducción a Array

Supongamos que tenemos las notas de un grupo de estudiantes y queremos calcular su promedio. Esto ya lo sabemos hacer. Así, una versión en pseudocódigo sería:

```
suma = 0
cantidad = 0;
mostrar ("¿algo a ingresar?")
leer resp
mientras (resp == "S")
    leer nota
    suma = suma + nota
    cantidad++
    mostrar ("¿algo más?")
    leer resp
fin mientras
mostrar suma/cantidad
```

En JS:

```
let suma = 0;
let cant = 0;
let resp = prompt("¿algo a ingresar?");
while (resp=="S"){
    let nota = parseInt(prompt("¿nota? "));
    suma = suma + nota;
    cant++;
    resp = prompt("¿algo más?");
}
alert("promedio "+ suma/cant);
```

Si quisiera además, luego de imprimir el promedio, indicar cuántos estudiantes estuvieron por encima de dicho promedio, ¿cómo lo haríamos?

Dado que necesita el promedio, primero debo calcularlo y luego hacer el cálculo. Esto implicaría que se deben ingresar dos veces todos los datos o almacenarlos de alguna forma. No podría usar variables “sueltas” porque no sé la cantidad de datos a ingresar. Tendría que tener alguna forma de nombrar genéricamente variables, por ejemplo: nota1, nota2,

Tenemos la estructura array. Un array es un conjunto de datos. Es un tipo de variable que permite guardar varios datos. Cada dato es almacenado en una posición específica y numerada del array. El número correspondiente a cada lugar se llama índice o posición.

Ejemplo:

notas índice	dato
0	23
1	45
2	54
3	15

En este ejemplo, el array contiene 4 elementos. El índice comienza en 0. Para obtener el valor de la segunda posición se debe indicar el nombre del array y la posición:

```
notas[1]
```

Para modificar ese valor, por ejemplo, por el número 56:

```
notas[1] = 56;
```

¿Como lo definimos en JS? El array del ejemplo podría ser:

```
let notas= [23,45,54,15];
```

Para definir uno vacío:

```
let datos = [];
```

Para saber el largo del array: `datos.length`. La propiedad **length** retorna un número más que el índice mayor del array.

Para agregar un elemento: **datos.push(valor);**

Para eliminar el último elemento se puede utilizar **datos.pop();**

Ejemplo en JS:

```
let datos =[];  
datos.push(2);  
datos[datos.length]=15;  
datos[datos.length]=23;  
alert(datos);
```

Podría dejar “huecos”, que quedan undefined:

```
let datos =[];  
datos.push(2);
```



```
datos[datos.length]=15;  
datos[datos.length+3]=23;  
alert(datos[3]); // da undefined.
```

6.2 Carga y recorrida de array

6.2.1 Carga

¿Cómo cargar el array? Podríamos pedirle al usuario que ingrese datos.

```
let datos = [];  
let resp = "S";  
while (resp == "S"){  
    let valor = parseInt(prompt("ingrese dato"));  
    datos.push(valor);  
    resp= prompt("¿ Seguir?").toUpperCase();  
}  
alert (datos);
```

6.2.2 Recorrida de array

Hay varias formas de recorrer un array.

6.2.2.1 Recorrida con for “tradicional”

Una forma de recorrer es utilizando for, con control “explícito” del índice. El ejemplo muestra cada posición y el valor.

```
// mostrar los elementos a mano  
for (let i=0; i< datos.length; i++){  
    alert("elemento "+i+ " vale "+ datos[i]);  
}
```

6.2.2.2 Recorrida con for in

El mismo ejemplo utilizando "for in". La sentencia for...in itera sobre todos los elementos de un objeto, en un orden arbitrario. Para cada uno de los elementos, se ejecuta las sentencias especificadas.

Dado que el orden de iteración es arbitrario, la iteración sobre un array no accederá a los elementos en orden numérico. Por ello, es mejor utilizar el for tradicional con un índice numérico cuando se itere sobre arrays.

```
// mostrar los elementos a mano
for (let i in datos){
  alert("elemento "+i+ " vale "+ datos[i]);
}
```

6.2.2.3 Recorrida con for of

Una versión similar utilizando "for of". Va ejecutando las sentencias de cada iteración con el valor del elemento que corresponda.

```
// mostrar los elementos a mano
for (let valor of datos){
  alert( "el contenido es "+ valor);
}
```

6.3 Ejercicios básicos de array

6.3.1 Suma

Hacer una función que retorne la suma de un array.

```
function suma(datos){
  let suma=0;
  for (let i =0; i< datos.length; i++){
    suma = suma+ datos[i];
  }
  return suma;
}
```

Para probar:

```
let datos= [1,132, 335,-6,12,54,14,2];
alert(suma(datos));
```

6.3.2 Promedio

Utilizando la función de suma, implementar la función promedio.

```
function promedio(datos){
  return suma(datos)/datos.length;
}
```

6.3.3 Máximo

Implementar una función que retorne la posición del máximo de un array dado.

```
function posicionMaximo(lista){
  let max = lista[0];
  let pos = 0;
  for (let i = 1; i < lista.length; i++) {
    if (lista[i] > max) {
      max = lista[i];
      pos = i;
    }
  }
  return pos;
}
```

6.3.4 Desviación estándar

A partir de los datos de notas, calcularemos la desviación estándar. Podemos utilizar la función de promedio que ya implementamos.

Explicaremos en detalle cómo se calcula la desviación estándar.

Supongamos 3 grupos de estudiantes, cada uno con 4 alumnos. Si las notas son: Grupo A: (0, 0, 14, 14), Grupo B: (0, 6, 8, 14) y Grupo C: (6, 6, 8, 8). Cada grupo tiene un promedio de 7, pero su dispersión es distinta. Esta idea tiene que ver con la noción de varianza. Da una idea de cuán dispersos están los datos. Si la varianza es baja, están muy agrupados alrededor de la media. Se usa habitualmente para comparar conjuntos de datos. La desviación estándar es la raíz de la varianza. La fórmula de la desviación es:

$$\text{Desviación estándar} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

Se calcula la suma del cuadrado de la diferencia de cada número con el promedio, se divide sobre la cantidad de elementos y a ese valor final, se le calcula la raíz cuadrada.

Por ejemplo, para el Grupo A: notas: 0 0 14 14

numerador: $(0-7)^2 + (0-7)^2 + (14-7)^2 + (14-7)^2 = 49+49+49+49= 196$

$196/4 = 49$

desviación estándar: 7

Para el Grupo B: notas 0 6 8 14

numerador: $(0-7)^2 + (6-7)^2 + (8-7)^2 + (14-7)^2 = 49+1+1+49= 100$

$100/4 = 25$

desviación estándar: 5

Para el Grupo C: notas 6 6 8 8

numerador: $(6-7)^2 + (6-7)^2 + (8-7)^2 + (8-7)^2 = 1+1+1+1 = 4$

$4/4 = 1$

desviación estándar: 1

Sus desviaciones estándar poblacionales son 7, 5 y 1, respectivamente. La tercera población tiene una desviación mucho menor que las otras dos porque sus valores están más cerca de 7.

```
function desviacion(datos){
  let numerador = 0;
  let prom = promedio(datos);
  for (let i = 0; i < datos.length; i++){
    numerador += Math.pow((datos[i]- prom),2);
  }
  let desviacion = Math.sqrt(numerador/datos.length);
  return desviacion;
}
```

6.4 Búsqueda en array

6.4.1 Búsqueda manual

Se desea implementar una función para buscar un valor en un array, recibe por parámetros el array y el valor a buscar. Retorna true si lo encuentra.

Ejemplo: se tiene:

```
let datos= [1,335,-6,12,54,14,2];
alert(buscar(datos, 132)); // dará false
```

Se podría implementar utilizando la estructura for:

```
function buscar(lista, valor){
  let esta = false;
  for(let pos =0; pos<lista.length && !esta; pos++ ){
    if (lista[pos] == valor) {
      esta = true;
    }
  }
  return esta;
}
```

Otra posible implementación de la función buscar puede ser:

```
function buscar(lista, valor){
  let termine = false;
  let esta = false;
  let pos = 0;
  while (!termine) {
```

```

    // verifico si ya recorrí todo
    if (pos == lista.length) {
        termine = true;
    }
    else {
        if (lista[pos] == valor) {
            esta = true;
            termine = true;
        }
    }
    pos = pos + 1;
}
return esta;
}

```

6.4.2 Otras formas de buscar

El método `indexOf` devuelve la posición de un elemento dado o -1 si no está.

Ejemplo:

```

let dat1 =[12,23,434];
alert (dat1.indexOf(2223)); // sale -1

```

También está el método `includes`, que retorna `true` si el valor está o `false` si no está.

Ejemplo:

```

let dat1 =[12,23,434];
alert (dat1.includes(2223)); // sale false

```

6.5 Ordenación de array

6.5.1 Ordenación por selección

Hay diferentes algoritmos para ordenar. Aquí se presenta `sort` (ordenación) por selección. Este algoritmo inicialmente ubica el máximo y lo intercambia con el último elemento. Este elemento ya quedará ubicado en su posición final. Se repite este proceso sin considerar en cada paso estos elementos finales reubicados.

```

function ordenar(datos){
  for (let i = datos.length-1; i >=0; i--){
    let max = datos[i];
    let posMax = i;
    for (let j = 0; j <=i; j++){
      if (datos[j]> max){
        max = datos[j];
        posMax = j;
      }
    }
    // intercambio
  }
}

```

```
let aux = datos[i];
datos[i]=datos[posMax];
datos[posMax]= aux;
}
}
```

Para probar:

```
//Ordenacion array
let datos= [1,335,-6,12,54,14,2,21];

alert(datos); // antes
ordenar(datos);
alert(datos); // después
```

Observar que ordena sobre el propio array, pues se pasa por referencia. Este concepto lo veremos en los próximos capítulos.

6.5.2 Ordenación usando método Sort

Probemos este código:

```
// defino array
let datos =[8,20, 3,-4,8];
alert(datos.sort()); // Sale: -4,20,3,8,8
```

Observar cuidadosamente el resultado. Los ordena como Strings

Es posible establecer el criterio usando una función de expresión, donde defino el criterio. Siendo a y b dos elementos a comparar, dicha función que se llama automáticamente al ordenar y debe devolver un valor negativo si el elemento a debe ir ubicado antes del elemento b, 0 si son iguales y un valor positivo en otro caso.

Si son números, al hacer la resta a-b se obtiene el resultado deseado, como se muestra a continuación.

```
datos.sort(function (a,b){
    return a-b;
});
console.log(datos); // sale -4,3,8,8,20
```

6.6 Más ejercicios de array

6.6.1 Largo de palabras

Crear una función que a partir de un array con palabras, cree otro que tenga en cada posición el largo de la respectiva palabra.

```
function verLargosPalabras(palabras){
  let retorno = [];
  for(let pos=0; pos<=palabras.length-1; pos++){
    palabra = palabras[pos];
    retorno[pos] = palabra.length;
  }
  return retorno;
}
```

6.6.2 Intercalar

Hacer una función que dados dos arrays, donde el primero se llama nombres y el segundo apellidos, cree uno nuevo que tenga intercalados cada nombre y apellido. Ejemplo: Si se tiene como nombres: Juan y Ana y apellidos: González y López, el nuevo array contendrá Juan, González, Ana, López.

```
function intercalar(nombres,apellidos){
  let retorno = [];
  for(let pos=0; pos<=nombres.length-1; pos++){
    let nombre = nombres[pos];
    let apellido = apellidos[pos];
    retorno.push(nombre);
    retorno.push(apellido);
  }
  return retorno;
}
```

6.6.3 Eliminar

Escribir una función que reciba un array y un número entero. La función debe eliminar el elemento del array que se encuentra en la posición dada por el entero recibido.

```
function eliminar(datos,pos){
  datos.splice(pos,1);
}
```

6.6.4 Ejercicio: Comparación

Escribir una función que compare 2 arrays con números recibidos como parámetro y devuelva true si son iguales y false si son diferentes. Para ser iguales, tiene que tener todos los mismos datos, posición a posición.

```
function comparo(datos1, datos2){
  let result = true;
  if (datos1.length != datos2.length){
    result = false;
  }
  if (result){
    for (let i = 0; i < datos1.length&& result; i++){
      if (datos1[i] != datos2[i]){
        result = false;
      }
    }
  }
  return result;
}
```

Pruebo:

```
let datos1= [1,335,-6,12,54,14,2];
let datos2= [1,335,-6,12,54,14,2];
alert(comparo(datos1, datos2));
```

6.6.5 Más cercano

Escribir una función que reciba un array cargado con números y un número. La función debe devolver el valor más cercano (por exceso o por defecto) al número recibido.

```
function masCercano(datos,numero){
  let valor = 0;
  let minDiferencia = Number.MAX_VALUE;
  for(let i=0; i< datos.length; i++){
    if(Math.abs(datos[i]-numero)<minDiferencia){
      minDiferencia = Math.abs(datos[i]-numero);
      valor = datos[i];
    }
  }
  return valor;
}
```


7 Introducción a Web, HTML y CSS

7.1 Introducción a Web

Trabajaremos con varios conceptos vinculados a Internet y Web.

7.1.1 Resumen de definiciones

- **Internet:** Su nombre proviene por la conjunción de dos términos del inglés; Interconnection y Network, que significan interconexión y red respectivamente. Esta interconexión se realiza a través de líneas telefónicas, por satélites, antenas de microondas y cables de fibra óptica entre otros.
- **Protocolo:** con tantas computadoras conectadas a Internet, se necesita un lenguaje común para "hablar" con las demás. Ese lenguaje es una serie de protocolos que describen la información que se envía a través de la red y que explican qué hacer con ella cuando llega a su destino.
- **TCP/IP:** el protocolo fundamental se llama TCP/IP (Protocolo de Control de Transmisión/Protocolo Internet). La principal característica del TCP/IP es que establece la comunicación por medio de paquetes de información. Cuando un computador quiere mandar a otro un fichero de datos, lo primero que hace es partirlo en trozos pequeños y posteriormente enviar cada trozo por separado.
- **IP:** Cada computadora que se conecta a Internet se identifica por medio de una dirección IP. Se compone de 4 números entre 0 – 255 y separados por puntos. Ej. 155.210.13.45. Esta composición de 32 bits, permite generar 4300 millones de direcciones únicas. Un usuario de Internet no necesita conocer ninguna de las direcciones IP. Las manejan las computadoras en sus comunicaciones mediante el Protocolo TCP/IP. A medida que Internet y el número de personas que lo utilizan crecen exponencialmente, crece también la necesidad de contar con más direcciones IP.
- **IPv6:** IPv6 permite que un mayor número de usuarios y dispositivos se comuniquen a través de Internet utilizando números de mayor tamaño para crear direcciones IP. Las direcciones IPv6 se componen de 128 bits, lo que permite la existencia de aproximadamente 340 billones de direcciones IP únicas.
Ejemplo: 2001:db8:ffff:1:201:02ff:fe03:0405
- **DNS:** Para nombrar de alguna manera a las computadoras en Internet lo hacemos por medio de los Nombres de Dominio. Enmascaran las direcciones IP.
Ej. - www.google.com
- www.ort.edu.uy
El número de palabras no es fijo (2, 3, 4...) y están separadas por puntos. El Sistema de Nombres de Dominio (DNS) ayuda a los usuarios a navegar en Internet. Como las direcciones IP son difíciles de recordar, el DNS permite usar una cadena de letras (el Nombre de Dominio) para que por ejemplo se pueda escribir www.icann.org en vez de 192.0.34.65. El DNS traduce el Nombre de Dominio a la dirección IP que le corresponde y lo conecta al sitio web que desea. La ICANN (Internet Corporation for Assigned Names and Numbers) es

la responsable de la coordinación y administración de los elementos técnicos del DNS para garantizar una resolución unívoca de los nombres. Las convenciones son: La última palabra del Nombre de Dominio representa:

.com	entidades comerciales
.org	organizaciones diversas
.edu	instituciones educativas
.net	organismos relacionados con la red

También el país a donde pertenece el sitio:

.uy	Uruguay
.es	España
.ar	Argentina
.br	Brasil

Actualmente, también se encuentran habilitados los dominios que no explicitan el tipo de organización, los cuales son de uso general, por ejemplo: www.midominio.uy

- **URL:** cada sitio Web, y cada página de ese sitio tiene una dirección única. Esa dirección se llama URL (Localizador Uniforme de Recursos). Es un apuntador hacia una sección específica de información en Internet. Ej: <http://www.ort.edu.uy/cursos/curweb.html>
- **Navegadores:** La gente puede acceder a documentos (páginas) almacenados en distintas computadoras alrededor del mundo por intermedio de un software llamado navegador (browser). El navegador se usa para ver y navegar páginas de la Web.

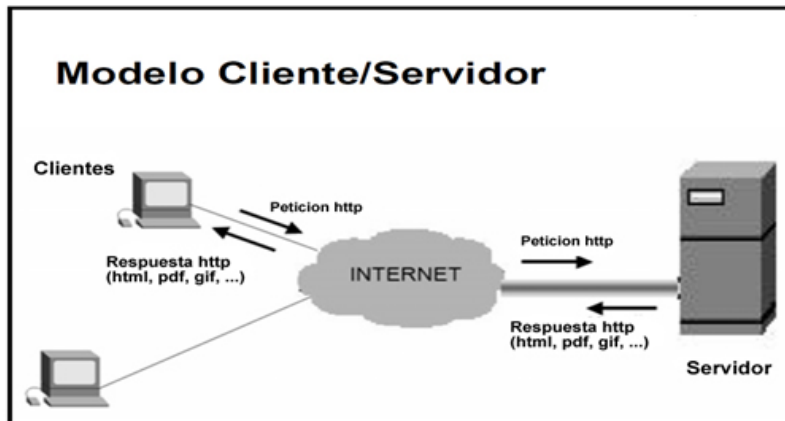
7.1.2 Arquitectura Web

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores. Las posibilidades que ofrece Internet se denominan servicios. Algunos de estos servicios son:

- Correo Electrónico
- FTP (File Transfer Protocol)
- Grupo de Noticias
- Servicios de Telefonía
- World Wide Web (W.W.W.)

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor realizando conexiones con el cliente generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse el protocolo HTTP.

Dada una dirección URL el navegador se comunica con el servidor Web mediante el protocolo HTTP. Despliega documentos Web que recibe del servidor en el sistema que el usuario utilice. Reconoce los tipos de archivo que recibe y los presenta utilizando programas adecuados denominados plug-ins (si conoce el tipo de archivo)



La Web es realmente un sistema de protocolos intercambiados entre un cliente (PC) y un servidor (la aplicación del sistema central que reparte páginas web) para que los documentos puedan compartirse entre computadoras y otra gran variedad de dispositivos en la red.

Desde 1990, el protocolo HTTP (Protocolo de transferencia de hipertexto) es el protocolo más utilizado en Internet. El propósito del protocolo HTTP es permitir la transferencia de archivos (principalmente, en formato HTML) entre un navegador (el cliente) y un servidor web localizado mediante una cadena de caracteres denominada dirección URL. HTTPS es una combinación del protocolo HTTP y protocolos criptográficos. Se emplea para lograr conexiones más seguras en la Web, generalmente para transacciones de pagos o cada vez que se intercambie información sensible (por ejemplo, claves).

W.W.W. es un sistema gráfico de información de hipertexto, distribuido, global, interactivo, dinámico e independiente de la plataforma, que funciona en Internet. Proporciona la capacidad de incorporar gráficos, sonido y video.

La principal actividad del W3C (Consorcio World Wide Web) es desarrollar protocolos y standards que aseguren el crecimiento a largo plazo para la Web. Está integrado por MIT, INRIA y DARPA, entre otros.

7.1.3 Otros conceptos relevantes: Accesibilidad e Internacionalización.

La accesibilidad web se centra en garantizar un acceso equivalente para todas las personas. Beneficia a todos. Algunos casos pueden ser:

- limitaciones situacionales temporales
- uso de dispositivos móviles, televisores y otros canales de acceso
- uso de computadoras viejas, con poco ancho de banda y otras limitaciones
- novatos en las computadoras, en la Web o en un sitio web particular
- no dominio del idioma de un sitio web en particular

Son ejemplos de accesibilidad:

- subtítulos sobre contenido de audio y multimedia (audición)
- diseño claro y consistente (discapacidades cognitivas y de aprendizaje)
- soporte de teclado (discapacidades físicas/ no uso de mouse);
- alternativas de texto (para personas con discapacidad visual y que utilizan lectores de pantalla)

En relación a internacionalización, se trata de asegurar que cualquier contenido o aplicación que diseñe o desarrolle esté preparado para respaldar las características internacionales.

Los caracteres representan las letras del alfabeto, puntuación y otros símbolos. Unicode es un set de caracteres universal, un standard que define los caracteres para usar en las computadoras. Se recomienda usar UTF-8 como codificación de caracteres.

7.2 Introducción a HTML

JS (JavaScript), HTML (Hyper Text Markup Language) y CSS (Cascade Style Sheets) son 3 tecnologías fundamentales del desarrollo Web. Ya vimos las nociones de JS, ahora veremos conceptos básicos de HTML (que tiene que ver con el contenido) y luego de CSS (que tiene que ver con el estilo o aspecto del documento). JS le da la parte de funcionalidad.

Fue desarrollado por Tim Barners-Lee en el CERN y popularizado por el navegador Mosaic. Es un lenguaje que describe la estructura general del contenido de un documento, no el aspecto en sí de la página en pantalla.

7.2.1 Primera página de ejemplo

Veamos este ejemplo. Codificar en Notepad++ o VisualCode y guardarlo como index.html:

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Ejemplo con varios elementos</title>
</head>
<body>
  <p>Hola</p>
</body>
</html>
```

Probarlo en Chrome. Tenemos una página.

Aclararemos qué se escribió. Haremos una descripción inicial y luego profundizaremos.

Los archivos HTML (html) son archivos de texto plano (ASCII), pueden leerse con cualquier editor.

Al comienzo se indica que es un documento html y el lenguaje. Lang refiere al lenguaje, por ejemplo “es” es español, “en” es inglés. Siempre debería estar.

El contenido se estructura mediante etiquetas (tags). Las etiquetas HTML son comandos escritos entre signos < >. Existen versiones de apertura y de cierre para la mayoría de las etiquetas, pero no todas. Tanto las etiquetas de apertura como las de cierre usan el mismo comando pero la etiqueta de cierre agrega una /.

ej.: <html> </html>
“abre” “cierra”

En el ejemplo anterior, el párrafo “Hola”, va entre etiquetas p.

Las etiquetas no son sensibles a mayúsculas o minúsculas, y el espaciado y la nueva línea se ignora. Por ejemplo: se verá igual:

```
<p>Hoy es viernes.</p>
```

que:

```
<p>Hoy es      viernes.</p>
```

7.2.2 Análisis detallado de la página

Veremos en detalle lo que escribimos en la página.

- 1) **<!doctype html>**: es la secuencia de caracteres más corta que cuenta como un doctype válido. En versiones anteriores de HTML tenía más parámetros.
- 2) **<html></html>**: El elemento <html>. Este elemento engloba todo el contenido de la página y es conocido como el elemento raíz.
- 3) **<head></head>**: Incluye entre otros palabras clave y la descripción de la página que se quiere mostrar en los resultados de búsqueda, así como la hoja de estilo para formatear el contenido, declaraciones de codificación de caracteres, entre otros elementos.
- 4) **<meta charset="utf-8">**: Este elemento establece que el documento HTML usará la codificación utf-8, que incluye la gran mayoría de caracteres de todos los lenguajes conocidos.
- 5) **<title></title>**: Este elemento establece el título de la página, que aparece en la pestaña/ventana del navegador cuando la página se carga y se utiliza para describir la página cuando se la agrega a los marcadores/favoritos.
- 6) **<body></body>**: El elemento <body> contiene todo el contenido que se mostrará a los usuarios cuando visiten la página ya sea texto, imágenes, vídeos, música, etc.

Si se desea colocar comentarios se utiliza este formato: **<!-- comentario -->**.

7.2.3 Encabezados y párrafos

En este ejemplo mostraremos varios de los elementos que se pueden incluir en una página.

En general, los textos están estructurados en párrafos, con encabezados. En HTML hay 6 tipos de encabezados (heading): h1 a h6 (h1 es el de mayor destaque).

NOTA: también existen los tags semánticos que permiten organizar el contenido de una página. Algunos de ellos son : section, article, footer. Se detallan más adelante.

Ejemplos:

```
<h1> .... </h1>
<h2> .... </h2>
```

```
...  
<h6> .... </h6>
```

El código quedaría:

```
<!doctype html>  
<html lang="es">  
<head>  
  <meta charset="utf-8">  
  <title>Ejemplo con varios elementos</title>  
</head>  
<body>  
  <!-- Tipos de encabezados -->  
  <h1>Ejemplo de h1</h1>  
  <h2>Ejemplo de h2</h2>  
  <h3>Ejemplo de h3 </h3>  
  <h4>Ejemplo de h4 </h4>  
  <h5>Ejemplo de h5</h5>  
  <h6>Ejemplo de h6</h6>  
  <!-- Párrafo-->  
  <p>Hola!</p>  
</body>  
</html>
```

¿Por qué se necesita estructura? Entre otras razones por que:

- a) las personas miran una página web rápidamente para buscar información relevante, a menudo leyendo sólo los titulares
- b) los motores de búsqueda indexan la página tomando que los contenidos de los heading son importantes
- c) hay personas con problemas de visión que oyen las páginas, con un “screen reader”. Ese software puede dar un resumen del texto sobre los headings, sino hay, debe leer en voz alta todo el texto

Hay elementos que tienen información semántica. Por ejemplo, en la vida diaria, un semáforo en rojo implica que se debe detener. En HTML, h1, implica que se debe destacar, y el navegador por defecto lo resaltará (aunque se pueda modificar).

7.2.4 Listas

Podemos incluir listas. Una opción es usando el tag ul (que define una lista no ordenada) y el tag li (ítem de lista). Veamos el código.

El código queda:

```
<!doctype html>  
<html lang="es">  
<head>  
  <meta charset="utf-8">
```

```

<title>Ejemplo con varios elementos</title>
</head>
<body>
    <!-- Tipos de encabezados -->
    <h1>Ejemplo de h1</h1>
    <h2>Ejemplo de h2</h2>
    <h3>Ejemplo de h3 </h3>
    <h4>Ejemplo de h4 </h4>
    <h5>Ejemplo de h5</h5>
    <h6>Ejemplo de h6</h6>
    <!-- Párrafo-->
    <p>Hola!</p>

    <hr> <!-- pone una línea -->
    <p>Este es otro párrafo</p>
    <!-- lista -->
    <ul> Esta es una lista
        <li>Item Lista 1</li>
        <li>Item Lista 2</li>
        <li>Item Lista 3</li>
        <li>Item Lista 3</li>
    </ul>
</body>
</html>

```

7.2.5 Tablas

Una tabla es una forma estructurada de presentar información, en forma de filas y columnas. Las tablas se deben usar para información, no para “acomodar” la presentación (pues en ese uso reducen accesibilidad y no son automáticamente “responsive”, o sea, probablemente se verían mal en dispositivos diferentes).

Las tablas se definen con el tag **<table>**, puede tener un encabezado, filas y pie. Las etiquetas **<tr>** ("table row") definen cada fila de la tabla y encierran todas las columnas. La etiqueta **<td>** ("table data cell") define cada una de las columnas de las filas, aunque realmente HTML no define columnas sino celdas de datos. Los tags **<th>** indican que se trata de celdas que sirven como encabezado de tabla y suelen visualizarse en negrita y con sus datos en forma centrada.

Se pueden agregar en forma opcional las etiquetas **thead**, **tbody** y **tfoot**. **thead** agrupa todo lo que va en el encabezado o header. **tfoot** agrupa todo lo que va en el pie. **tbody** agrupa el resto del contenido. Estas agrupaciones son útiles para la aplicación de estilos y para los casos en los cuales necesitemos imprimir tablas largas, teniendo repetir la información de la cabecera y el pie en todas las páginas que contengan datos de la tabla.

Veremos en el ejemplo los distintos tags:

El código quedaría:

```
<head>
  <meta charset="utf-8">
  <title>Ejemplo con varios elementos</title>
</head>
<body>
  <!-- Tipos de encabezados -->
  <h1>Ejemplo de h1</h1>
  <h2>Ejemplo de h2</h2>
  <h3>Ejemplo de h3 </h3>
  <h4>Ejemplo de h4 </h4>
  <h5>Ejemplo de h5</h5>
  <h6>Ejemplo de h6</h6>
  <!-- Párrafo-->
  <p>Hola!</p>

  <hr> <!-- pone una línea -->
  <p>Este es otro párrafo</p>
  <!-- lista -->
  <ul> Esta es una lista
    <li>Item Lista 1</li>
    <li>Item Lista 2</li>
    <li>Item Lista 3</li>
    <li>Item Lista 3</li>
  </ul>
  <!-- Tabla-->
  <table>
  <thead>
    <caption>Este es el título o caption de la tabla</caption>
    <tr>
      <th>Mes</th>
      <th>Gastos</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Enero</td>
      <td>$1000</td>
    </tr>
    <tr>
      <td>Febrero</td>
      <td>$2000</td>
    </tr>
  </tbody>
  <tfoot> <!-- la ubicación del tfoot varía, en HTML5 puede ir luego del tbody -->
    <tr>
      <td>Total</td>
```



```

                <td>$3000</td>
            </tr>
        </tfoot>

    </table>
</body>
</html>

```

7.2.6 Links

Se pueden incluir links en la página. Es una conexión desde un recurso a otro, por ej. desde una parte de un documento a otro documento.

Un link se crea envolviendo el texto (u otro contenido) que se quiere convertir en un link dentro de un elemento <a>, y dándole el atributo href, que contiene hacia dónde quiero apuntar.

Agregamos ahora que si se clickea en el link, lleve al comienzo de la página.

El código es:

```

<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Ejemplo con varios elementos</title>
</head>
<body>
    <!-- Ejemplos de links -->
    <a id= "tope"></a> <!-- defino ancla de nombre tope-->

    <!-- Tipos de encabezados -->
    <h1>Ejemplo de h1</h2>
    <h2>Ejemplo de h2</h2>
    <h3>Ejemplo de h3 </h3>
    <h4>Ejemplo de h4 </h4>
    <h5>Ejemplo de h5</h5>
    <h6>Ejemplo de h6</h6>
    <!-- Párrafo-->
    <p>Hola!</p>

    <hr> <!-- pone una línea -->
    <p>Este es otro párrafo</p>
    <!-- lista -->
    <ul> Esta es una lista
        <li>Item Lista 1</li>
        <li>Item Lista 2</li>
        <li>Item Lista 3</li>
        <li>Item Lista 3</li>
    </ul>

```

```

</ul>
<!-- Tabla-->
<table>
<thead>
    <caption>Este es el título o caption de la tabla</caption>

    <tr>
    <th>Mes</th>
    <th>Gastos</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>Enero</td>
        <td>$1000</td>
    </tr>
    <tr>
        <td>Febrero</td>
        <td>$2000</td>
    </tr>
</tbody>
<tfoot> <!-- la ubicación del tfoot varía, en HTML5 puede ir luego del tbody -->
    <tr>
        <td>Total</td>
        <td>$3000</td>
    </tr>
</tfoot>

</table>
<!-- link al comienzo -->
<a href="#tope">ir al tope</a>

</body>

</html>

```

7.2.7 Otros elementos: caja de texto, combo, botón

Agregar estos elementos en la página, luego de la tabla. Analizar qué es cada uno.

```

<label for="nombre">Ingrese el nombre
<input type="text" id="nombre" name="nombre"></label>

<label for="edad">Edad
<input type="number" id="edad"></label>

<!------- Horario (combo - select/option) ----->
<label for="preferencia">Horario preferido </label>

```

```

<select name="horario" id="preferencia" >
  <option value="mañana" selected >De mañana</option>
  <option value="tarde">De tarde</option>
  <option value="noche">De noche</option>
</select>

<button type="button"> Consultar</button>

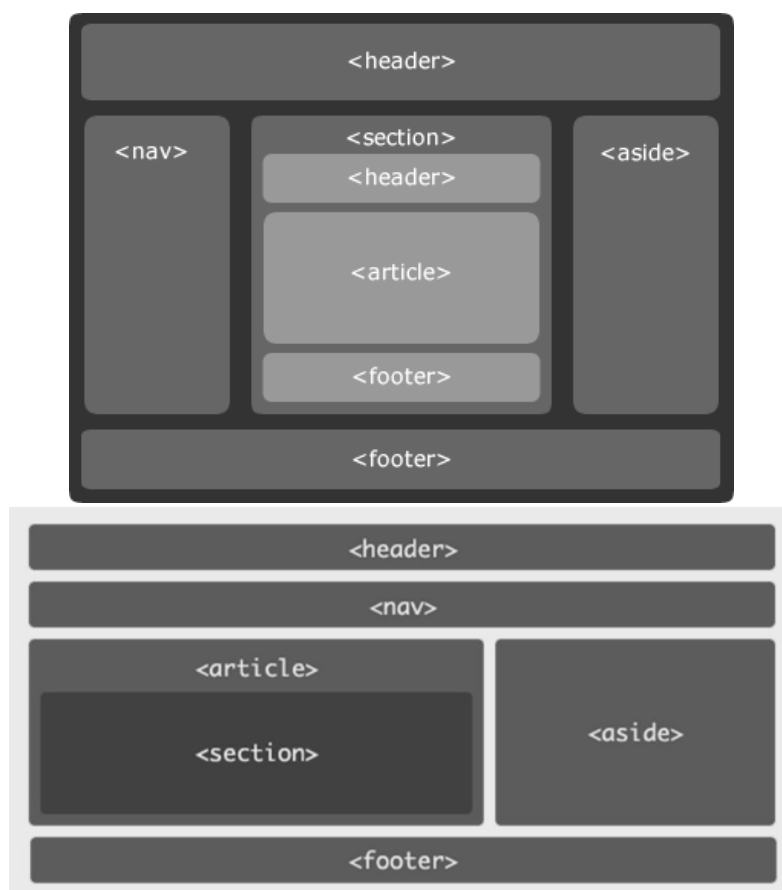
```

Estos ejemplos mostraron una página estática, que siempre se muestra igual. Más adelante veremos cómo interactuar.

7.2.8 Tags semánticos

Además de las partes individuales de una página, como por ejemplo un párrafo, hay elementos en HTML para definir áreas específicas, como un encabezado o barra de navegación.

Se muestran ejemplos de posibles organizaciones de una página con los elementos citados:



Los contenidos típicos de cada área son:

header: usualmente atraviesa toda la página, con un gran encabezado y logo. También contiene información común al sitio

nav (navigation bar): en general son menús y links

main: principal

section: sección

article: artículo

aside: información periférica, relacionada generalmente con el contenido principal (por ej. si en la parte principal se ve una noticia, en el aside se puede ver la información del autor).

footer: atraviesa toda la página, en general información de contacto, copyright, etc.

Los tags para su representación son:

header: <header>

navigation bar: <nav>

main content: <main> (debe ser único), con varias secciones: <article>, <section>, y <div> (div es elemento no semántico para agrupar).

sidebar: <aside>; a menudo dentro de <main>.

footer: <footer>.

7.3 CSS básico y vinculación con HTML

Las hojas de estilo en cascada o CSS (Cascade Style Sheets) se usan para dar estilo a las páginas, por ejemplo, cambiar el font, color, tamaño y espaciado, dividir entre columnas, etc.

La práctica recomendada es crear otro archivo donde se colocarán las reglas. Para esto, crearemos la carpeta css y dentro colocamos el archivo estilo.css. Para vincularlo con la página agregamos dentro del head:

```
<link rel="stylesheet" href="css/estilo.css">
```

Los navegadores aplican las reglas de CSS al documento. Una regla está formada por:

- selectores (a qué elementos se quiere aplicar la regla, por ejemplo h1) y
- un conjunto de propiedades que se le quiere aplicar al selector, por ejemplo: "color: red".

Ejemplo de archivo estilo.css :

```
p {  
  font-size: 18px;  
}
```

El ejemplo corresponde al selector p, la propiedad tamaño de letra y el valor que tiene: 18 pixeles.

7.3.1 ¿Cómo funciona CSS?

Cuando el navegador despliega un documento, combina el contenido con el estilo. Se hace en etapas. Primero carga el HTML, luego carga el CSS y va formando el DOM (se verá más adelante), que representa el documento en la memoria del computador y combina contenido y estilo.

7.3.2 ¿Dónde ubicarlo?

Como observamos en el ejemplo, se puede poner en archivo aparte (indicándolo dentro del head con `<link rel="stylesheet" href="css/estilos.css">`, que es la forma recomendada. Se puede poner también en el propio documento, con las etiquetas `style` dentro del head o en la propia línea:

```
<p style="color:red;">Este es el primer ejemplo CSS</p>
```

Estas dos últimas formas son menos recomendadas, debido a que complican la lectura del código de la página HTML y no favorecen el buen mantenimiento de la misma.

7.3.3 Ejemplo Introductorio de HTML y CSS

Veamos este ejemplo y luego se explicará en detalle:

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title> Ejemplo Estilos </title>

</head>

<body>
  <h1>Prueba de Estilos <em>CSS</em></h1>
  <h2>Titulo h2 </h2>

  <p>Este es el primer párrafo</p>
  <p>Este es el segundo párrafo de ejemplo</p>
  <p id="párrafoEspecial"> texto con el párrafo especial </p>
  <p class="azul">Este es el cuarto párrafo</p>
  <p>Este es el <em>párrafo</em> final</p>

</body>
</html>
```

Se verá sin formato:

Prueba de Estilos CSS

Titulo h2

Este es el primer parrafo

Este es el segundo párrafo de ejemplo

texto con el parrafo especial

Este es el cuarto párrafo

Este es el *párrafo* final

Observar que agregamos la palabra “class” en el p. Enseguida veremos que representa.

Queremos que la página luzca así:

Prueba de Estilos CSS

Titulo h2

Este es el primer parrafo

Este es el segundo párrafo de ejemplo

texto con el parrafo especial

Este es el cuarto párrafo

Este es el *párrafo* final

El archivo estilo.css que contiene las reglas a usar podría ser:

```
/* los elementos p tendrán su texto centrado y color red */
p{
  text-align: center;
  color: red;
}

/* los elementos que tengan la clase azul, tendrán color blue */
.azul{
  color: blue;
```

```

}

/*el elemento con id parrafoEspecial tendrá tamaño de letra 30 px */
#parrafoEspecial{
    font-size: 30px;
}

/* los elementos em que estén dentro de un p, tendrán color brown y tamaño de letra 40px */
p em{
    color: brown;
    font-size: 40px;
}

/* los elementos h1 y los elementos h2 tendrán texto centrado y color violet */
h1, h2{
    text-align: center;
    color: violet;
}

```

Otras reglas interesantes que podríamos agregar son:

```

/* se puede aplicar la misma regla a varios elementos, se deben separar con "," */
h1, h2, h3, h4{
    color: green;
    font-size: 20px;
}

/* cuando se pase el ratón encima, cambia el color de fondo */
h6:hover {
    background-color: blue;
}

```

Observar que los comentarios se agregan con `/* */`.

El archivo debe vincularse desde la página html.

8 HTML y JS

8.1 Nociones DOM

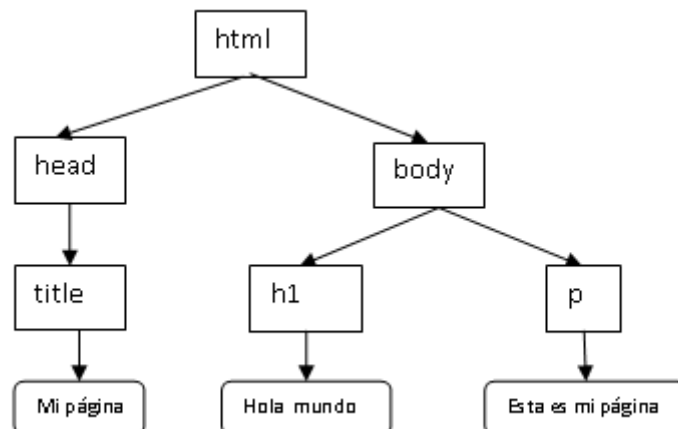
Una página HTML es una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Cuando se carga una página, el navegador genera una estructura que la representa y se puede alterar dinámicamente mediante JS. DOM es la abreviatura de Document Object Model. Es una estructura jerárquica donde existen varios objetos y unos dependen de otros.

Por ejemplo, este código:

```
<html>
<head>
  <title>Mi página</title>
</head>
<body>
  <h1>Hola mundo</h1>
  <p>Esta es mi página.</p>
</body>
</html>
```

se representa:



Cada cajita de esta representación es un nodo. Son los elementos básicos del DOM. Cada nodo puede tener hijos (0, 1, varios). Los del mismo nivel son hermanos.

El nodo principal se llama Document. Hay varios tipos de nodos. Algunos de los más relevantes son:

Document: nodo raíz del que derivan todos los demás.

Element: Cada una de las etiquetas HTML se transforma en un nodo de tipo "Element". Por ejemplo, el nodo raíz tiene 2 hijos element: nodo head y nodo body. La transformación de cada etiqueta

genera 2 nodos: uno de tipo elemento y otro de tipo texto (que es hijo), que contiene el texto que está dentro de la etiqueta.

Una vez que está construido el árbol, se puede acceder desde JS. Para acceder a un nodo puedo hacer a través de su padre o por acceso directo.

La función `getElementById()` es una de las más utilizadas cuando se desarrollan aplicaciones web dinámicas. Se trata de la función para acceder directamente a un nodo y poder leer o modificar sus propiedades.

La propiedad `innerHTML` devuelve el contenido de un elemento HTML y permite cambiarlo.

8.2 Incorporando la interfaz: vínculo HTML y JS

En los ejemplos previos, se puso el código desde un “snippet” y se usó `console.log`, `prompt` y `alert` para mostrar y pedir datos. Ahora veremos cómo pedir los datos desde una página web interactuando con ella, hacer el proceso con código JS similar al que estuvimos probando y mostrar el resultado en la página.

8.3 Ejemplo: Ingresar una frase y pasarla a mayúsculas

El ejercicio es ingresar un texto y al presionar el botón, mostrarlo en mayúsculas en la propia página. Separamos la parte de interfaz de la lógica.

8.3.1 Parte interfaz

El diseño de la página sería:

Ejemplos de prueba

Ingrese la frase

Crearemos el HTML.

```
<!doctype HTML>
<html lang="es">
<head>
  <charset = "UTF-8">
  <title> Ejemplo: leer String y pasarlo a Mayúsculas</title>
</head>
<body>
  <h1>Ejemplos de prueba </h1>
  <label for="frase">Ingrese la frase</label>
  <input type="text" id="frase" >
  <!-- Para poder solicitar el resultado, habrá que presionar el botón -->
  <input type="button" id = "boton" value="Consultar">
  <!-- El resultado se desplegará en un párrafo -->
```

```
<p id="resultado"> </p>
</body>
</html>
```

Nota: si se omite el “for” en el label, al presionar sobre el texto no hará foco en la cajita.

Poner en cada campo su identificación (que no debe estar repetido en ese archivo), para que posteriormente nos podamos referir a él desde JS.

Si lo probamos, vemos el diseño previsto pero falta el código.

8.3.2 Parte lógica

Antes poníamos el código en el “snippet”, ahora crearemos un archivo de texto, que contendrá el código JS. El archivo contendrá una función que recibe un texto y lo retorna pasado a mayúsculas.

```
function pasarMayusculas(frase){
    return frase.toUpperCase();
}
```

El nombre del archivo puede ser: ejemplo.js.

¿Cómo vinculo ambos archivos? Hay distintas formas, pero la más recomendable es desde el HTML poner en el head el vínculo (lo mejor es hacer un subdirectorío js, e incluir ahí todos los archivos js que es la que utilizaremos).

```
<html lang="es">
<head>
<charset = "UTF-8">
<script src="js/ejemplo.js"></script>
<title> Ejemplo: leer String y pasarlo a Mayúsculas</title>
</head>
```

Lo probamos, pero vemos que no hace nada.

Debemos agregar ahora que, cuando se haga “click” en el botón, se haga el cálculo o proceso y se muestre en la pantalla. Todo el código JS debe ejecutarse una vez que esté toda cargada la página, no antes.

En JS el código entero del archivo ejemplo.js que explicaremos más adelante es:

```
window.addEventListener("load", inicio);
function inicio(){
    document.getElementById("boton").addEventListener("click", proceso);
}
function proceso(){
    let texto = document.getElementById("frase").value;
```

```

    let respuesta = pasarMayusculas(texto);
    mostrarEnPantalla(respuesta);
}
function pasarMayusculas(frase){
    return frase.toUpperCase();
}
function mostrarEnPantalla(textoMostrar){
    document.getElementById("resultado").innerHTML = textoMostrar;
}

```

Explicaremos cada elemento.

El objeto **window** representa la ventana del navegador y todos los objetos, funciones y variables automáticamente serán miembros del objeto window. Las variables globales son propiedades del objeto window. Las funciones globales son métodos del objeto window.

Le indicamos que cuando esté cargada la página, ejecute la función “inicio”. Debemos esperar que se cargue porque sino podríamos estar refiriendo a un elemento, por ej. un botón, que todavía puede no estar cargado.

```

window.addEventListener('load', inicio);

```

Otras formas de escribir prácticamente lo mismo es:

```

window.onload=inicio;

```

y:

```

onload= inicio;

```

Observar que no se ponen los (), estamos asignándole la función, no el resultado de la función.

En el caso onload= inicio; asume que refiere a window.

En la función inicio, se hace lo necesario, por ejemplo asociar los eventos que me interesan. En este caso, el click del botón del documento html. Cuando un documento HTML se carga en un navegador, se convierte en un objeto **document**. Una forma de referirse a un elemento es con el método “getElementById”.

```

function inicio(){
    document.getElementById("boton").addEventListener("click", proceso);
}

```

El proceso es, cuando se haga click en el botón, tomar el texto, pasarlo a mayúsculas y ponerlo en la otra parte.

Para tomar el texto de pantalla, al document le pedimos que ubique el elemento con el identificador “frase”, y a ese elemento le pedimos el valor. Value retorna el valor actual de un elemento de entrada como por ejemplo un input. Esto devuelve el texto que se ingresó. Luego lo pasamos a mayúsculas y le decimos que lo ubique como innerHTML. InnerHTML refiere a todo el contenido dentro de un elemento HTML.

```
function proceso(){
    let texto = document.getElementById("frase").value;
    let respuesta = pasarMayusculas(texto);
    mostrarEnPantalla(respuesta);
}
function mostrarEnPantalla(textoMostrar){
    document.getElementById("resultado").innerHTML = textoMostrar;
}
function pasarMayusculas(frase){
    return frase.toUpperCase();
}
```

Probemos... ¡ya funciona como queremos!

8.3.3 Ampliación: mostrar los datos en lista

Para ello agregamos en el html:

```
<ul id="lista"> Palabras ingresadas en formato lista
</ul>
```

Agregamos en el JS: incluimos que muestre también en la lista:

```
function mostrarEnPantalla(textoMostrar){
    document.getElementById("resultado").innerHTML = textoMostrar;
    agregarElementoEnLista(textoMostrar);
}
function agregarElementoEnLista(texto){
    // creo un elemento de lista
    let node = document.createElement("LI");
    // creo nodo de texto, dentro lleva el texto que quiero
    let textnode = document.createTextNode(texto);
    // engancho al nodo de lista el nodo de texto
    node.appendChild(textnode);
    // engancho el nodo a la lista
    document.getElementById("lista").appendChild(node);
}
```

8.3.4 Ampliación: mostrar los datos en una tabla

Agregamos en el html

```
<table id="tabla">
  <caption> Tabla de datos </caption>
  <th>Palabras ingresadas </th>
</table>
```

Agregamos en el JS:

```
function mostrarEnPantalla(textoMostrar){
  document.getElementById("resultado").innerHTML = textoMostrar;
  agregarElementoEnLista(textoMostrar);
  agregarFilaEnTabla(textoMostrar);
}

function agregarFilaEnTabla(texto){
  let tablaPantalla = document.getElementById("tabla");
  let fila = tablaPantalla.insertRow();
  let celda = fila.insertCell();
  celda.innerHTML= texto;
}
```

8.4 Ejemplos básicos de vinculación HTML-JS

8.4.1 Votar. Ingreso de dato y presentación en párrafo.

Este ejemplo incluye el ingreso de un dato y presentación de la respuesta en un párrafo. Ingresar una edad e indicar si puede votar o no (si es mayor a 18). Lucirá similar a:

Ingrese la edad

Puede votar

El html puede ser:

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <script src="js/EjemploEdad.js"></script>
  <title>Ejemplo edad </title>
</head>
<body>
  <label for="edad">Ingrese la edad</label>
  <input type="number" id="edad" >
  <!-- Para poder solicitar el resultado, habrá que presionar el botón -->
```

```

        <input type="button" id = "boton" value="Consultar">
        <!-- El resultado se desplegará en un párrafo -->
        <p id="resultado"> </p>
    </body>
</html>

```

y el código JS:

```

window.addEventListener('load', inicio);

function inicio(){
    document.getElementById("boton").addEventListener("click", calculo);
}

function puedeVotar(edad){
    let respuesta = "No puede votar";
    if (edad>=18){
        respuesta="Puede votar";
    }
    return respuesta;
}

function calculo(){
    let numero = parseInt(document.getElementById("edad").value);
    document.getElementById("resultado").innerHTML = puedeVotar(numero);
}

```

8.4.2 Múltiplos

Ingresar dos números, generar un array con los números múltiplos de 5 entre esos dos números y mostrar el array en una tabla. Desde JS se generará el array y la tabla. Lucirá similar a:

Ingrese número inicial Ingrese número final

Tabla de resultados

Valor 5

Valor 10

Valor 15

El código HTML puede ser:

```

<!doctype html>
<html lang="es">
<head>
<meta charset="utf-8">
<script src="js/EjemploMultiplos5.js"></script>

<title>Ejemplo Múltiplos de 5 </title>
</head>

```

```

<body>

  <label for="desde">Ingrese número inicial</label>
  <input type="number" id="desde" >
  <label for="hasta">Ingrese número final</label>
  <input type="number" id="hasta" >

  <!-- Para poder solicitar el resultado, habrá que presionar el botón -->
  <input type="button" id = "boton" value="Consultar">
  <p> Tabla de resultados </p>
  <table id="tabla">
    <th>Números</th>
  </table>
</body>
</html>

```

En el archivo de JS:

```

window.addEventListener('load', inicio);

function inicio(){
  document.getElementById("boton").addEventListener("click", calculo);
}

function calculo(){
  let desde = parseInt(document.getElementById("desde").value);
  let hasta = parseInt(document.getElementById("hasta").value);
  let resultado = proceso(desde,hasta);
  mostrarEnTabla(resultado);
}

function proceso(desde, hasta){
  let datos=[];
  let menor= Math.min(desde,hasta);
  let mayor = Math.max(desde,hasta);
  desde = menor;
  hasta = mayor;
  let resto = desde%5;
  if(resto>0){
    desde = desde - resto +5;
  }
  for (let i = desde; i <= hasta; i= i+5){
    datos.push(i);
  }
  return datos;
}

function mostrarEnTabla(lista){

```

```

let tablita = document.getElementById("tabla");
tablita.innerHTML = ""; // limpio todo
for(let j=0; j<lista.length; j++){
    // creo elemento de tipo TR
    let filita = document.createElement("TR");
    // creo nodo de texto con el texto que quiero
    let nodoTextoFila = document.createTextNode("Valor " + lista[j]);
    // engancho al elemento el nodo de texto
    filita.appendChild(nodoTextoFila);
    // engancho el nodo a la tabla
    tablita.appendChild(filita);
}
}

```

8.4.3 Extensión: múltiplos (uso de radio button)

Según la indicación del usuario, se mostrará en una tabla o en una lista. Entonces agregaremos en la interfaz un radio button y una lista. Lucirá similar a:

Ingrese número inicial Ingrese número final

☒ Resultados en lista
☐ Resultados en tabla

Resultados

- 5
- 10
- 15

Agregaremos en la interfaz un radio button y una lista.

```

<head>
  <meta charset="utf-8">
  <script src="js/EjemploMultiplos5.js"></script>
  <title>Ejemplo Múltiplos de 5 </title>
</head>
<body>

  <label for="desde">Ingrese número inicial</label>
  <input type="number" id="desde" >
  <label for="hasta">Ingrese número final</label>
  <input type="number" id="hasta" >

  <br />
  <input type="radio" id = "formatoLista" name="respuesta"
  value="Lista" checked>Resultados en lista<br>
  <input type="radio" id = "formatoTabla" name="respuesta"
  value="Tabla">Resultados en tabla<br>

```



```

<br />

<!-- Para poder solicitar el resultado, habrá que presionar el boton -->
<input type="button" id = "boton" value="Consultar">

<p> Resultados </p>
<table id="tabla">

</table>
<ul id = "listaDesordenada"></ul>

</body>
</html>

```

Código en JS: cuando se solicita el cálculo, debemos agregar el chequeo de si seleccionó en forma de tabla o de lista. Antes de cargarlas, oculto ambas con la función ocultar que crearemos y luego en cada caso se mostrará lo necesario.

```

window.addEventListener('load', inicio);

function inicio(){
    document.getElementById("boton").addEventListener("click", calculo);
}

function calculo(){
    let desde = parseInt(document.getElementById("desde").value);
    let hasta = parseInt(document.getElementById("hasta").value);
    let resultado = proceso(desde,hasta);
    ocultar();
    let ejemplo = document.getElementById("formatoLista");
    if (ejemplo.checked){
        mostrarEnLista(resultado);
    }
    else {
        mostrarEnTabla(resultado);
    }
}

function ocultar(){
    let tablita = document.getElementById("tabla");
    tablita.style.display = 'none'; // oculto la tabla
    let listita = document.getElementById("listaDesordenada");
    listita.style.display = 'none'; // oculto la lista
}

function proceso(desde, hasta){
    let datos=[];

```

```

    let menor= Math.min(desde,hasta);
    let mayor = Math.max(desde,hasta);
    desde = menor;
    hasta = mayor;
    let resto = desde%5;
    if(resto>0){
        desde = desde - resto +5;
    }
    for (let i = desde; i <= hasta; i= i+5){
        datos.push(i);
    }
    return datos;
}

function mostrarEnTabla(lista){
    let tablita = document.getElementById("tabla");
    tablita.style.display = 'block';
    tablita.innerHTML = ""; // limpio todo
    for(let j=0; j<lista.length; j++){
        // creo elemento de tipo TR
        let filita = document.createElement("TR");
        // creo nodo de texto con el texto que quiero
        let nodoTextoFila = document.createTextNode("Valor " + lista[j]);
        // engancho al elemento el nodo de texto
        filita.appendChild(nodoTextoFila);
        // engancho el nodo a la tabla
        tablita.appendChild(filita);
    }
}

function mostrarEnLista(lista){
    let listita = document.getElementById("listaDesordenada");
    listita.style.display = 'block';
    listita.innerHTML=""; // limpio los elementos
    for (let i = 0; i < lista.length; i++){
        let nodo = document.createElement("li");
        let nodoTexto = document.createTextNode(lista[i]);
        nodo.appendChild(nodoTexto);
        listita.appendChild(nodo);
    }
}

```

9 Depuración

9.1 Introducción a Prueba de Programas

Los programas se prueban para ver si están mal. Si se hace una prueba y no encontré ningún error, probablemente esté mal probado. Los principios que guían la prueba de programas son:

- Definir los resultados esperados: tener definidos claramente qué es lo que se quiere probar, qué se ingresará y qué se espera obtener.
- Otra persona que pruebe: los programas no deberían ser probados sólo por quien los hace. Es altamente recomendable que otra persona pruebe en forma independiente el programa.
- Ver a fondo cada salida o resultado. Debe verificarse cuidadosamente cada una de las salidas o resultados obtenidos.
- Probar casos esperados y correctos así como inesperados o incorrectos. Por ejemplo, si se está probando el ingreso de la edad de un funcionario y el rango es entre 18 y 65 años, hay que probar ingresar valores correctos, como 20 o 45; valores de borde: 18, 65; y valores no válidos, como por ejemplo 17, -3, 66, 100.
- Ver que el programa no haga cosas "de más". Por ejemplo, que el sistema luego de calcular los sueldos de los funcionarios borre todos los datos...
- Las pruebas deben ser completas. Siempre debe probarse en su totalidad.
- Pensar en probabilidad de más errores. Hay que pensar que si "pude cometer x errores, ¡quizás haya x+1!".

9.2 Estrategias de Prueba de Programas

Planteemos la siguiente analogía: se tiene un reloj de agujas y se desea saber si funciona bien. Una alternativa es mirar la hora que marca, esperar una hora y ver nuevamente la hora. O sea, considerar el reloj como una "caja negra" y analizar las salidas. Otra opción es abrir el reloj y estudiar si todos los engranajes funcionan bien. Es una prueba de "caja blanca". Estos dos enfoques, de caja negra y de caja blanca, se aplican para la prueba de programas.

9.2.1 Caja Negra

Para hacer pruebas de caja negra, se elaboran datos de prueba, indicando que es lo que se quiere probar, qué se espera obtener y qué se obtiene realmente. Siempre se debe indicar el valor con el que efectivamente se realizó cada prueba.

Así, en el caso de la edad del funcionario se podría armar una tabla de datos de prueba como la que sigue:

Qué se quiere probar	Qué se espera obtener	Qué se obtiene
se quiere ingresar un funcionario de edad 24, para lo cual se digita el valor 24 al solicitar la edad	que informe que está correcto	se acepta el resultado

se quiere incorrectamente ingresar un funcionario de edad negativa, por ejemplo: -3	que informe que no es correcto	se obtiene lo esperado
---	--------------------------------	------------------------

Nota: este es un ejemplo parcial.

9.2.2 Caja Blanca

La prueba de caja blanca implica una revisión "dentro" del programa. Se define cobertura como el grado de revisión del programa. Se puede hacer cobertura de sentencias, de decisiones o caminos. En el caso de sentencias, se trata de asegurar que toda sentencia es ejecutada al menos una vez. En el de decisiones se debe verificar que toda decisión alguna vez sea verdadera y alguna vez sea falsa. En la de caminos, se trata de chequear que todos los caminos posibles en el programa se dan alguna vez.

9.3 Desarrollo y Prueba de métodos

Cuando se diseña un método, función o procedimiento, se puede realizar de dos formas: tratar de desarrollar desde el comienzo la versión lo más completa posible o realizar aproximaciones sucesivas al método final, desde una versión inicial simplificada. Así, se podría realizar una primera versión que devuelva, por ejemplo, un valor constante, un cálculo mínimo o que le solicite al operador el valor. Cuando se prueban los métodos debe tenerse en cuenta si ya es la versión final o una intermedia.

En el caso de tener versiones intermedias preliminares se tiene la ventaja de que se puede ir avanzando en diferentes tareas y luego se completa; la desventaja es que interpretar los resultados puede ser más complicado debido a esos valores ficticios. En el caso que se opte por poner la versión definitiva, tiene la ventaja de que es más fácil revisar las salidas pero puede demorarse más en tener la versión completa del sistema.

9.4 Manejo de errores

La depuración de un programa es algo no sistemático y no garantizado. Los pasos sugeridos para encontrar/corregir errores son:

- Determinar el error;
- Ubicarlo en el programa;
- Corregirlo; y
- Verificarlo.

Se recomienda:

- Llevar un registro: ir anotando qué cosas se han probado, qué falta, qué se descarta, etc.
- Trabajar con listado del código fuente actualizado: en la medida que se realizan modificaciones al programa y se van realizando anotaciones en los listados impresos, se complica entender o descubrir errores, pues la versión en papel difiere con la real en la máquina.
- Guardar la versión anterior, antes de hacer "cambios geniales". Recordar guardar una versión del código completo hasta ese momento.

- Pensar en la posibilidad de reescribir: a veces es más fácil y seguro reescribir el método problemático que seguir corrigiendo errores sobre el mismo.
- Aprender de los resultados negativos: cuando se supuso que un error estaba en determinado lugar y se comprueba que no es así, es un resultado negativo pero a su vez permite descartar ese lugar como posible fuente del error.

9.4.1 ¿Cómo se buscan?

- Pensar: analizar cuidadosamente qué es lo que pasa, en qué casos ocurre, descubrir la situación que lo genera.
- Descansar: luego de dedicar un rato a tratar de entender que es lo que pasa, conviene "alejarse" del código para luego, al retornar descansado, seguramente se encontrará el error;
- Pedir auxilio: recurrir a la ayuda de otro programador.
- Usar herramientas (*debugger*): hay software que permite realizar el seguimiento paso a paso del programa, mostrando las variables y la secuencia de instrucciones que se van ejecutando. Esta herramienta es útil si se ha pasado por las etapas de pensar, descansar y pedir auxilio. De otra manera, toda la información que genera la herramienta no nos será aprovechable, pues no estamos lo suficientemente familiarizados con el programa.
- Experimentar: si nada de lo anterior funciona... probar cambiar algo "por las dudas"...

Al corregir errores, pensar en la posibilidad de múltiples errores. Arreglar los errores, no los síntomas. Tener previsto que la corrección también puede ser errónea. Rediseñar si es necesario.

9.5 Depuración en HTML

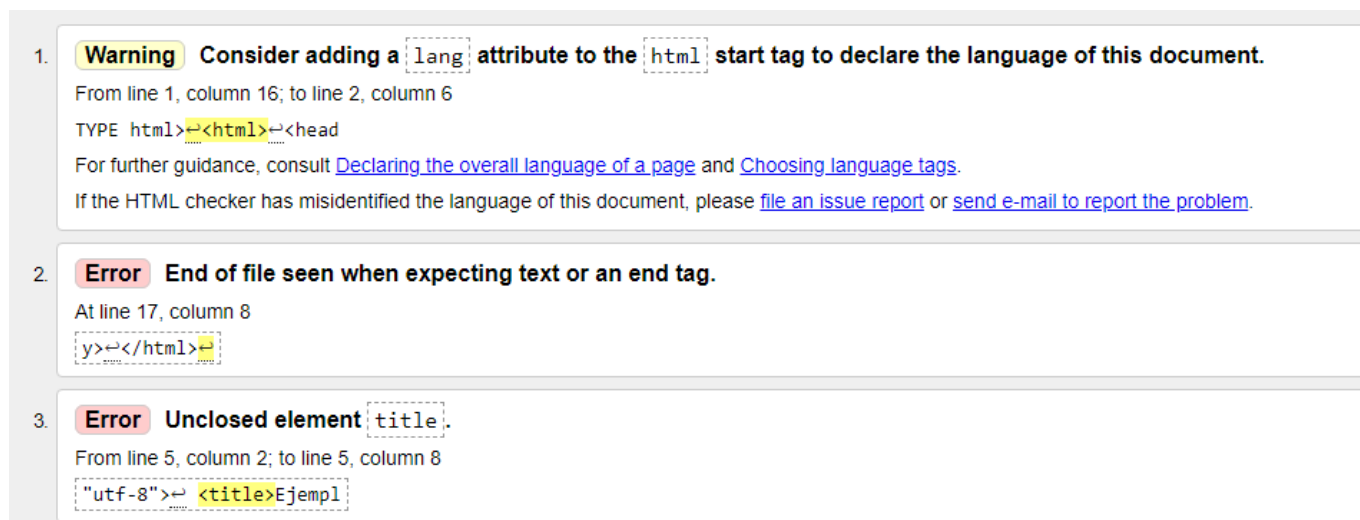
A veces ocurre que una página que desarrollamos no luce como queremos o no se ve bien en distintos navegadores.

Por ejemplo supongamos que tenemos este código HTML:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Ejemplo<title>
</head>
<body>
  <h1>Ejemplo de depuración</h1>
  <p>Este texto es de ejemplo
  <p> Con varias líneas para probar </p>
  <p>Link <a href="https://www.mozilla.org">link a Mozilla</a>
</ul>
</body>
</html>
```

Si la probamos, vemos que no se ve nada en el navegador y el título está erróneo. Podemos revisar nuestro código y ahí detectaremos por ejemplo que el tag de title no fue cerrado, entre otros errores.

Para verificar el código se puede utilizar el validador de W3C. Está disponible en: <https://validator.w3.org/>. Allí, si subimos esta página muestra mensajes de error tales como:



The screenshot displays three error messages from the W3C Validator. The first is a warning about missing the 'lang' attribute on the 'html' tag. The second is an error about an unexpected end tag 'y' at line 17. The third is an error about an unclosed 'title' element at line 5. Each message includes a line and column reference and a visual representation of the problematic code snippet.

- Warning** Consider adding a `lang` attribute to the `html` start tag to declare the language of this document.
From line 1, column 16; to line 2, column 6
TYPE `html>` `<html>` `<head`
For further guidance, consult [Declaring the overall language of a page](#) and [Choosing language tags](#).
If the HTML checker has misidentified the language of this document, please [file an issue report](#) or [send e-mail to report the problem](#).
- Error** End of file seen when expecting text or an end tag.
At line 17, column 8
`y>` `</html>`
- Error** Unclosed element `title`.
From line 5, column 2; to line 5, column 8
`"utf-8">` `<title>`Ejempl

Informa de “warning” o “aviso”, que es una sugerencia de mejora y “error”, que señala un problema. Podemos corregir los errores y avisos, y volver a subir la página hasta que se solucionaron todos.

Los navegadores tratan de interpretar los errores de escritura encontrados, por lo que seguirán funcionando aunque no produzcan el resultado esperado. O sea, una página podría visualizarse correctamente pero contener errores.

También, podría pasar que la referencia al archivo de js esté incorrecta. Cuando se organiza un sitio debe estar estructurado en carpetas. Debería contener la página html, una carpeta con JS, otra con CSS (que veremos más adelante) y otras más, por ejemplo con videos u otros recursos que se utilicen. A través de caminos relativos se puede acceder a los elementos. Por ejemplo, para acceder al archivo “ejemplo.js” dentro de la carpeta js, la dirección relativa es: “js/ejemplo.js”.

9.6 Errores comunes y consideraciones de JS

Algunos puntos para recordar:

- Es un error común usar el operador de asignación en vez del de comparación. Estar atento a utilizar `==` o `=` según corresponda.
- La sentencia switch utiliza comparación estricta.

Por ejemplo:

```
let x = 10;
switch(x) {
  case "10": alert("Hello");
  //Aquí van más opciones....
}
```

no sale el alert, pues compara estrictamente y el valor 10 no es idéntico a "10". En cambio si escribimos:

```
let x = 10;
switch(x) {
  case 10: alert("Hello");
}
```

sí sale el alert.

- Concatenar es para strings, sumar es para números. Ejemplo:

```
let x = 10 + 5;      // resultado es 15
let x = 10 + "5";    // resultado es 105
```

- Ubicación del ";". Ejemplo:

```
if (x == 19);
{
  // code block
}
```

El bloque se ejecutará independiente del valor de x, porque está ";"

- No cortar la línea del return. Ejemplo:

```
function miFuncion(a) {
  let
  power = 10;
  return
  a * power;
}
```

devuelve undefined, porque como la línea del return está completa, sale de la función en ese momento. En el caso del let, no está completa y sigue tomando los datos de la siguiente línea.

Para ayudar en la depuración de un programa, se puede poner: `console.log(...)` e incluir las variables o información que necesitemos. También se puede poner `alert(...)`. Otra opción es poner dentro del código la sentencia "debugger" como se muestra en el ejemplo:

```
let x = 15 * 5;  
debugger;
```

Esto hace que se abra la consola y podamos inspeccionar variables por ejemplo.

10 Más de HTML

10.1 HTML - Profundización

10.1.1 Caracteres especiales

En HTML, los caracteres <, >, " , ' y & son caracteres especiales. Son parte de la sintaxis HTML en sí. Para incluir esos caracteres en el texto se tienen que utilizar caracteres de referencia — códigos especiales que representan caracteres. Cada carácter de referencia empieza con un & y finaliza con un punto y coma (;).

Carácter	Equivalente
<	<
>	>
"	"
'	'
&	&

Otros caracteres se presentan en la tabla siguiente:

Entidad	Carácter	Descripción oficial
ñ	ñ	latin letter n with tilde
Ñ	Ñ	latin capital n letter with tilde
á	á	a acute
é	é	e acute
í	í	i acute
ó	ó	o acute
ú	ú	u acute
Á	Á	A acute
É	É	E acute
Í	Í	I acute
Ó	Ó	O acute
Ú	Ú	U acute
€	€	euro

Ejemplo:

```
<h1>Aprendo HTML: mi primera p&aacute;gina en HTML</h1>
```

10.2 Otras etiquetas en META

Hay otras etiquetas dentro de META: autor y descripción. Si abrimos por ejemplo la página de El País (elpais.com.uy) y observamos su código fuente, podemos ubicar la etiqueta **META description**:

```
<meta name="description" content="Noticias de Uruguay y el Mundo: Últimas noticias en deportes, economía, política y tecnología. Manténgase informado sobre las noticias de Uruguay en el diario EL PAIS Uruguay">
<meta name="keywords" content="noticias uruguay, ultimas noticias, prensa, el pais, noticias nacionales, noticias de hoy, actualidad">
```

Ahora, si buscamos en Google “diario el país Uruguay”, aparecerá esta información (observar la descripción):

Noticias Uruguay y el Mundo actualizadas - Diario EL PAIS Uruguay
<https://www.elpais.com.uy/> ▼
Noticias de **Uruguay** y el Mundo: Últimas noticias en deportes, economía, política y tecnología.
Manténgase informado sobre las noticias de **Uruguay** en el **diario EL PAIS Uruguay**.

Además, las palabras clave aparecen en negrita.

También está VIEWPORT, que es el área visible de una página web. Esa área varía con el dispositivo, por ejemplo, en un celular es chica y en un monitor de PC es grande. Antes, era sólo para PC, y era común que las páginas tuvieran tamaño fijo. Para ajustar a los distintos dispositivos, se escala. Tiene opciones: width=device-width setea el ancho de la página a seguir, initial-scale=1.0: setea el nivel inicial de zoom cuando se carga por primera vez la página.

Si analizamos el código fuente de www.ort.edu.uy podremos ver la opción de VIEWPORT (imagen parcial):

```
</><meta name="generator" content="InnovaPortal - Professional Content Management and Portal">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=2" />
```

10.3 Comentarios y listas

10.3.1 Comentarios

En el código HTML se pueden poner comentarios:

```
<!-- texto a comentar -->
```

10.3.2 Listas

HTML provee varios formatos de listas: desordenadas (como una lista de supermercado), lista ordenada (como una receta) y listas anidadas (lista dentro de listas).

Ejemplo parcial:

```
<ul>
  <li>Leche</li>
  <li>Harina</li>
  <li>Manzanas</li>
</ul>
<ol>
  <li>Primer paso</li>
  <li>Segundo paso</li>
  <li>Tercer paso</li>
</ol>

<ol>
  <li> .....</li>
  <li> .....</li>
  <li> .....</li>
  <ul>
    <li> .....</li>
    <li> .....</li>
  </ul>
</ol>
```

Se puede configurar por ejemplo el estilo del bullet (“style=“list-style-type:circle””), si el tipo es numérico (type=“1”), en qué número se comienza (ej. start=“50”).

10.4 Hipervínculos y énfasis

10.4.1 Hipervínculos

Como se indicó, es una conexión desde un recurso a otro. Un link se crea envolviendo el texto (u otro contenido) que se quiere convertir en un link dentro de un elemento <a>, y dándole el atributo href, que contiene hacia dónde quiero apuntar.

Ejemplo:

```
<p>Ejemplo de link a <a href="https://www.ort.edu.uy">ORT</a>. </p>
```

Se le puede agregar más información con el title:

```
<p>Ejemplo de link a <a href="https://www.ort.edu.uy" title="sitio de la Universidad">ORT</a>. </p>
```

Se ve de la siguiente forma:

Ejemplo de link a [ORT](#).

sitio de la Universidad

Se le puede indicar que abra en otra página incluyendo: target="_blank".

Si el link lleva a una página en el mismo directorio:

```
<a href="contactos.html">página de contacto</a>.
```

Si es en otra carpeta:

```
<p>Visitar <a href="proyecto/index.html">página principal</a>.</p>
```

Si se quiere acceder a la carpeta del padre:

```
<a href="../pdfs/proyecto-resumen.pdf">resumen</a>
```

Si quiero acceder a otra parte del documento, primero debo darle nombre con id.

```
<h2 id="direccion">dirección</h2>
```

Si quiero acceder desde otra parte del mismo documento:

```
<p>La <a href="#direccion">dirección de correo de la empresa</a> puede ser encontrada en la parte inferior.</p>
```

El link puede ser un e-mail:

```
<a href="mailto:ana@gmail.com">Enviar mail a Ana</a>
```

10.4.2 Énfasis

Se pueden destacar o hacer énfasis en ciertos elementos. NOTA: es altamente recomendable utilizar estilos en CSS en vez de incluirlos en HTML. Se presentan aquí como información.

Utilización de em:

```
<p>Hoy es <em>viernes</em> y viene el <em>fin de semana</em>.</p>
```

Utilización de i:

```
<p>Hoy es <i>viernes</i> y viene el <i>fin de semana</i>.</p>
```

Se puede aplicar negrita con strong:

```
<p>La nota máxima es <strong>100</strong>.</p>
```

También se puede usar b (bold):

```
<p>La nota máxima es <b>100</b>.</p>
```

En ambos casos se ve: La nota máxima es **100**.

Todo el manejo de estilos se verá más adelante en forma detallada con CSS.

10.5 Abreviaturas, subíndices y superíndices

Para abreviaturas se usa abbr.

Ejemplo:

```
<p>Se usa <abbr title="Hypertext Markup Language">HTML</abbr> para estructurar los documentos</p>
```

Se ve:

Se usa **HTML** para estructurar los documentos

Hypertext Markup Language

Los subíndices y superíndices se logran respectivamente con sub y sup.

Ejemplo:

```
<p>Mi cumpleaños es el 1<sup>ero</sup> de enero de 2001.</p>
```

```
<p>La fórmula de la cafeína es
```

```
  C<sub>8</sub>H<sub>10</sub>N<sub>4</sub>O<sub>2</sub>.</p>
```

```
<p>Si x<sup>2</sup> es 9, x debe ser 3 o -3.</p>
```

Se ve:

Mi cumpleaños es el 1^{ero} de enero

La fórmula de la cafeína es C₈H₁₀N₄O₂.

Si x² es 9, x debe ser 3 o -3.

10.6 Imágenes

Para poner una imagen se usa el tag "img", que no tiene tag de cierre. Como mínimo requiere el atributo src que refiere a la fuente. La fuente es una ruta que apunta a una imagen (puede estar en la misma página, en una url relativa o absoluta, igual que cuando se usa el href).

Ejemplo:

```

```

Si está en una subcarpeta se indica:

```

```

El atributo **alt** sirve para dar una descripción textual de la imagen:

```

```

Se puede indicar ancho y alto (aunque conviene hacerlo por CSS) y un título (que aparece como "tooltip" o descripción emergente):

```

```

Para vincular semánticamente la imagen con su caption (leyenda), en HTML5 tenemos el tag "figure" y "figcaption".

```
<figure>
  
  <figcaption>Empresa SRL</figcaption>
</figure>
```

10.7 Tablas

Con colspan puedo agrupar columnas. Por ejemplo, para obtener:

Nombre	Telefonos
Juan Perez	4221313 096123123

el código puede ser:

```
<table>
  <tr>
    <th>Nombre</th>
    <th colspan="2">Telefonos</th>
  </tr>
  <tr>
```

```

<td>Juan Perez</td>
<td>4221313</td>
<td>096123123</td>
</tr>
</table>

```

Análogamente se puede realizar para filas, utilizando rowspan

10.8 Formularios (form, label, input, textarea)

Los formularios son uno de los puntos importantes en la interacción entre el usuario y la aplicación. Permiten enviar datos al servidor y también pueden ser usados en la propia página.

Primero conviene diseñar a mano el formulario, por ejemplo:

Veremos cómo implementar el form diseñado. Este ejemplo incluye estas etiquetas: `<form>`, `<label>`, `<input>`, `<textarea>`, y `<button>`. El form se define:

```

<form action="/handling-form-page" method="post">

</form>

```

El atributo **action** define la URL donde la información ingresada será enviada. Puede ser una url absoluta, relativa o si no dice nada, es a la propia página (en versiones viejas, para indicar que se envía a la misma página se ponía `<form action="#">`).

El atributo **method** indica la forma en que será enviada ("get" o "post"). Si envío con get se ve en la url, por ejemplo: `www.foo.com/?usu=Ana&tel=98765432`. Cuando se envía con post, la información no se agrega a la URL (va aparte el contenido).

```

<form action="/handling-form-page" method="post">
  <div>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
  </div>
  <div>
    <label for="mail">E-mail:</label>

```

```
<input type="email" id="mail" name="mail">
</div>
<div>
  <label for="msg">Mensaje:</label>
  <textarea id="msg" name="msg"></textarea>
</div>
<div>
  <button type="submit">Enviar mensaje</button>
</div>
</form>
```

Los “div” se usan para estructurar.

En el label se pone un “for” que asocia esa etiqueta con el área de entrada correspondiente. Es importante por la accesibilidad. Además, reaccionan también al click.

Cada elemento tiene su “id”, que lo identifica.

El type del input puede ser text, email, password, number, range, date entre otros. No hay tag de cierre.

Se pueden indicar parámetros, por ejemplo en number se puede indicar mínimo y máximo

El botón si es de tipo submit, realiza el envío del formulario. Para este curso, utilizaremos botones de tipo button, no submit.

```
<input type="email" id="email" name="email">
<input type="password" id="pwd" name="pwd">
<input type="number" name="edad" id="edad" min="1" max="10" step="2">
<input type="range" name="unidades" id="unidades" min="0" max="500" step="10">
<input type="datetime-local" name="fechayhora" id="fechayhora">
<input type="color" name="color" id="color">
```

Notar que el control **textarea** lleva tag de cierre.

El atributo “**name**” es enviado con el formulario.

El tag **button** permite el tipo “submit”, que envía los datos a la página definida en el action del form. Podría también ser de tipo “reset”, que resetea todos los campos del formulario. Otro tipo puede ser el tipo “button”, que no hace nada por defecto, pero puede ser utilizado desde JS.

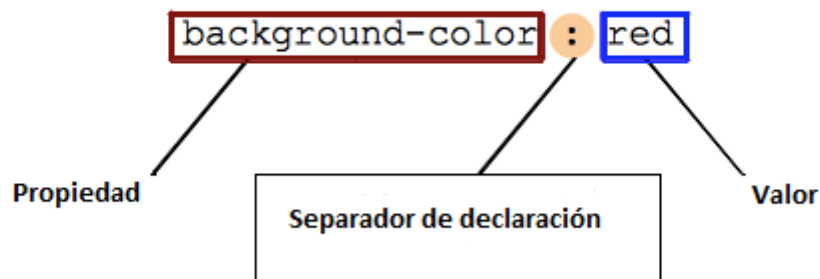
No se puede anidar un form dentro de otro.

11 Más de CSS

11.1 Profundización en Reglas y selectores

11.1.1 Reglas

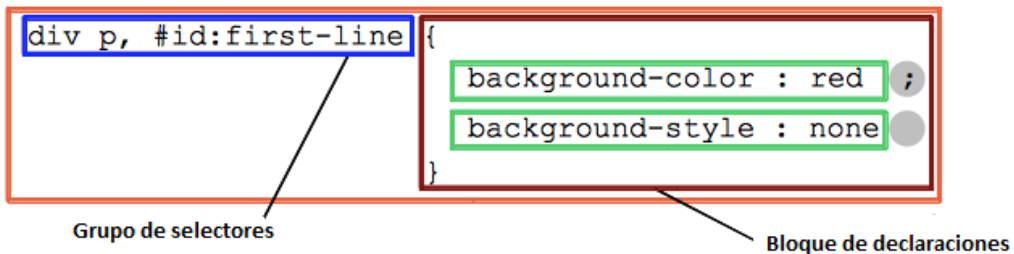
Una declaración tiene la forma: **propiedad: valor**



Se pueden poner varias:

Una regla está formado por: selectores bloque de declaración.

Reglas CSS



Un selector puede tener muchas reglas y se aplican en cascada.

El orden es:

- 1) importance (tiene la palabra "importance")
- 2) especificidad
- 3) orden en el código

Los selectores pueden ser:

- 1) simple (ej. class, id)
- 2) atributos
- 3) seudoclases: elementos en cierto estado por ejemplo elemento apuntado por el mouse
- 4) seudoelementos: por ej. 1era palabra de cada párrafo
- 5) combinaciones
- 6) múltiples

A continuación se presentarán ejemplos de algunos de ellos.

11.1.2 Selectores simples

```
/* Todos los elementos p son rojos*/  
  
p {  
  color: red;  
}  
  
/* Todos los elementos div son azules */  
  
div {  
  color: blue;  
}
```

11.1.3 Selectores simples: class

Dada la siguiente página (parcial):

```
<ul>  
  <li class="primera cuarta">Este es un texto de prueba</li>  
  <li class="segunda cuarta">Uso de hojas estilos</li>  
  <li class="tercera">en documento HTML</li>  
</ul>
```

Se puede observar que el primer elemento tiene las clases “primera” y “cuarta”. Definimos una regla para que sea aplicada a todos los elementos de cada una. Por ejemplo, para la clase primera:

```
/* Los elementos con la clase "primera" tendrán su texto en negrita */  
  
.primera {  
  font-weight: bold;  
}
```

El selector de clase se indica con el punto.

Notar que la clase funciona como un agrupador de elementos. Cuando utilizamos un selector de tipo clase, se le aplicarán las reglas definidas en la clase a todos los elementos que pertenecen a la misma.

11.1.4 Selectores simples: id

Cada elemento de una página html puede tener un ID (el cual debería ser único), fijado con el atributo id. Para referirme a él uso “#”.

Es la forma más eficiente para acceder a un elemento particular.

Ejemplo (parcial):

```
<p id="central"> — "Buenos días."</p>
```

Estilos:

```
#central {  
  font-family: cursive;  
}
```

Notar que, si no colocamos un id único al elemento que queremos hacer referencia, no podremos asegurarnos poder acceder al mismo mediante este selector.

Por otro lado, el selector * hace referencia a toda la página.

11.1.5 Seudoclases

Las pseudoclases son palabras clave precedidas por :. Por ejemplo,

```
a:visited {  
  color: blue;  
}  
  
/* Se destacará el link cuando se pase el mouse por  
encima, se haga foco o sea activado */  
  
a:hover,  
a:active,  
a:focus {  
  color: darkred;  
  text-decoration: none;  
}
```

Cuando se visita el link, el color es azul. Cuando tiene el foco no tiene text decoration.

11.1.6 Seudoelementos

Son parecidos a las pseudoclases. Van con "::".

Opciones:

- ::after
- ::before
- ::first-letter
- ::first-line
- ::selection
- ::backdrop

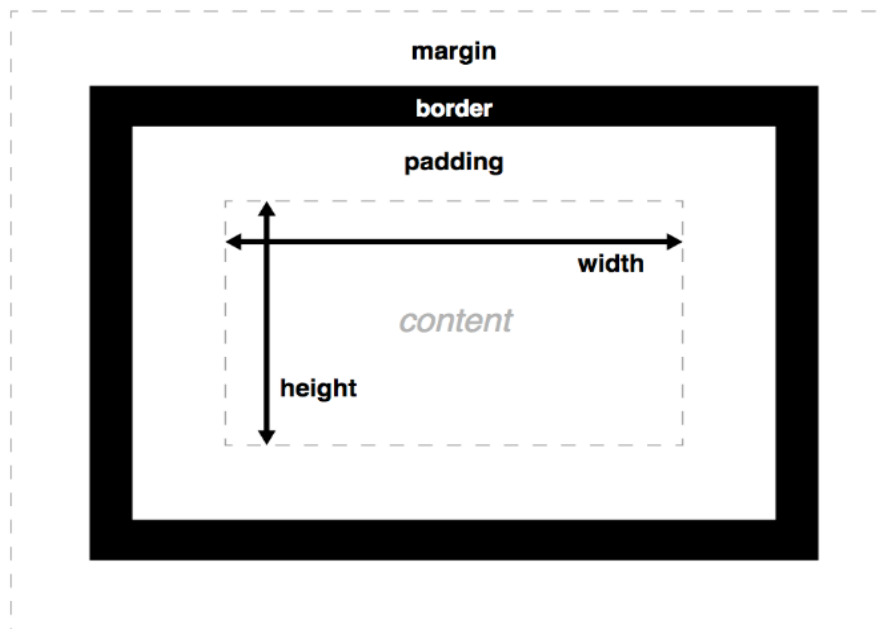
Ejemplo:

```
p::first-line {  
  font-weight: bold;  
}  
  
p::first-letter {  
  font-size: 3em;  
  border: 1px solid black;  
  background: red;  
  display: block;  
  float: left;  
  padding: 2px;  
  margin-right: 4px;  
}
```

La primera línea del párrafo tendrá texto en negrita, mientras que a la primera letra del párrafo se le aplicarán otras reglas.

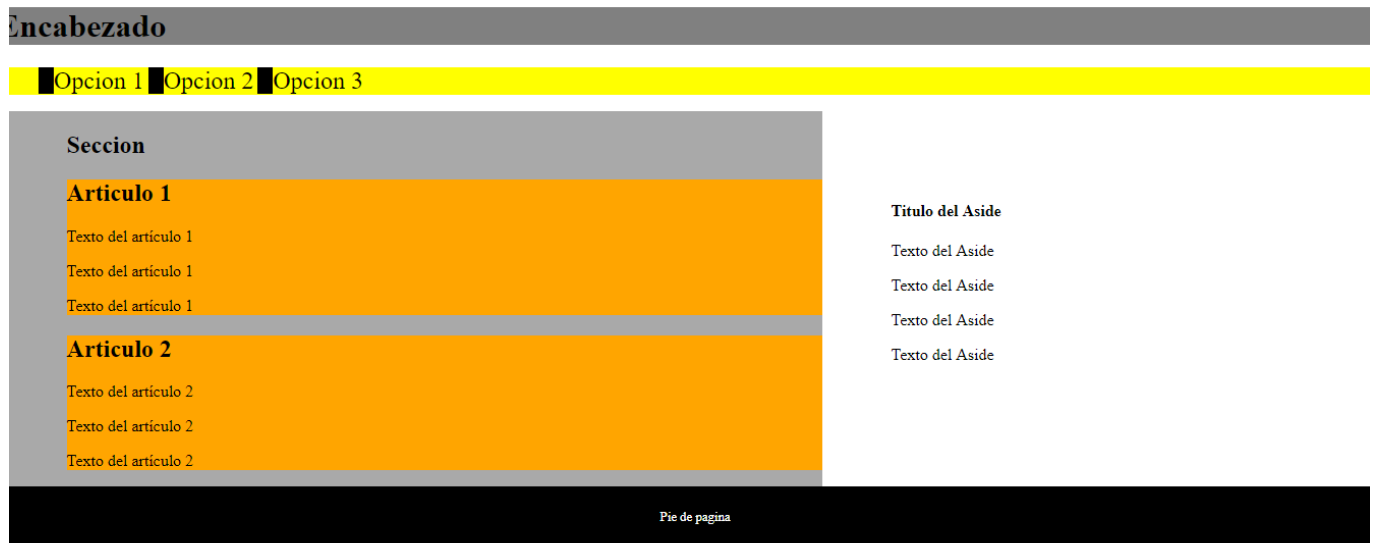
11.2 Modelo de Caja

El modelo de caja de CSS es la base del formato en la Web. Cada elemento se representa por una caja rectangular con el contenido, padding (“relleno”), border (por defecto, ancho 0) y margen.



11.3 Ejemplo de Tags semánticos y estilos

Veremos un ejemplo aplicando tags semánticos y estilos. La página lucirá:



El HTML es:

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title> Ejemplo Tags Semánticos </title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/ejemplo.css">
</head>
<body>
<header>
  <h1>Encabezado</h1>
</header>
<nav>
  <ul>
    <li>Opción 1</li>
    <li>Opción 2</li>
    <li>Opción 3</li>
  </ul>
</nav>
<section>
  <h2>Sección </h2>
  <article>
    <h2>Artículo 1</h2>
    <p>Texto del artículo 1</p>
    <p>Texto del artículo 1</p>
    <p>Texto del artículo 1</p>
  </article>
```

```

        <article>
            <h2>Artículo 2</h2>
            <p>Texto del artículo 2</p>
            <p>Texto del artículo 2</p>
            <p>Texto del artículo 2</p>

        </article>
</section>
<aside>
    <h4>Título del Aside</h4>
    <p>Texto del Aside</p>
    <p>Texto del Aside</p>
    <p>Texto del Aside</p>
    <p>Texto del Aside</p>
</aside>
<footer>
    <p>Pie de página</p>
</footer>
</body>
</html>

```

Un posible CSS para este ejemplo es (ejemplo.css):

```

header{
    background-color:gray;
    width:100%;
}

nav {
    background-color: yellow;
}

nav ul li{
    display:inline; /* así los pone en la misma línea */
    font-size: 1.5em; /* así se ven mas grandes las opciones */
    border-left-width: 15px;
    border-left-color: black;
    border-left-style: solid;
}

section{
    width:55%;
    background-color:darkgray;
    float:left;
    padding-left:5%;
}

article {
    background-color: orange;
}

```

```
}  
  
aside{  
    width:25%;  
    float:left;  
    padding:5%;  
}  
footer{  
    width:100%;  
    background-color:black;  
    text-align:center;  
    float:left;  
    color:white;  
    font-size:80%;  
    padding: 10px;  
}
```

12 Objetos en JS

12.1 Introducción

Probemos este ejemplo en la consola:

```
> let deportistaPreferido = ["Federer", "tenis"];  
< undefined  
> deportistaPreferido[0];  
< "Federer"  
> deportistaPreferido[1];  
< "tenis"
```

Se está guardando un array con los datos del deportista preferido. Miremos ahora este código:

```
> let deportistaPreferido = {  
  apellido: 'Federer',  
  deporte: 'tenis'  
};  
< undefined  
> deportistaPreferido.apellido;  
< "Federer"  
> deportistaPreferido.deporte;  
< "tenis"
```

Si observamos detenidamente, es parecido al anterior. En los dos ejemplos utilizamos el mismo nombre de variable, pero en el segundo utilizamos `{}` y `“:”`. El segundo caso representa un conjunto de propiedades, cada una con su valor. Formalmente, es un conjunto de pares clave/valor. Para obtener los datos puedo utilizar el punto o `[]`.

Esta estructura se llama objeto o también array asociativo.

Un array asociativo es un array cuyos índices no son numéricos sino textos o claves. Se definen utilizando `{}`.

Nota: en muchos lenguajes de programación existe la distinción entre un array común (llamado también indexado y las claves son números) y un array asociativo (también llamado diccionario y las claves son strings). JS usa arrays para representar arrays indexados y objetos para representar arrays asociativos.

Existen definiciones que afirman que los objetos son llamados a veces arreglos asociativos, ya que cada propiedad está asociada con un valor de cadena que puede ser utilizado para acceder a ella.

Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre y un valor. Un valor de propiedad puede ser una función, la cual es conocida entonces como un método del objeto. Además de los objetos que están predefinidos en el navegador, se pueden definir objetos propios.

Veamos otro ejemplo de un auto:

```
let coche = {"color":"rojo", "marca":"ford","modelo":"fiesta"};
```

Para las claves puedo poner o no las "":

```
let coche= {color:"verde ",marca:"ford",modelo:"fiesta"};
```

Nota: es obligatorio poner las comillas si el nombre de la propiedad tiene espacios o es una palabra reservada.

¿Cómo accedo a los datos del array coche? Hay dos formas como señalamos:

```
alert("coche"+ coche.color); // notación del punto  
alert("coche" +coche["color"]); // notación del corchete
```

Si ponemos:

```
alert("coche" +coche[color]);
```

da error, porque asume que "color" sería una variable.

Para agregar elementos a un array asociativo, se puede hacer agregando directamente el elemento usando []:

```
let coche2= {"color":"verde ",marca:"ford",modelo:"fiesta"};  
coche2["nuevoDato"]= "un texto";  
alert(coche2["nuevoDato"]);
```

También se puede agregar usando la notación "punto":

```
coche2.nuevodata = "un texto";
```

Para recorrerlo, dado que no hay un índice, se utiliza la estructura for in:

```
for (let clave in coche) {  
  alert(clave+": " +coche[clave]); // sale la clave y el atributo  
}
```

12.2 Elementos, propiedades, métodos y this

12.2.1 Elementos, propiedades y métodos

Cuando utilizamos arrays, nos referimos a elementos del array. Cuando son objetos, hablamos de propiedades. Una propiedad podría ser una función, en cuyo caso hablamos de métodos.

Por ejemplo:

```
> let miPerro = {  
  nombre:'Benji',  
  ladrar:function(){  
    alert('gufff, gufff');  
  }  
};  
< undefined  
> miPerro.ladrar();
```

Nombre es una propiedad. Ladrar es un método. Para utilizarlo se pone la notación del punto, ya que es una propiedad y se ponen los (). Si quisiera usar [], debo agregar también ().

```
miPerro["ladrar"]()
```

En general, las propiedades son sustantivos (ej. nombre) y los métodos son verbos (ej. ladrar).

Nota: esta forma de definir el objeto se denomina “objeto literal”.

12.2.2 this

Veremos cómo hacer para que en el ejemplo anterior, si quisiera que cuando ladre, aparezca también el nombre del perro. Si simplemente utilizo el nombre en el método, no funciona:

```
> let miPerro = {  
  nombre:'Benji',  
  ladrar:function(){  
    alert(nombre + 'gufff, gufff');  
  }  
};  
< undefined  
> miPerro.ladrar();  
> Uncaught ReferenceError: nombre is not defined  
  at Object.ladrar (<anonymous>:1:55)  
  at <anonymous>:1:9
```

Si quiero acceder a una propiedad desde otro método, debo agregar la palabra this, que refiere a “este objeto”.

```
> let miPerro = {  
  nombre:'Benji',  
  ladrar:function(){  
    alert(this.nombre + 'gufff, gufff');  
  }  
};  
< undefined  
> miPerro.ladrar();
```

12.3 Introducción al concepto de clase

Supongamos que quisiéramos representar varios perros. ¿Qué características tiene un perro? Tiene nombre y sabe ladrar, según nuestro ejemplo. Se podría crear un modelo de perro con las características generales, lo que se denomina clase, luego crear perros y cada uno tener sus propios valores.

En la versión ES6 de JavaScript se pueden definir fácilmente. Veamos el código y luego lo detallaremos.

```
> class Perro{  
  constructor(nombre){  
    this.nombre= nombre;  
  }  
  ladrar(){  
    alert(this.nombre+ "gufff guffff");  
  }  
};  
< undefined  
> let miPerro= new Perro("Dogui");  
< undefined  
> miPerro.ladrar();  
< undefined
```

Al ejecutar este código aparece el nombre del perro y su ladrido.

Si utilizamos `console.table(miPerro)` se muestran los datos en formato tabla.

La clase se define usando la palabra clave `class`, seguida del nombre de la clase.

Se define el constructor (que es único) y que se ejecutará cuando se cree un objeto con la palabra `new`. Si se intenta poner más de un constructor, da error de sintaxis.

Puede además contener varios métodos, cada uno es un nombre y los parámetros, no se pone la palabra `function`.

A diferencia de las funciones, la clase debe ser declarada antes de usarse.

12.4 Método toString

En el ejemplo, agregamos:

```
alert(miPerro);
```

observamos que dice “object Object”.

Automáticamente, JS cuando debe desplegar un objeto lo pasa a string. Por ejemplo:

```
> let t = [1, 2, 3];  
undefined
```

Si ponemos:

```
> t.toString();  
sale: "1,2,3"
```

Ahora probemos estas dos instrucciones que darán el mismo resultado:

```
> alert(t);  
> alert(t.toString());
```

Generalmente, se agrega en la clase el método toString que retorna una descripción en texto del objeto:

```
class Perro{  
  constructor(nombre){  
    this.nombre= nombre;  
  }  
  ladrar(){  
    alert(this.nombre+ "gufff guffff");  
  }  
  toString(){  
    return this.nombre;  
  }  
}
```

12.5 Ejercicio: Contactos

Desarrollaremos un ejemplo para manejar contactos. Permitirá profundizar en los conceptos que estamos presentando.

Crearemos primero la clase Contacto, que tiene nombre, apellido, edad y teléfono.

```

class Contacto {
    constructor(nombre, apellido, edad, telefono) {
        this.nombre=nombre;
        this.apellido= apellido;
        this.edad= edad;
        this.telefono=telefono;
    }

    toString(){
        return this.nombre + " " + this.apellido+ " " + this.edad + " " +this.telefono ;
    }
};

```

Definimos la clase agenda, que contendrá un array donde se guardarán los contactos.

```

class Agenda {
    constructor(){
        this.lista=[];
    }
    agregar(unElemento){
        this.lista.push(unElemento);
    }
    darTodos(){
        return this.lista;
    }
};

```

Diseñaremos la página para agregar los contactos. Podría lucir similar a esta:

Contactos

Construimos el html.

```

<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title> Contactos </title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/contactos.css">
    <script src="js/contactos.js"></script>

```

```

</head>

<body>

<header>
    <h1>Contactos</h1>
</header>

<section>
    <h2>Ingreso de datos </h2>
    <article>
        <form id="formulario" action="#" method="post">
            <label for="txtNombre">Nombre: </label>
            <input type="text" id="txtNombre" >
            <br>
            <label for="txtApellido">Apellido</label>
            <input type="text" id="txtApellido" >
            <br>
            <label for="txtEdad">Edad</label>
            <input type="number" id="txtEdad" >
            <br>
            <label for="txtTelefono">Teléfono</label>
            <input type="tel" id="txtTelefono" value="">
            <input type="button" id = "boton" value="Agregar">

        </form>
        <input type="button" id = "botonConsultar" value="Consultar">

    </article>

</section>

<aside>
    <h4>Contactos ingresados</h4>
    <ul id="lista">
    </ul>
</aside>

</body>
</html>

```

El css es el mismo que definimos en el ejemplo de Tags Semánticos:

```

header{
    background-color:gray;
    width:100%;
}

```

```

nav {
    background-color: yellow;
}

nav ul li{
    display:inline;
    font-size: 1.5em;
    border-left-width: 15px;
    border-left-color: black;
    border-left-style: solid;
}

section{
    width:55%;
    background-color:darkgray;
    float:left;
    padding-left:5%;
}

article {
    background-color: orange;
}

aside{
    width:25%;
    float:left;
    padding:5%;;
}

footer{
    width:100%;
    background-color:black;
    text-align:center;
    float:left;
    color:white;
    font-size:80%;
    padding: 10px;
}

```

Agregaremos ahora la funcionalidad.

En el archivo .js indicaremos que cuando se presione el botón de "agregar", tome los datos de pantalla y agregue el contacto. Por otro lado, cuando se presione el botón de "consultar" muestre en una lista los elementos que tiene. Otra posible opción sería que automáticamente al agregar un elemento se recargue la lista. Para recorrer la lista utilizaremos la estructura for of.

El código completo es:

```

window.addEventListener("load", inicio);

class Contacto {

    constructor(nombre, apellido, edad, telefono) {
        this.nombre=nombre;
        this.apellido= apellido;
        this.edad= edad;
        this.telefono=telefono;
    }

    toString(){
        return this.nombre + " " + this.apellido+ " " + this.edad + " " +this.telefono ;
    }
}

class Agenda {
    constructor(){
        this.lista=[];
    }
    agregar(unElemento){
        this.lista.push(unElemento);
    }
    darTodos(){
        return this.lista;
    }
}

let agenda = new Agenda();

function inicio(){

    document.getElementById("boton").addEventListener("click", agregarContacto);
    document.getElementById("botonConsultar").addEventListener("click", consultar);

}

function agregarContacto(){
    let nombre = document.getElementById("txtNombre").value;
    let apellido = document.getElementById("txtApellido").value;
    let edad = document.getElementById("txtEdad").value;
    let telefono = document.getElementById("txtTelefono").value;
    agenda.agregar(new Contacto(nombre, apellido, edad, telefono));
    document.getElementById("formulario").reset();
}

```



```
function consultar(){
    let lista = document.getElementById("lista");
    lista.innerHTML="";
    let datos= agenda.darTodos();
    for (elemento of datos){
        let x = document.createElement("LI");
        let nodo = document.createTextNode(elemento);
        x.appendChild(nodo);
        lista.appendChild(x);
    }
}
```

Podría también separarse en dos archivos el código JS: uno sólo con las clases y otro con el resto del código JS.

12.5.1 Extensión: mayores

Agregar un botón que al ser presionado, arme una tabla con los datos de todas las personas mayores de 18.

Una forma posible es que cada contacto sepa si es mayor o no. Para ello en la clase Contacto agrego:

```
esMayor(){
    return this.edad>=18;
}
```

En Agenda agrego:

```
darMayores(){
    let respuesta = [];
    let datos = this.darTodos();
    for (let elem of datos){
        if (elem.esMayor()){
            respuesta.push(elem);
        }
    }
    return respuesta;
}
```

En el html agrego un botón y la tabla:

```
<input type="button" id = "botonMayores" value="Consultar Mayores">
<table id="tablaMayores">
</table>
```

En la función inicio agrego:

```
document.getElementById("botonMayores").addEventListener("click", consultarMayores);
```

Y el código:

```
function consultarMayores(){
    let tablita = document.getElementById("tablaMayores");
    tablita.innerHTML="";
    let datos= agenda.darMayores();
    for (elemento of datos){
        let fila = tablita.insertRow();
        let celda = fila.insertCell();
        celda.innerHTML=elemento;
    }
}
```

12.5.2 Extensión: ordenar

Queremos ordenar la agenda por apellido. Además de las formas que vimos anteriormente en arrays, podemos ordenar estableciendo el criterio como veremos a continuación, para lo que agregaremos la clase Agenda:

```
ordenar(){
    this.lista.sort(Agenda.compararApellido);
}

static compararApellido(primerio, segundo){
    return primerio.apellido.localeCompare(segundo.apellido);
}
```

Observar que el método `compararApellido` es static. Se los llama incluyendo el nombre de la clase primero.

Luego de agregar cada elemento podemos incluir que ordene ya la lista:

```
agregar(unElemento){
    this.lista.push(unElemento);
    this.ordenar();
}
```

Hay otras formas posibles de especificar el criterio de ordenación. Por ejemplo, podría agregarse en la clase `Contacto`:

```
compararCon(otro){
    return this.apellido.toUpperCase().localeCompare(otro.apellido.toUpperCase());
}
```

y en clase Agenda:

```
agregar(unElemento){
    this.lista.push(unElemento);
    this.lista.sort(function (a,b){
        return a.compararCon(b);
    });
}
```

12.5.3 Extensión: apellido repetido

Queremos saber si, dado un apellido, ya existe una persona con ese mismo apellido. Agrego en Agenda una función que se invocaría al agregar un apellido o al presionar un botón nuevo para consultar:

```
existeContactoApellido(apellido){
    let esta = false;
    for (let elem of this.darTodos()){
        if (elem.apellido === apellido){
            esta= true;
        }
    }
    return esta;
}
```

12.5.4 Extensión: eliminar un contacto

Agrego en agenda:

```
eliminar(contacto) {
    let seguir = true;
    for(let i = 0; i < this.lista.length && seguir; i++) {
        let c = this.lista[i];
        if(c.telefono === contacto) {
            this.lista.splice(i, 1); //para eliminar se utiliza el método splice
            seguir = false;
        }
    }
}
```

En la interfaz agregar un botón para eliminar y vincular el evento desde JS.

Nota: se podría separar en dos archivos JS: uno con la definición de las clases y el otro con la interacción con la página.

13 Clases y objetos

13.1 Asociación

Veremos el ejemplo de asociación a través del siguiente ejemplo.

Tenemos la clase Persona y la clase Mascota. Cada mascota tiene una persona que es su dueño. Un objeto puede contener otro. Veremos cómo definir las clases para representar esta asociación o vínculo entre las clases.

La implementación de las clases mencionadas podría ser:.

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad= edad;
  }
  presentar() {
    alert("hola soy " + this.nombre + " y tengo "+this.edad);
  }
}

class Mascota {

  constructor(nombre, dueño) {
    this.nombre = nombre;
    this.dueño = dueño;
  }

  saludar() {
    alert("hola soy " + this.nombre + " y mi dueño es " + this.dueño.nombre);
  }
}
```

Puedo crear objetos de la clase Persona y Mascota. Luego podemos asociarlos, para representar que una mascota tiene un dueño:

```
let propietario = new Persona("Ana",12);
let perro = new Mascota("Fido",propietario ); // asocio
perro.saludar();
```

Nota: a diferencia de las funciones, las clases deben definirse antes de usarse. Esto se debe a que JS pone en memoria ("izar", "hoisting") las declaraciones de variables y de funciones antes de ejecutar cualquier otro segmento de código y esto permite, por ejemplo, utilizar una función antes

de declararla en el código. Las declaraciones de clases no son "izadas", por lo que deben definirse antes de usarse.

13.2 Ejemplo integrador: Consultora

Una consultora desea realizar un relevamiento sobre el trabajo en Uruguay y los sueldos.

En el sistema se debe poder:

1. registrar tarea: cada tarea tiene un nombre, descripción y tipo (1-administrativo, 2-mano de obra, 3-especializado, 4-altamente especializado, 5-otro).
2. registrar empresa: se indica, nombre, dirección y rut.
3. registrar puesto de trabajo. Se indica la empresa, detalle, la tarea y el salario ofrecido por la empresa
4. consulta de sueldo promedio. Se indica una tarea y debe mostrarse el promedio de sueldos ofrecidos por las empresas para esa tarea.
5. Tipo más solicitado: indicar el tipo de tarea con más puestos.
6. Listado de sueldos: mostrar los trabajos ofrecidos, ordenados decreciente por sueldo.

Definiremos las clases y métodos necesarios.

La implementación de las clases podría ser:

```
class Empresa {

    constructor(nombre,direccion,rut) {

        this.nombre=nombre;
        this.direccion= direccion;
        this.rut=rut;
    }
    toString(){
        return this.nombre +" en: "+ this.direccion+ " RUT: "+ this.rut ;
    }
}

class Tarea {

    constructor(nombre,descripcion,tipo) {
        this.nombre=nombre;
        this.descripcion= descripcion;
        this.tipo=tipo;
    }
    toString(){
        return this.nombre +" "+ this.descripcion+ " tipo: "+ this.tipo ;
    }
}
```

```

class Puesto{
    constructor(empresa, tarea, detalle, sueldo){
        this.empresa=empresa;
        this.tarea= tarea;
        this.detalle= detalle;
        this.sueldo=sueldo;
    }
    toString(){
        return this.empresa + "\n " + this.tarea + "\n " + this.detalle+ "\n " + this.sueldo ;
    }
}

class Sistema {
    constructor(){
        this.listaEmpresas=[];
        this.listaTareas=[];
        this.listaPuestos=[];
    }

    agregarEmpresa(unElemento){
        this.listaEmpresas.push(unElemento);
    }

    agregarTarea(unElemento){
        this.listaTareas.push(unElemento);
    }

    agregarPuesto(unElemento){
        this.listaPuestos.push(unElemento);
    }

    darEmpresaIndice(unIndice){
        return this.listaEmpresas[unIndice];
    }

    darTareaIndice(unIndice){
        return this.listaTareas[unIndice];
    }

    toString(){
        return "Empresas " + this.listaEmpresas + " tareas " + this.listaTareas+ " Puestos " +
this.listaPuestos;
    }

    ordenar(){
        this.listaPuestos.sort(Sistema.compararSueldo);
        return this.listaPuestos;
    }
}

```

```

static compararSueldo(first, second){
    return second.sueldo-first.sueldo;
}

promedioTarea(unaTarea){

    let suma = 0;
    let cant = 0;
    for (let elem of this.listaPuestos){
        if(elem.tarea==unaTarea){
            suma += elem.sueldo;
            cant++;
        }
    }
    let retorno ="Sin datos";
    if (cant>0){
        retorno = suma/cant;
    }
    return retorno;
}

tipoMasSolicitado(){
    let tot = [];
    for (let i = 1; i <=6; i++){
        tot[i]=0;
    }
    for (let elem of this.listaPuestos){
        let tipo= elem.tarea.tipo;
        tot[tipo]++;
    }
    let max = 0;
    let posMax = 0;
    for (let i = 1; i <=6;i++){
        if (tot[i]>max){
            max = tot[i];
            posMax= i;
        }
    }
    return posMax;
}
}

```

13.3 Referencias y objetos

Cuando se define una variable, como `let x = 10`; `let nombre= "Ana"`; o `let autor = {nombre:'Ana', apellido:'García'}`, lo que ocurre es:

- si el valor es primitivo (número, string o boolean), la variable contiene el valor directamente
- si el valor es un objeto, la variable contiene la dirección de memoria del objeto. Se dice que la variable apunta o refiere al objeto.

Ejemplos:

```
let x = 2; // x contiene el valor primitivo 2
let y = { a: 2 } // y refiere al objeto {a: 2}

// "Copio" variables
let x2 = x;
let y2 = y;
let y3 = y;

// Modifico las copias
x2 = 3;
y2 = { a: 3 };

// Verificación
x; // 2 <- x no se modifica, contiene un valor primitivo
y; // { a: 2 } <- y no se modifica, porque y2 no apunta al mismo objeto

y3.a = 4;
y; // { a: 4 } <- el objeto referido por y e y3 es modificado (ambos apuntan al mismo lugar de memoria)
```

Las reglas también se aplican a las propiedades de objetos.

Ejemplo:

```
let chofer= {
  nombre: 'Juan'
};

let auto= {
  color: 'rojo',
  chofer: chofer
};

chofer.nombre = 'Alberto';
auto.chofer.nombre; // 'Alberto'
```


13.4 Pasaje de parámetros

Cuando se pasa una variable a una función como parámetro, se copia el valor. Se dice que JS “pasa por valor”.

Ejemplo:

```
function sum(a, b) {  
  a = a + b;  
  return a;  
}  
let x = 2;  
sum(x, 3); // retorna 5  
x; // 2 <- pero x sigue siendo 2
```

Cuando se trabaja con objetos, se copia la referencia del objeto. Esto implica que se puede modificar el objeto referenciado.

Ejemplo:

```
function sumar(a, b) {  
  a.x += b;  
}  
  
let obj = { x: 2 }  
sumar(obj, 3);  
obj.x; // 5 <- El objeto referido se modifica
```

13.5 Comparación de objetos

La comparación de dos objetos devolverá true si apuntan al mismo objeto, o sea, tienen la misma referencia. Dos objetos del mismo tipo, con los mismos valores que lucen idénticos, no serán iguales si no refieren a la misma posición en memoria.

```
> let original= {nombre:'Ana'};  
< undefined  
> let copia = original;  
< undefined  
> copia === original  
< true  
> let otro= {nombre:'Ana'};  
< undefined  
> copia === otro  
< false
```

13.6 Objeto window

JS se ejecuta en un ambiente que es usualmente un navegador. También se utiliza para codificar aplicaciones del lado del servidor.

El ambiente define un “objeto global” denominado window. Las variables globales definidas con var son propiedades de ese objeto. Las variables definidas con let no son propiedades.

Ejemplo:

```
> var a = 1;
< undefined
> a;
< 1
> window.a;
< 1
> window['a'];
< 1
> let z = 1; // let no define propiedades del objeto window
< undefined
> window.z
< undefined
```

a y window.a son la misma variable.

navigator y window.navigator son lo mismo, document y window.document son lo mismo.

```
> document === window.document;
< true
> navigator === window.navigator
< true
```

Las funciones predefinidas son métodos del objeto global window:

```
> parseInt('10 datos');
< 10
> window.parseInt('10 datos');
< 10
> alert === window.alert
< true
> prompt === window.prompt
< true
> window.addEventListener === addEventListener
< true
```

13.7 Clase Object

Todos los objetos heredan las propiedades y métodos de una clase especial que se llama Object. Es equivalente:

```
> let o = {}; // creación de un objeto vacío
< undefined
> let o = new Object(); // otra forma de hacer lo anterior
< undefined
```

13.8 Clase Number

Puede ser usada para transformar strings en números, pero se recomienda usar parseInt o parseFloat

```
> let n = Number('3.1416');
< undefined
> n
< 3.1416
> typeof n;
< "number"
> let n = parseInt('3.1416'); // convierte un string en un número
< undefined
> n
< 3
> let n = parseFloat('3.1416'); // convierte un string a un float
< undefined
> n
< 3.1416
```

Hay valores especiales: MAX_VALUE, MIN_VALUE, MAX_SAFE_INTEGER y MIN_SAFE_INTEGER:

```
> Number.MAX_VALUE;
< 1.7976931348623157e+308
> Number.MIN_VALUE;
< 5e-324
> Number.MAX_SAFE_INTEGER
< 9007199254740991
> Number.MIN_SAFE_INTEGER
< -9007199254740991
```

Algunos métodos útiles: toFixed() y toString(). Analizar los ejemplos:

```
> let n = 123.456;
< undefined
```

```
> n.toFixed(1);
< "123.5"
> let n = 255;
< undefined
> n.toString();
< "255"
> n.toString(10);
< "255"
> n.toString(16); // lo muestra en base 16
< "ff"
> (3).toString(2); // lo muestra en binario
< "11"
> (3).toString(10);
< "3"
```

13.9 Clase Date

La clase Date permite definir objetos para representar fechas.

Si se crea sin parámetros toma la fecha actual:

```
> let date = new Date();
< undefined
> date;
< Sat Dec 08 2018 15:29:47 GMT-0300 (hora estándar de Uruguay)
```

Probar otras formas de crear fechas:

```
> d = new Date('1988 03 10'); //con un string
< Thu Mar 10 1988 00:00:00 GMT-0300 (hora estándar de Uruguay)
> d= new Date('1991 03 15 21:30');
< Fri Mar 15 1991 21:30:00 GMT-0300 (hora estándar de Uruguay)
> new Date('2017 1 2 8:30');
< Mon Jan 02 2017 08:30:00 GMT-0300 (hora estándar de Uruguay)
```

Se indica por orden: año, mes, día, hora, minutos, segundos y milisegundos.

Algunos métodos útiles siendo:

```
d= new Date('1988 03 10 22:32');
```

d.getDay()

4 // el valor 0 corresponde a domingo, 1 a lunes, etc.

d.getMonth()

2 // el valor 0 corresponde a enero, 1 a febrero, etc.

Ejercicio: calcular cuántas veces cae el 1ero de enero en cada día de la semana entre los años 2010 y 2050.

```
//En el array quedarán las frecuencias
let dias = [0,0,0,0,0,0,0];
for (let anio = 2010; anio <= 2050; anio++) {
  dias[new Date(anio, 1, 1).getDay()]++;
}
```

14 JSON

14.1 Introducción a JSON

JSON (JavaScript Object Notation) es un estándar para convertir un objeto en una representación de string y sirve para intercambio de información con un servidor.

Si queremos pasar un objeto a formato JSON:

```
let jsonStr = JSON.stringify(obj);
```

Para pasar de formato JSON a un objeto JS:

```
let jsObj = JSON.parse(jsonStr);
```

En los ejemplos siguientes, se verán las representaciones en JSON de un objeto simple, de un array, de un objeto simple y de un objeto que contiene un array de objetos. Lucen similar a los objetos, con "" en los nombres de las propiedades.

Ejemplos:

```
> let ejemploObject = {x:12, y:30};  
< undefined  
> JSON.stringify(ejemploObject);  
< '{"x":12,"y":30}'
```

```
> let miArray = ['Lunes', 'Martes', 'Miércoles'];  
< undefined  
> JSON.stringify(miArray);  
< '["Lunes","Martes","Miércoles"]'
```

```
> let otroObject = {nombre:'Beatles',  
  albums:[  
    {nombre:"Abbey Road", anio:1969},  
    {nombre:"Let It Be", anio:1970}  
  ]  
};  
< undefined  
> JSON.stringify(otroObject);  
< '{"nombre":"Beatles","albums":[{"nombre":"Abbey Road","anio":1969},{"nombre":"Let It Be","anio":1970}]}'
```

```
> let metallica= {nombre:'Metallica', albumes:[{nombre:"Master of Puppets", anio:1986},  
{nombre:"Black Album", anio:1991}]};  
< undefined  
> let metallicaJSON = JSON.stringify(metallica);  
< undefined
```

El formato JSON guarda propiedades y valores como string. Cuando utilizamos el método `JSON.parse()`, se tiene un objeto JS real. Recupero:

```
> let obj = JSON.parse(metallicaJSON); // JSON -> object  
< undefined  
> obj.nombre; // es un objeto  
< "Metallica"
```

14.2 Local Storage

En el browser existe "LocalStorage", que puede ser usada como una pequeña base de datos para aplicaciones web y permite guardar pares clave/valor en formato texto. Si tenemos objetos de JS, se puede convertir JSON y almacenarlos. Luego, al tomar datos del repositorio, convertimos de JSON a JS.

Por ejemplo, se definen personas con sus autos, se almacenan en un array y se guarda en el LocalStorage:

```
let p1 = {nombre:"Ana", direccion:"18 de julio 1234"};  
let auto1 = {propietario : p1, chapa:100};  
let p2 = {nombre:"Luis", direccion:"Av Italia 2222"};  
let auto2 = {propietario : p2, chapa:300};  
  
let lista=[];  
lista.push(auto1);  
lista.push(auto2);  
localStorage.autos = JSON.stringify(lista);
```

Puedo observar como quedó almacenado abriendo en el navegador con F12, en la pestaña Application. Allí elegir LocalStorage.

Para recuperar los datos, podemos utilizar:

```
lista = JSON.parse(localStorage.autos);
```