Estructuras de Control:

- Secuencias de instrucción

- Operadores:

- **Unitarios (! - +)**
- **Binarios (+ * / < > <= = == % ....)**
- **Ternario (? :)**

- Condicionales (if, switch, ? :)

- Saltos de línea

- (GOTO, en desuso)
- Saltos hacia atrás:
  - Bucles: for, **while**, do-while,
  - Saltos en el bucle: continue
- Saltos hacia adelante:
  - break,
  - Control de excepciones:
    - try - catch - finally.

- Función:

- No importa el tipo: TODAS pueden recibir valores (parámetros o argumentos) separados por comas. Si no, que no reciben parámetros (reciben **void).**
- No importa el tipo: TODAS pueden devolver un valor (**return**). Si no devuelve un valor, se les suele llamar procedimientos (**void**).

Se pueden anidar invocaciones a placer, es decir, una función puede llamar aun método que puede llamar a una funciana estática, etc...

- **Funciones estáticas (métodos estáticos):**
  - Son funciones independientes, no depende de ningún objeto ni propiedad externa (como mucho variables estáticas). En el fondo son funciones globales, muy generales, como una función matemática, una de parseo, un Log…. Podrían estar en cualquier parte. NO PUEDEN acceder a variables miembro.
- **Métodos (funciones miembro de clase/estructura):**
  - Es como una acción/mensaje de una instancia, por lo que puede acceder a sus valores. Internamente se pueden acceder a todas las variebles miembro y a otros métodos (sean **private** o **public**,) externamente sólo a las **public**.

- Constructores…

Funciones que configuran o inicializan las variable miembro de un objeto/instancia. Estos no reservan memoria. Quién reserva memoria es **new**. Se suelen usar junto.

Digrama de flujo

Formas de control de datos

Variables globales: SÓLO hay UNA por APLICACION

Variables estáticas, es decir, que SÓLO hay UNA por APLICACION (parecidas a las globales)

Variables locales:

- Con **var**: Locales a la función
- Con **let:** Existen SÓLO dentro de un bloque (un if, un for, una función...)

Variables miembro:

- Existen una por cada instancia de una estructura ó por cada instancia de una clase (objetos).

Como parámetros:

- Paso por valor:

- **Tipos primitivos, strings, structs.**
- Se hace una copia en memoria (como un clon) y se pasa como argumento ese clon, ese valor.

- Paso por referencia:

- Arrays, clases (objetos), colecciones por defecto se pasan como referencia
- Se pasa la dirección de memoria al mismo objeto, osea, el mismo objeto/ instancia/variable, es decir, una referencia a la misma variable.
- Si queremos pasar por referencia un tipo primitivo o struct, hay que usar la palabra clave **ref ó out.**

Estructuras de datos

- Tipos primitivos:

- **1byte: boolean,**
- **8 bytes: Number**
- **null**
- **undefined**
-

- String: nativo / objeto tiene métodos pero se comporta como un primitivo

- Arrays

- Objects:

  Existentes: Date, String, DOMElement…

  Propios: { “propiedad”: “valor”, }

- Clases:
  - Propias:
    - Clases normales
  - Ya creadas:
    - Date
  - Colecciones: Array, Maps
- Programación Orientada a Objetos:
  - Encapsulación (**private, public, protected**…)
  - Herencia
  - Polimorfismo

Diagrama de clases

Diagrama de datos / bb.dd.