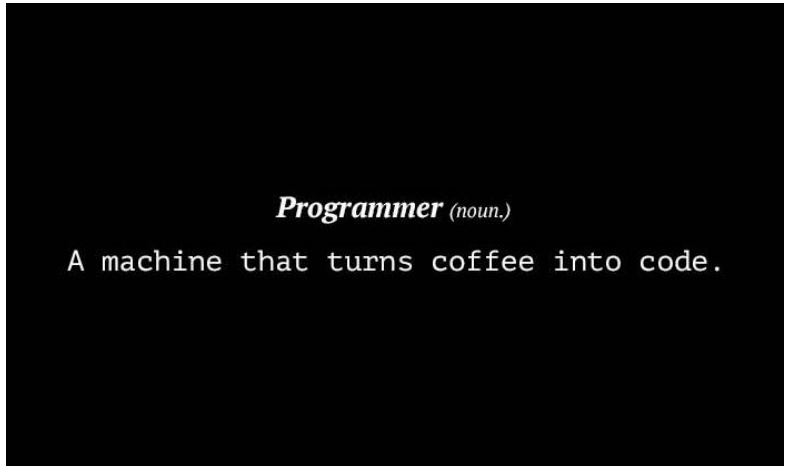
PROGRAMACIÓN

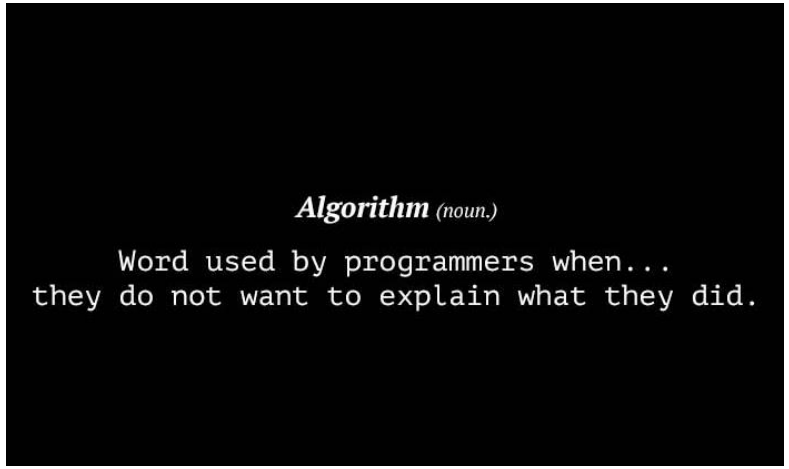
CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Álvaro Holguera – CIFP José Luis Garci (Madrid)

¿QUÉ ES PROGRAMAR?

- La tarea de crear un programa o programar consiste en escribir detalladamente las instrucciones que debe seguir una computadora para realizar una tarea.
- Estas instrucciones deben escribirse en un lenguaje que la computadora pueda entender (lenguaje máquina), ya sea en forma directa o tras una traducción realizada por un "intérprete" también llamado compilador.
- Algo importante a tener en cuenta es que la computadora carece de "sentido común" y, por tanto, no habrá en el programa nada que nosotros no hayamos previsto, por lo que al programar debemos dejar todas las situaciones posibles bien cubiertas. Los algoritmos que creemos deben ser:
  - **Precisos**: el orden de la secuencia no puede ser alterado.
  - **Finitos**: debe tener un inicio y un final
  - **Correcto** tanto en la presentación formal (debe ajustarse a un estándar de programación) y en el resultado (la salida final debe ser la esperada)
  - **Eficiente**: debe optimizar los resultados de almacenamiento y de procesamiento





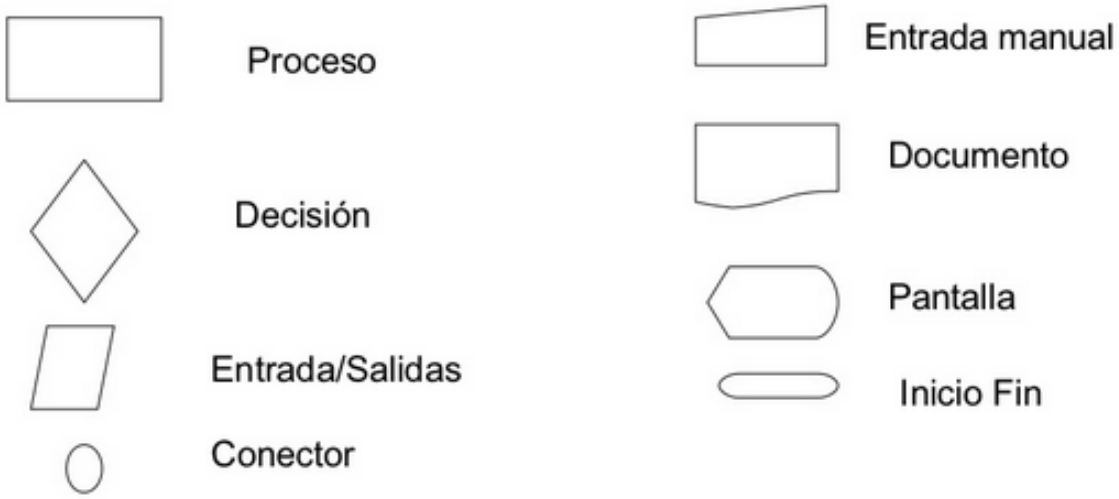
Fuente: https://www.hongkiat.com/blog/programming-jokes/

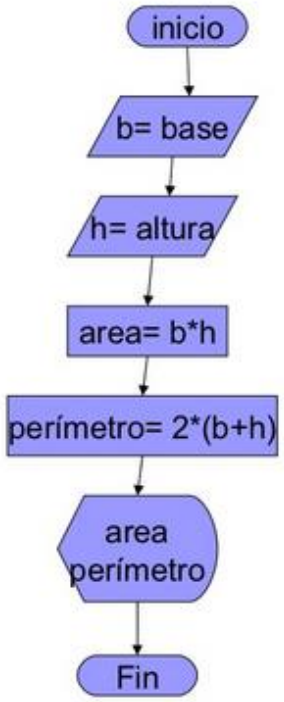
DIAGRAMA DE FLUJO DE DATOS

El diseño de algoritmos consiste en la creación de un conjunto de reglas para desarrollar un cálculo o resolver un problema.

Programar algoritmos es lo más parecido a elaborar diagramas de flujos, en los que una serie de tareas se van desarrollando para lograr un objetivo final.

Reflejan de forma gráfica la secuencia de pasos realizados para la resolución de un determinado problema. Son independientes del lenguaje de programación empleado.





Ejemplo: Calcular el área y perímetro de un rectángulo

APLICACIÓN PRÁCTICA DE UN DIAGRAMA DE FLUJO: ALGORITMO DE SHELDON PARA HACER AMIGOS

https://www.youtube.com/watch?v=3sU6KIxjqMI

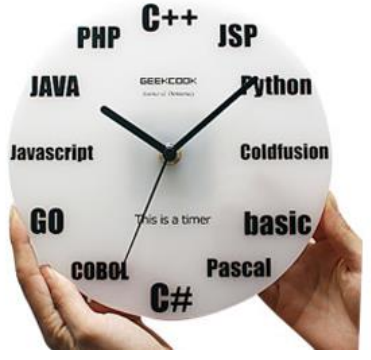
LENGUAJES DE PROGRAMACIÓN

- Los lenguajes de programación (Javascript, C#, PHP, Java, Pascal, Visual Basic, etc.) son herramientas que nos permiten crear programas y software, creando una vía de comunicación entre el ser humano y las máquinas

- Facilitan la tarea de programación ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas y resultan independientes del modelo de computadora a utilizar.

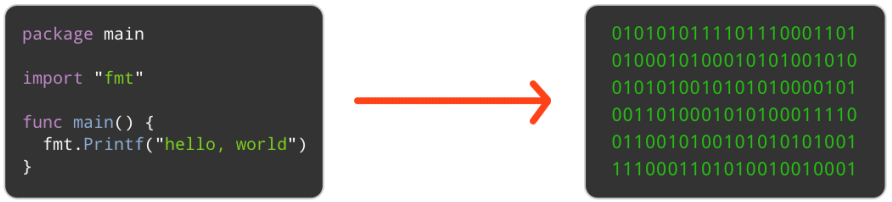
- Se suele distinguir entre:
  - Lenguajes de alto nivel, son independientes de la máquina y requieren por tanto de un intérprete o compilador.
  - Lenguajes de bajo nivel, totalmente dependientes de la máquina y muy alejados del lenguaje humano.

Existen muchos programas para editar código, aunque cualquier editor básico de texto (como el bloc de notas) sería suficiente para crear nuestros propios scripts, pero es recomendable contar con un buen programa que ayude con la sintaxis del lenguaje, o bien con un entorno de desarrollo integrado (IDE)



LENGUAJES INTERPRETADOS VS. LENGUAJES COMPILADOS

- Todo lenguaje de programación tiene que ser interpretado por la máquina. Eso implica hacer la conversión de los códigos que escribe el programador a unos y ceros comprensibles por el ordenador. Al realizar este proceso, existen dos tipos de lenguajes:
  - **Interpretado**: la conversión se realiza al mismo tiempo que se ejecuta el script.
    - Ventajas: el tiempo entre la escritura del código y la ejecución disminuye.
    - Inconvenientes: requiere un programa que interprete (por ejemplo, el navegador web)
  - **Compilado**: requiere de un paso previo antes de ser ejecutado por la máquina, la denominada "compilación"
    - Ventaja: su ejecución es más rápida y eficiente, ya que está en lenguaje máquina.
    - Inconvenientes: una vez compilado solo funciona en el SO para el que ha sido compiladoEjemplos de

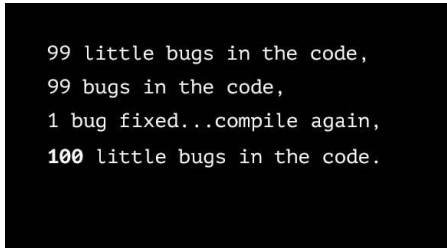


Lenguaje de alto nivel que entiende el programador

Lenguaje de máquina que entiende el procesador

Ejemplos de lenguajes compilados: C, C++, Java y C#, entre muchos otros.

Ejemplos de lenguajes interpretados: Ruby, Python y JavaScript, entre muchos otros.



LENGUAJES DE LADO SERVIDOR O CLIENTE

- En el desarrollo web, es importante distinguir entre los lenguajes que se ejecutan en el servidor (como PHP, ASP ó PERL) y los que se ejecutan en el cliente, es decir en el navegador del usuario (como javascript. Recordemos que HTML no es un lenguaje de programación)

- Los lenguajes de **lado cliente** son los que pueden ser interpretados por el navegador.
  - Ej.- `<script>alert("Hola mundo")</script>`
  - Tiene la ventaja de funcionar de forma autónoma, pero carga el proceso sobre la máquina del cliente.
  - Además, es dependiente de la versión del navegador para funcionar, por lo que escapa a nuestro control.
  - Otra desventaja es que el código utilizado es visible por el usuario, mientras que en los lenguajes de lado servidor sólo "ve" el resultado, pero no el script que ha llevado a ese resultado.

- Los lenguajes de **lado servidor** son aquellos que son reconocidos, ejecutados e interpretados por el servidor y es devuelto en un lenguaje comprensible para el navegador.
  - Ej.- `<?php echo "Hola mundo" ?>` Es una orden en PHP que nuestro navegador no es capaz de procesar por sí mismo.
  - Tienen la ventaja de liberar al ordenador cliente del proceso, pero la desventaja de hacerlo dependiente del servidor (normalmente de pago, alojado por una empresa de *hosting*)
  - Además, permiten acceder a recursos del servidor que un lenguaje cliente no podría (por ejemplo, variables POST o a los archivos alojados)

CONCEPTOS BÁSICOS

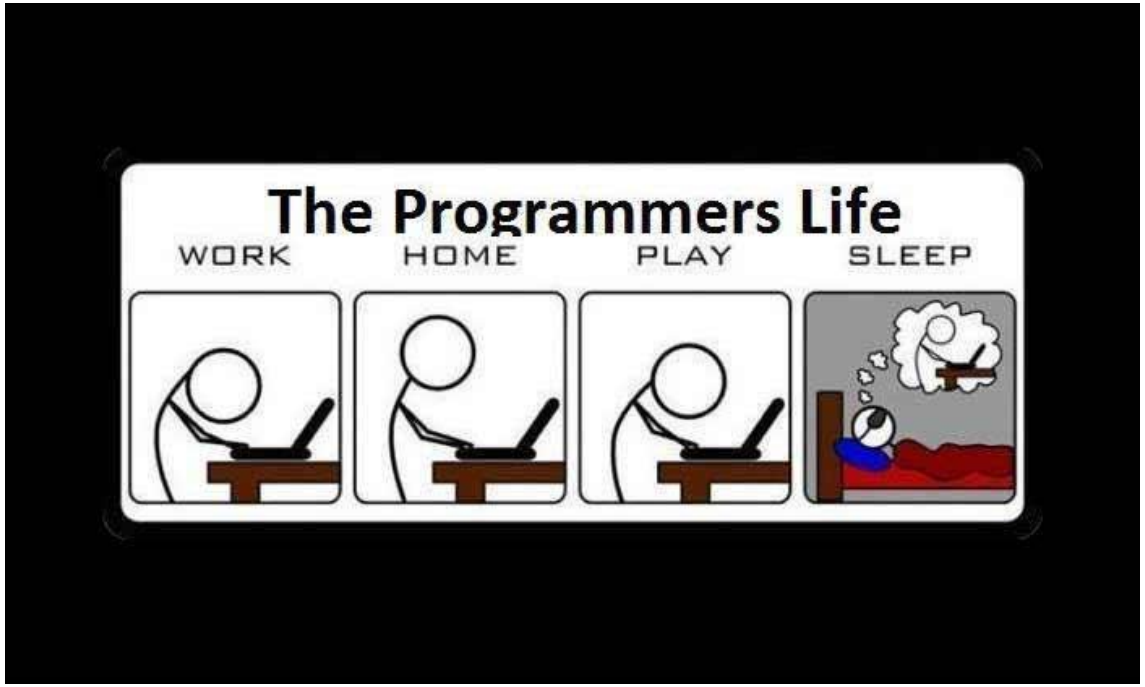
ELEMENTOS QUE SON COMUNES A CUALQUIER LENGUAJE DE PROGRAMACIÓN

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Hay una serie de conceptos básicos que tenemos que comprender, ya que son independientes del lenguaje de programación que estemos utilizando.

Son estos 4:

1. **Tipos de datos:** Constantes vs Variables
2. **Operadores**
3. **Estructuras de control**
4. **Funciones**



CONSTANTES VS. VARIABLES. ÁMBITO

- Los datos en un programa, a la hora de ser definidos, pueden ser de dos tipos:
  - **Constantes**: un valor que no se modifica a lo largo de toda la ejecución del programa. Se suelen escribir en mayúsculas.
    - Los lenguajes suelen tener unas constantes predefinidas
    - En c# se utiliza el tipo de campo "const" para declarar una constante
  - **Variable**: un dato referenciado por un identificador y cuyo valor puede cambiar a lo largo de la ejecución del código

- Una variable se compone de dos elementos:
  - Nombre. No puede cambiar, y dos variables no pueden tener el mismo nombre (dentro del mismo ámbito). No debe tener espacios en blanco ni se recomienda que empiecen con un número
  - Valor: cada variable crea un apunte en la memoria del programa y le asigna el valor que indiquemos. Ese valor podrá ir cambiando a lo largo de la ejecución

Ámbito (scope) de las variables

- Al crear una variable, ésta estará disponible solo dentro de su denominado "ámbito"
  - Por ejemplo, si declaramos una variable dentro de una función, no estará disponible fuera de ella
- Según su ámbito de aplicación, puede ser:
  - Una variable **local** sólo estará disponible en la porción de código donde ha sido declarada (por ejemplo, dentro de una función)
  - Una variable **global** estará disponible en todas las porciones del script
- En **POO**, debido a una característica de las clases conocida como "encapsulamiento", podemos crear variables de diferentes tipos:
  - Públicas: accesibles desde otras clases
  - Privadas: accesibles solo desde dentro de la clase
  - Protegidas: un tipo de variable privada, que permite acceder también desde las clases heredadas
  - Estáticas: variables que no van a cambiar de valor nunca, y son compartidas por todas las instancias de la clase

TIPOS DE VARIABLES SEGÚN SU CONTENIDO

- Por lo general, al comenzar nuestros scripts declararemos las variables, asignándoles un valor en ese momento o más adelante.
- Es importante saber qué tipo de datos van a contener esas variables, lo que determinará qué podremos hacer con ellas.
  - En algunos lenguajes, como en c#, debemos especificar qué tipo de variable es en el momento de declararla
- Si nos atenemos a la naturaleza del contenido, tanto las constantes como las variables pueden adoptar varios tipos:
  - **Numéricos** . Principalmente se distinguen dos tipos de números (aunque hay más)
    - Enteros (integer)
    - Decimales (float)
  - **Cadenas** (string) de texto: delimitados por comillas (podemos usar "\" para introducir comillas en la cadena)
  - **Booleanos** (bool): Pueden adoptar dos valores: true/false
  - **Arrays**: tablas de datos asociativos. Una misma variable puede contener varias variables, o incluso varios arrays asociados.
- En programación orientada a objetos (POO), un tipo de variable específico son los **Objetos,** que instancian la clase y contienen dentro todas sus variables y métodos (las veremos más adelante)

VARIABLES EN UNITY

- Gracias a la herencia de los métodos contenidos en la clase principal de Unity "MonoBehaviour", tenemos disponibles algunas variables específicas.
- Algunas de las más utilizadas son:
  - Vector2 / Vector 3: indican posiciones en el espacio del juego, definidas por los valores de los ejes X,Y y, en el caso de Vector 3, también Z
  - Quaternion: representa rotaciones en los diferentes ejes
  - Transform: contiene los parámetros de posición, rotación y escala del GameObject
  - GameObject: nos permite acceder a todos los componentes del GameObject
- Los dos últimos, Transform y GameObject, tienen la peculiaridad de que no son variables como tal, sino estructuras con sus propias propiedades y métodos
  - Declararemos variables de tipo Transform o GameObject cuando queramos acceder o cambiar los parámetros que contienen, por ejemplo mover un objeto a una posición de tipo Vector3, o modificar sus componentes.

```
public class myScript : MonoBehaviour {

    private Vector2 nombreVar1;
    private Vector3 nombreVar2;
    private Quaternion nombreVar3;
    private Transform nombreVar4;
    private GameObject nombreVar5;

    // Use this for initialization
    void Start () {
        //Damos un valor a la variable de tipo Vector 3
        nombreVar2 = new Vector3(0, 0, 0);

    }

}
```

Este tipo de variables solo pueden contener valores que coincidan con su naturaleza.

Por ejemplo, una variable de tipo Vector3 solo puede contener objetos de ese tipo

Javascript (var)

```
//Declaramos una variable local
var nombre;

//Le asignamos un valor
nombre = "Pedro"

//Si la definimos fuera de una función,
será una variable global
var nombre = "Pepe";

function miFuncion() {
    console.log(nombre);
}
```

si se declara una variable sin "var" será global, disponible dentro de las funciones

```
//Ejemplo de Array
var miArr = [];
miArr[] = "Mi primer valor";
miArr[] = "mi segundo valor" ;
console.log(miArr[0];

//otra forma de crear el array
var coches= ["Saab", "Volvo", "BMW"];
```

PHP ($)

```
//Creamos una variable y la mostramos
$nombre = "Paco";
echo $nombre;

//Si queremos que sea una variable global
$nombre = "Pepe";

function miFuncion() {
    global $nombre;
    echo $global;
}


//Declaramos una constante
define("NOMBRE", "Pepe");
echo NOMBRE;
```

Estará disponible en todo el script (por ejemplo mediante un include("archivo")

```
//Ejemplo de Array
$foo = array();
$foo[] = "mi primer elemento";
$foo[] = "mi segundo elemento";
echo $foo[0];


//otra forma de crear el array
$foo = array("1" => primer valor; "2" =>
"segundo valor"; "3" => "tercer valor");
```

C# para Unity

Al ser POO en C# debemos indicar el ámbito de la variable (si no lo hacemos entiende que es privada) seguida del tipo y el nombre

```
//Declaramos una variable de tipo privada, de tipo cadena de texto y llamada "nombre"
private string nombre;

//Le asignamos un valor, por ejemplo en la ejecución del método Start
void Start () {
    nombre = "Pepe";
    //Sacamos el valor de la variable a consola
    Debug.Log(nombre);
}
```

Para crear un array en Unity, tenemos que añadir unos corchetes al final del tipo de variable, NO al final del nombre de la variable

```
//Ejemplo de Array
public string[] miArr;
```

Si el array es público, podremos indicar en Unity el nº de elementos, si no, lo podemos indicar mediante el código

```
//Indico que mi array recién creado tendrá 5 elementos
miArr = new string[5];

//Ya puedo introducir valores en mi array
miArr[0] = "Mi primer valor";
miArr[1] = "mi segundo valor" ;
Debug.Log(miArr[0];
```

OPERADORES

- Los operadores nos permiten ejecutar operaciones tanto con datos numéricos como con cadenas de texto.
  - Muchos son comunes a todos los lenguajes

- Tipos de operadores:
  - **Aritméticos**: sumas (+), restas (-), divisiones (/), multiplicaciones (*), potencias (**)
  - **Alfanuméricos**: principalmente usados para concatenar cadenas de texto ("+" en javascript y C#, ó "." en PHP)
  - **Relacionales**: permiten establecer relaciones entre unos valores y otros:
    - Igual a "==" (opcionalmente "===" que no solo compara los contenidos sino también el tipo de variable)
    - Distinto a "!="
    - Mayor que ">", Menor que "<", Mayor o igual que ">=", etc.
  - **Lógicos**
    - "!" ó "not" para la negación
    - "&&" ó "and" para la conjunción
    - "||" para la disyunción

EJEMPLOS CON OPERADORES

- Aritméticos:

```
A = 3;
B = 2;

X = A+B
//donde X valdrá 5

X = A*B
//donde X valdrá 6

X = A/B
//Donde X valdrá 1,5

X = A**2
//Donde X valdrá 9
```

- Alfanuméricos:

```
A = "Hola";
B = " mundo";

X = A+B
//donde X valdrá "Hola mundo"
```

- Relacionales:

```
Si x == A
   instrucciones
Si no
   instrucciones
Fin si
```

- Lógicas:

```
//Comprobamos si la variable "a" existe
Si !a entonces
   la variable está vacía
Fin si
//Comprobamos si se cumple una doble condición
Si a == 3 && b == 3
   A y B son iguales a 3
Fin si

//Una estructura condicional muy habitual es:
Variable = (condicion) ? "Sí se da la condicion" : "No se da la
condición";
```

```
    //Creamos dos variables de tipo int
    public int num1 = 5;
    public int num2 = 4;

    //Declaramos otra variable que contendrá el resultado
    private int result;

void Start () {
        //Sumamos los 2 números y asignamos el resultado
        result = num1 + num2;

        //Creamos operadores lógicos
        if(result <= 9)
        {
            Debug.Log("El resultado es menor que 10");
        }
        else if(result > 10)
        {
            Debug.Log("El resultado es menor que 10");
        }
        else
        {
            Debug.Log("El resultado es igual a 10");
        }
    }
```

En este operador relacional en C# para Unity, comprobamos si el resultado de una suma es mayor o menor que 10 (si no es ni mayor ni menor, lógicamente es igual a 10).

Es importante ser preciso en las comprobaciones: ejemplo, un nº puede ser menor que 10 (<10) o menor o igual que 9 (<=9), en ambos casos el resultado es el mismo.

Si deseamos comprobar una igualdad, usaremos "=="

Siempre que se vayan a hacer varias comprobaciones seguidas, con una opción final por defecto, se recomienda usar la sentencia "switch" en lugar de "if"

ESTRUCTURAS DE CONTROL

ESTRUCTURAS DE CONTROL

- En Programación Estructurada (PE) se utilizan un limitado número de estructuras de control que regulan el flujo de datos y hacen más sencilla la programación
- Podemos hablar de 3 tipos de estructuras básicas:
  - **Secuenciales**: conjuntos de instrucciones ejecutadas en el mismo orden que han sido escritas
    - Es importante recordar que cada fila de instrucción debe terminar con un punto y coma;
  - **Selectivas**: se evalúan las condiciones y en función del resultado se realizan una acción u otra. Para ello s utilizan expresiones lógicas (como "if", "else", "case", etc.). Pueden ser:
    - Simples (una sola condición)
    - Múltiples (varias condiciones que se van ejecutando secuencialmente ("if" + "else if", etc.), normalmente terminando con una acción por defecto ("else")
      - Cuando se hace una comprobación múltiple sobre una misma variable, es más eficiente usar la estructura "switch"
  - **Repetitivas**: secuencias de instrucciones que se repiten un número determinado de veces.
    - While
    - For

EJEMPLOS CON ESTRUCTURAS DE CONTROL (JAVASCRIPT)

- Secuenciales:

```
<script>

  var texto;

  texto = "Hola";

  texto += " ";

  texto += "mundo";

  console.log(texto);

//La consola mostrará
"Hola mundo"

</script>
```

- Selectiva múltiple:

```
//Ejecutamos unas instrucciones dependiendo del
valor de una variable
var X = 3;

if(X > 4) {
   console.log("X es mayor que 4");
}
else if(X == 3) {
   console.log( "X es igual a 3");
}
else {
   console.log( "X no es ni mayor que 4 ni es
igual a 3");
}

//Sistema opcional mediante "switch":
switch (X)  {
      case 3:
            console.log("X es igual a 3");
            break;
       case 4:
            console.log("X es igual a 4");
            break;
       default:
            console.log( "X no es ni 3 ni 4");
}
```

- Repetitivas:

```
//Crearemos un bucle que
finalizará cuando se de la
condición
var X = 0;
while(X < 10)
{
  console.log(X);
  //Sumamos 1 al valor X
  X++;
}


//El bucle se ejecutará 10 veces
y mostrará en la consola el
número cada vuelta

//Una variante es el do/while

//Esta estructura se podría
realizar de forma más simple
mediante "for" (el 3er parámetro
es opcional):

for (X=0;X < 10; X++)
{
   console.log(X);
}
```

* Podemos detener un bucle mediante el comando "break" para evitar bucles infinitos

EJEMPLOS CON ESTRUCTURAS DE CONTROL (PHP)

- Secuenciales:

```
<?php

  echo "Hola";

  echo " ";

  echo "mundo";

?>
//La pantalla mostrará
"Hola mundo" (los espacios en blanco entre líneas se ignoran)
```

NOTA: en PHP el "elseif" se escribe junto, a diferencia de Javascript que se escribe separado

- Selectiva múltiple:

```
<?php

//Ejecutamos unas instrucciones dependiendo del valor de una variable

$X = 3;

if(X > 4) {
   echo "X es mayor que 4";
}
elseif(X == 3) {
   echo "X es igual a 3";
}
else {
   echo "X no es ni mayor que 4 ni es igual a 3";
}

//Otro sistema mediante switch
switch (x)  {
      case 3:
            $n = "X es igual a 3";
            break;
       case 4:
            $n = "X es igual a 4";
            break;
       default:
            $n = "X no es ni 3 ni 4";
}

echo $n
?>
```

- Repetitivas:

```
<?php

//Crearemos un bucle que finalizará cuando se de la condición
$x = 0;
while(X < 10)
{
  echo X;
  //Sumamos 1 al valor X
  X++;
}
//El bucle se ejecutará 10 veces y mostrará en la pantalla "0123456789";

//Esta estructura se podría realizar de forma más simple mediante "for":

for (X=0;X < 10; X++)
{
   echo X;
}

?>
```

EJEMPLOS CON ESTRUCTURAS DE CONTROL (C#)

- Secuenciales:

```
string texto = "hola";
void Start () {
    texto += " mundo";
    Debug.Log(texto);
    }
//La pantalla mostrará "Hola mundo" (los espacios en blanco entre líneas se ignoran)
```

- Selectiva múltiple:

```
//Ejecutamos unas instrucciones dependiendo del valor de una variable

int myNum = 5;

void Start () {
   if(myNum != 5)
   {
         Debug.Log("El número es distinto a 5");
   }
   else if(myNum < 5)
   {
          Debug.Log("El número es menor que 5");
   }
   else
   {
          Debug.Log("Ninguna de las opciones anteriores");
   }

}
```

- Repetitivas:

```
//Crearemos un bucle que finalizará cuando se de la condición
int myNum = 5;
void Start () {
     while(myNum >= 0)
     {
         Debug.Log(myNum);
         myNum--;
     }
 }

//El bucle se ejecutará 5 veces y mostrará en la pantalla "43210";

//Esta estructura se podría realizar de forma más simple mediante "for":

int myNum;
void Start () {
    for(myNum = 5; myNum >= 0; myNum--)
    {
        Debug.Log(myNum);
    }
}
```

NOTA: en C# para Unity, el método Start se ejecuta al lanzarse el script en el juego

MÉTODOS Y FUNCIONES

FUNCIONES

- Una función, también denominada "método" en POO, es un conjunto de instrucciones que permiten procesar las variables para obtener un resultado.
- Son una herramienta básica para organizar ciertos procedimientos que se repiten en un programa, optimizando enormemente los recursos y reduciendo el tiempo de programación.
  - Cuanto más compleja es la función, mejor tienen que estar definidas las variables y las operaciones, y más útil resultará al conjunto.
- Las funciones se declaran con un nombre que las identifique, igual que las variables, pero se identifican por ir acompañadas de unos paréntesis a continuación del nombre
  - Esos paréntesis pueden ir vacíos o incluir las variables que pasamos a la función
  - En algunos lenguajes como c#, lo primero es indicar qué tipo de dato devolverá la función (string, int, float, etc). Si no devolverá nada, se indicará mediante "void"
- Para ejecutar la función, sólo tenemos que escribirlas (con los paréntesis)
  - Lo normal es que ejecuten unas acciones y que retornen un valor (aunque también pueden no hacerlo), el cual se asigna a la variable que ha llamado la función.
  - El valor que devuelve se indica en la función mediante la orden "return"

```
// a la función le llegan
function ejecutarOrden(dato1, dato2 ){
    // declaración variables locales
    var datoLocal;

    // instrucciones
    datoLocal = dato1 + dato2;

    return datoLocal;
}
```

- Para llamara a esta función debo escribir su nombre y poner entre paréntesis las variables que me pide (ene ste caso, dos números)
- El resultado lo puedo asignar a una variable nueva. Ej.-

```
var resultado = ejecutarOrden(5,8);
```

- Recuerda: las variables dato1 y dato2 son locales, no están disponibles fuera de la función

PARÁMETROS DE UNA FUNCIÓN

- Los parámetros básicos de cualquier función son:
  - **Nombre de la función seguido de unos paréntesis**: con el que será llamada desde cualquier parte del código. Ej.-

```
function nombreDeMiFuncion();
```

  - **Variables**: las variables que le pasaremos a la función para que ejecute sus operaciones. Se declaran dentro de los paréntesis y sólo estarán disponibles dentro de la función (variables locales). Ej.-

```
function nombreDeMiFuncion (var1,var2){    }
```

    - Opcionalmente s puede dar el valor de una variable por defecto, y en ese caso no es necesario pasarla al llamar la función: Ej.-

```
function miFuncion(var1,var2="valor por defecto") {   }
```

  - **Resultado**: después de ejecutarse, la función nos devolverá un valor (aunque a veces no es necesario) el cuál es adoptado por la sentencia que ha llamado a la función. Ej:

```
NombreDeMiFuncion (var1,var2) { return "resultado" }
MyVar = NombreDeMiFuncion("primera variable", "segunda variable")
```

- LEn C#, la función se declara escribiendo primero el tipo de dato que va a devolver ("void" si no retornará nada), seguido del nombre de la función con sus paréntesis:

```
string nombreDeMiFuncion (var1,var2){    }
```

EJEMPLO PRÁCTICO DE USO DE UNA FUNCIÓN

- Creamos la función

```
//La llamamos ComprobarEdad y le asignamos dos variables, una con el año actual
y otra con el año de nacimiento
function comprobarEdad (birthYear, currentYear)
{
    var resultado = "";
    //Nos aseguramos de que ambas variables son números para evitar errores
   if(isNaN(birthYear) == false && isNaN(currentYear) == false)
    {
       //Restamos el año actual al año de nacimiento
        var edad = currentYear - birthYear;
        //Comprbamos el resultado
        if(edad >= 18)
        {
            resultado= "Es mayor de edad. Edad: " + edad;
         }
         else
         {
              resultado = "Es menor de edad. Edad: " + edad;
         }
    }
     else
     {
         resultado = "Los datos pasados no son numéricos";
     }

   //Devolvemos el resultado
   return resultado
}
```

- En nuestro programa queremos comprobar si una persona es mayor de edad a partir de su fecha de nacimiento. Como es un cálculo que estamos ejecutando constantemente, en lugar de repetir el código una y otra vez, creamos una función a la que podremos llamar en cualquier momento

- Una vez declarada la función, podremos llamarla siempre que queramos y obtendremos el valor retornado

- Llamamos a la función

```
//Tenemos el año de nacimiento (1976) y el año actual
(2015). Ejecutamos la función
var comprobacion = comprobarEdad(1976,2015);
console.log(comprobacion);

//Nos sacará por consola "Es mayor de edad. Edad: 39"
//NOTA: Esta función no es del todo precisa, ¿sabrías
decir por qué?
```

LIBRERÍAS

- Todo aquello que vaya a ser usado en más de una página de un sitio web, en vez de ser repetido en cada una de ellas se externaliza en un archivo aparte y se vincula a cada página que lo requiera
  - Esto permite no sólo reducir el tamaño de la página, sino que al compartir esos scripts o librerías se optimiza el proceso en caso de que haya que modificarlos, ya que en ese caso se hará solo una vez para todas las páginas.

- En C# usaremos el comando "Using" al comienzo del documento para incluir las librerías. Por defecto, en Unity ya tendremos incorporadas algunas de ellas, pero tendremos que incorporar otras para hacer uso de nuevas funcionalidades.

```
using UnityEngine;
```

- En desarrollo web, mediante la etiqueta <link> podemos enlazar el documento con un recurso externo. Se suele ubicar dentro de las etiquetas <head> de la página. Las más habituales son las librerías CSS y las javascript.

```
<link type="text/css" href="theme.css">
```

- Otras muy habituales son las librerías javascript, que se ubican al final de la página:

```
<link type="text/css" href="theme.css">
```