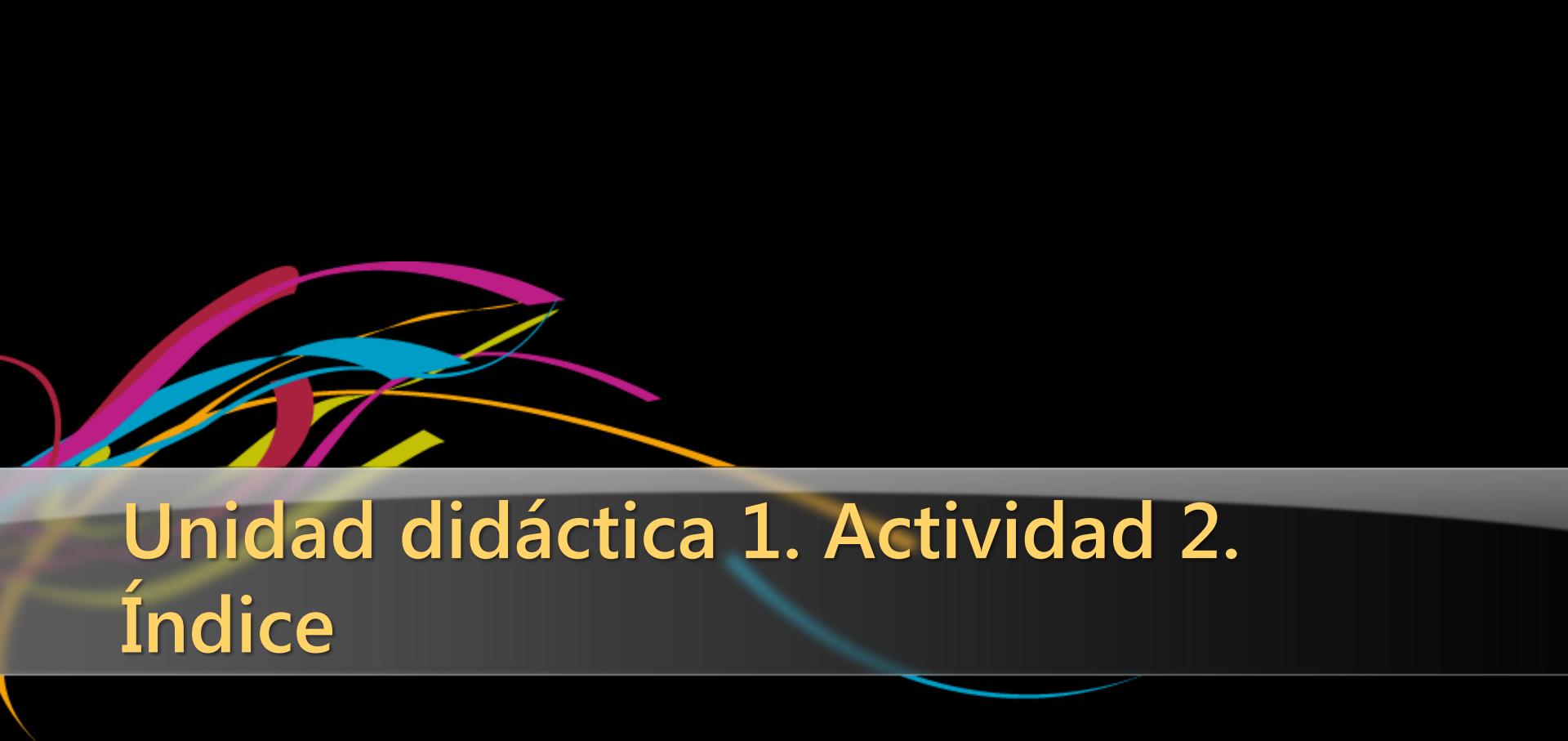




Unidad didáctica 1. Actividad 2.

Lenguajes de programación y herramientas de desarrollo

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

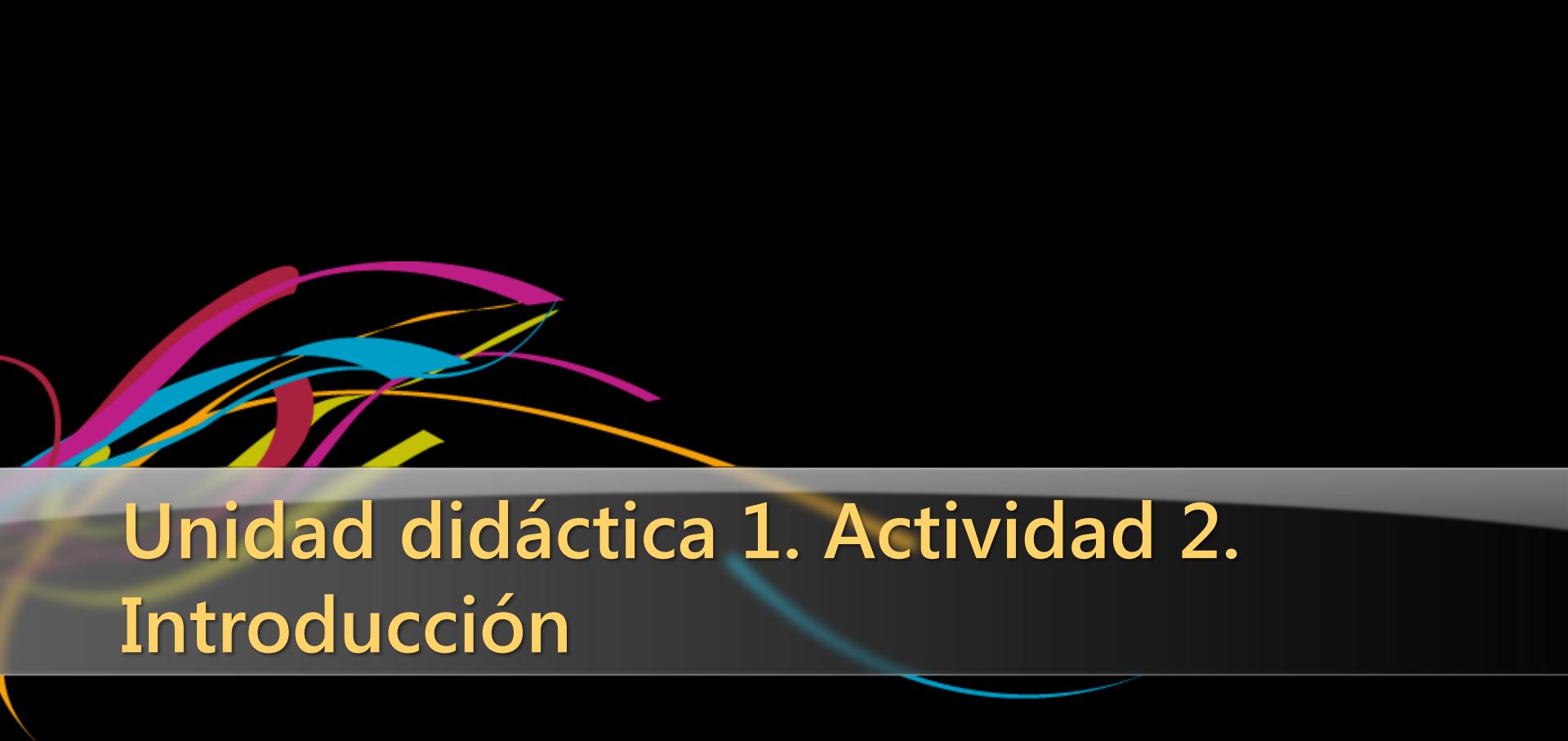


Unidad didáctica 1. Actividad 2. Índice

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

Índice

1. Introducción.
2. Clasificación de los lenguajes informáticos.
3. Clasificación de los lenguajes de programación.
4. Lenguajes más difundidos.
5. Procesos de generación de código.

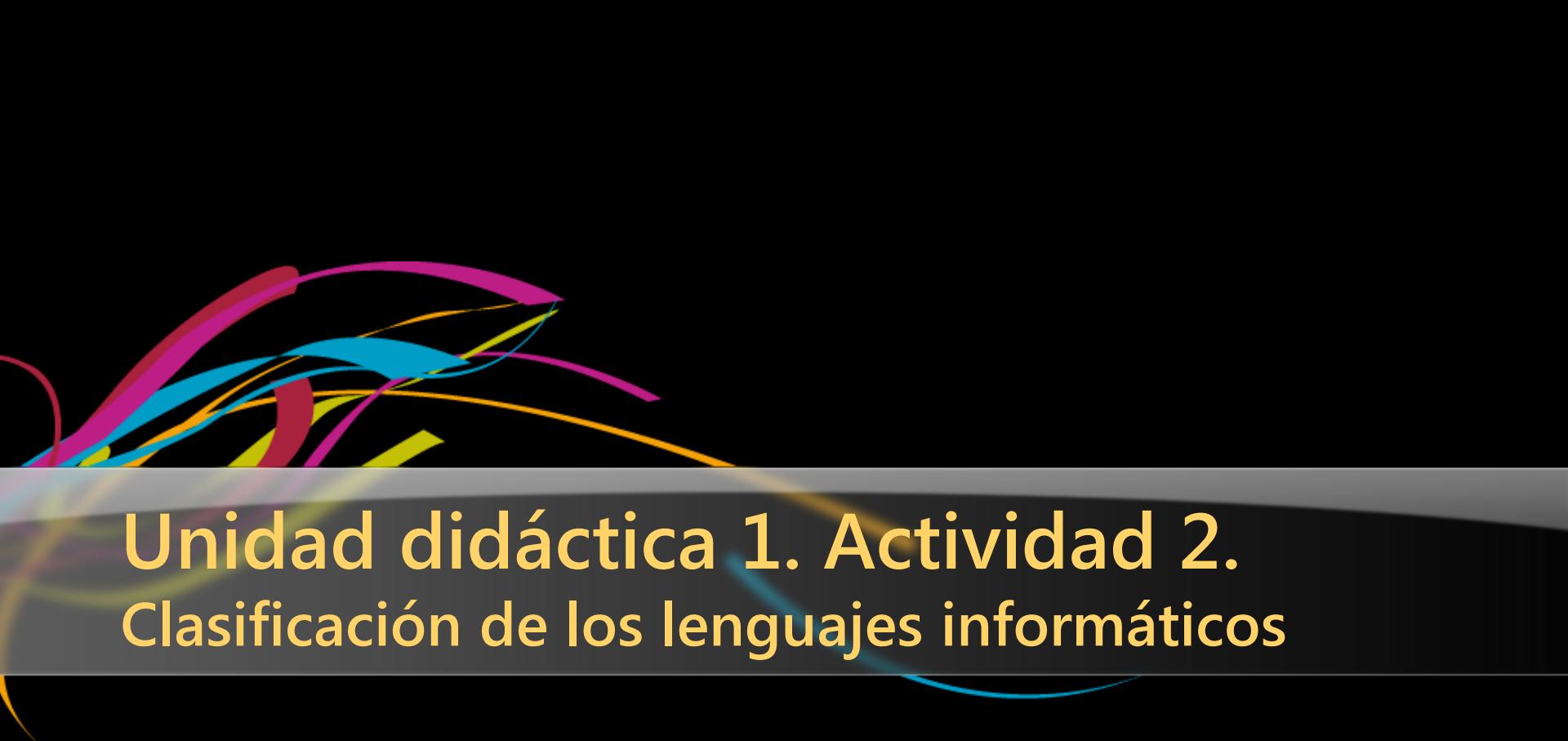


Unidad didáctica 1. Actividad 2. Introducción

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

1.- Introducción

- Los objetivos de la actividad *Lenguajes de programación y herramientas de desarrollo* son los siguientes:
 - Diferenciar lenguajes informáticos, clasificarlos e identificar y caracterizar los lenguajes de programación más populares.
 - Reconocer las características, código generado y herramientas utilizadas en la edición, compilación, enlace y ejecución para lenguajes de programación compiladas, interpretadas, de máquina virtual o de ejecución administrada.



Unidad didáctica 1. Actividad 2. Clasificación de los lenguajes informáticos

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

2.- Clasificación

- **Clasificación de los lenguajes informáticos**

Un lenguaje informático es un lenguaje que permite comunicarse con el ordenador.

- Está formado por un conjunto de símbolos y palabras que siguen unas normas de sintaxis.

- No existe una clasificación adoptada por la mayoría de los autores -> hay grandes diferencias.

- Una posible clasificación:

- | | |
|--|---|
| <ul style="list-style-type: none">■ Lenguajes de marcas.■ Especificación. | <ul style="list-style-type: none">■ Consulta.■ Transformación.■ Programación. |
|--|---|

2.- Clasificación

Lenguajes de marcas

- Permiten colocar distintivos o señales en el texto que serán interpretadas por aplicaciones o procesos.
- Ejemplo:
 - XML (*eXtensible Markup Language*): metalenguaje extensible pensada para la transmisión estructurada y que puede ser validada.
 - HTML (*Hiper Text Markup Language*) o XHTML (*eXtensible Hiper Text Markup Language*) =XML+HTML: sirven para publicar hipertexto en la WWW.

2.- Clasificación

Lenguajes de marcas

- Ejemplo código XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Exemplo básico de XML
-->
<alumnos>
    <alumno>
        <nome>Pepe</nome>
        <apelidos>Ruiz Arias</apelidos>
    </alumno>
    <alumno>
        <nome>María Dolores</nome>
        <apelidos>González Paz</apelidos>
    </alumno>
</alumnos>
```

2.- Clasificación

Lenguajes de marcas

- Ejemplo código XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Exemplo moi básico de XHTML con estilo externo -->
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Exemplo básico de xhtml</title>
        <meta content="text/html; charset=utf-8"
              http-equiv="Content-Type" />
        <link rel="stylesheet" href="exemplo_css.css"
              type="text/css" />
    </head>
    <body>
        <h1>Cabeceira principal</h1>
        <p class="principal">Este párrafo contén texto e un enlace a
            <a href="http://www.realacademiagalega.org/">Real Academia Galega
            </a>.
        </p>
    </body>
</html>
```

2.- Clasificación

Lenguajes de especificación

- Describen algo de forma precisa. Por ejemplo CSS (*Cascading Style Sheets*) es un lenguaje formal que especifica la presentación o estilo de un documento HTML, XML o XHTML.
- Ejemplo de código CSS aplicable al ejemplo XHTML anterior:

```
body{  
    font-family: "MS Sans Serif", Geneva, sans-serif;  
    font-size: 15px;  
    color: Black;  
    border: black 2px double;  
    padding: 40px;  
    margin: 20px;}  
  
h1 {  
    font: 40px "Times New Roman", Times, serif;  
    font-weight: bolder;  
    word-spacing: 25px;}  
  
p.principal{  
    text-align: center;  
    font: 10px "MS Serif", "New York", serif;}
```

2.- Clasificación

Lenguajes de consulta

- Permiten sacar o manipular información de un grupo de información.
- Ejemplo: lenguaje de consultas SQL (*Standard Query Language*) permite buscar y manipular información en bases de datos relacionales y el lenguaje XQuery permite buscar información en bases de datos XML nativas:

```
SELECT BusinessEntityID, JobTitle, HireDate  
FROM HumanResources.Employee  
ORDER BY DATEPART(year, HireDate);
```

2.- Clasificación

Lenguajes de consulta

- Ejemplo de expresión FLOWR (*for, let, order by, where, return*) en XQuery.

```
for $libro in doc("libros.xml")/bib/libro
let $editorial := $libro/editorial
where $editorial="MCGRAW/HILL" or contains($editorial, "Toxosoutos")
return $libro
```

2.- Clasificación

Lenguajes de transformación

- Actúan sobre una información inicial para obtener otra nueva.
- Ejemplo: el lenguaje XSLT (*eXtensible Stylesheet Language Transformations*) permite describir las transformaciones que se van a realizar sobre un documento XML para obtener otro archivo.

2.- Clasificación

Lenguajes de transformación

- Ejemplo de transformación XSL sencilla cuando se actúa sobre el ejemplo XML anterior para obtener una página HTML con una lista de alumnos:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo sinxelo de transformación XSL --&gt;
&lt;xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"&gt;
    &lt;xsl:output method="html"/&gt;
    &lt;xsl:template match="alumnos"&gt;
        &lt;html&gt;
            &lt;head&gt;
                &lt;title&gt;fundamentos.xsl&lt;/title&gt;
            &lt;/head&gt;
            &lt;body&gt;
                &lt;h1&gt;Alumnos&lt;/h1&gt;
                &lt;ul&gt;
                    &lt;xsl:for-each select="alumno"&gt;
                        &lt;li&gt;
                            &lt;xsl:value-of select="apelidos"/&gt;,
                            &lt;xsl:value-of select="nome"/&gt;
                        &lt;/li&gt;
                    &lt;/xsl:for-each&gt;
                &lt;/ul&gt;
            &lt;/body&gt;
        &lt;/html&gt;
    &lt;/xsl:template&gt;
&lt;/xsl:stylesheet&gt;</pre>
```

2.- Clasificación

Lenguajes de programación

- Permiten comunicarse con los dispositivos hardware y así poder realizar un determinado proceso.
- Para eso pueden manejar estructuras de datos almacenadas en memoria interna o externa, y utilizar estructuras de control.
- Disponen de un léxico, y tienen que cumplir reglas sintácticas y semánticas.
 - Léxico: conjunto de símbolos que se pueden usar y pueden ser: identificadores (nombres variables, tipos de datos, nombres de métodos, etc.), constantes, variables, operadores, instrucciones y comentarios.

2.- Clasificación

Lenguajes de programación

- Las **reglas de sintaxis**: especifican la secuencia de símbolos que forman una frase bien escrita en ese lenguaje, es decir, para que no tengan faltas de ortografía. Es decir, dado un texto, las reglas de sintaxis indican si es válido en un lenguaje concreto.
- Las **reglas de semántica**: definen cómo tienen que ser las construcciones sintácticas y las expresiones y tipos de datos utilizadas. Es decir, dado un texto sintácticamente correcto, indica si tiene significado. Los compiladores no los detectan porque la sintaxis es correcta pero el programa no hace lo que debe.

2.- Clasificación

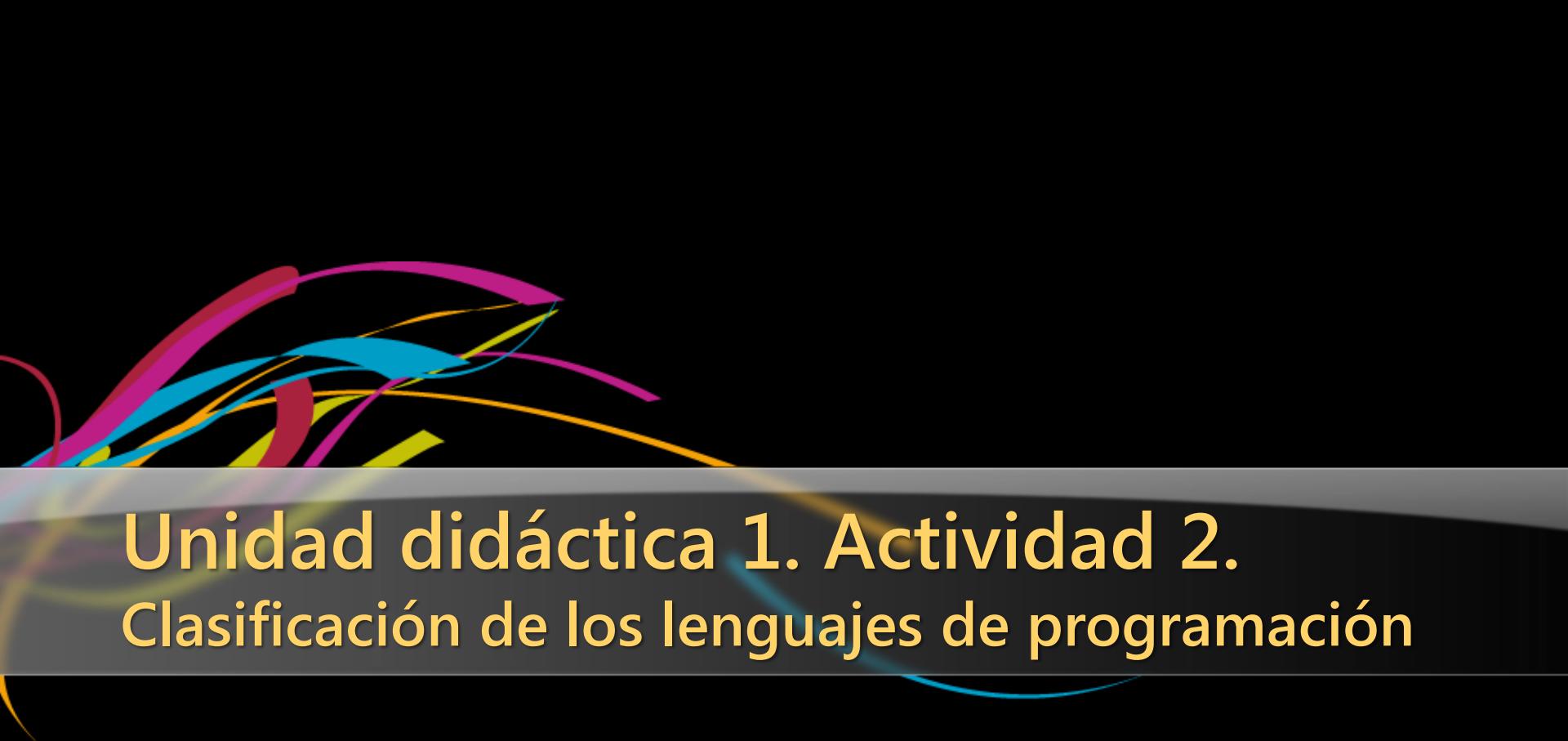
Lenguajes de programación

- Ejemplo de código Java

```
package parimpar;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        short n;
        Scanner teclado = new Scanner(System.in);
        System.out.printf("Teclee o número enteiro entre %d e %d:",
                          Short.MIN_VALUE, Short.MAX_VALUE);
        n = teclado.nextShort();
        if(n%2==0){
            System.out.printf("%d é par\n",n);
        }
        else{
            System.out.printf("%d é impar\n",n);
        }
    }
}
```



Unidad didáctica 1. Actividad 2.

Clasificación de los lenguajes de programación

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

3.- Clasificación

- Los lenguajes de programación pueden clasificarse según los siguientes criterios:
 - Distancia al hardware.
 - Por generaciones.
 - Por el paradigma de programación.
 - Por la forma de traducirse a la lenguaje máquina y de ejecutarse.
 - Según la arquitectura cliente-servidor.

3.- Clasificación Según su distancia al hardware

- Pueden clasificarse en lenguaje de bajo y alto nivel.

Lenguaje de bajo nivel

- Está basado directamente en los circuitos electrónicos de la máquina por lo que un programa escrito para una máquina no podrá ser utilizada en otra diferente.
- Pueden ser:
 - Lenguajes máquina.
 - Lenguaje ensamblador.

3.- Clasificación Según su distancia al hardware

- El lenguaje máquina es código binario (ceros y unos) y actúa directamente sobre el hardware.
- Es el único lenguaje que no necesita traducción ya que el procesador reconoce las instrucciones directamente.
- La codificación en lenguaje ensamblador es mnemotécnica, es decir, utiliza etiquetas para describir ciertas operaciones. No es más que una traducción de cadenas de 0 y 1 casi imposibles de recordar por nombres que permiten identificar las instrucciones.

-u 100 1a	
OCFD :0100	BA0B01
OCFD :0103	B409
OCFD :0105	CD21
OCFD :0107	B400
OCFD :0109	CD21

MOV	DX,010B
MOV	AH,09
INT	21
MOV	AH,00
INT	21

3.- Clasificación

Según su distancia al hardware

- Es necesario traducir el código ensamblador al lenguaje máquina para que el procesador reconozca las instrucciones.
- Ejemplo: suma de 3 + 5

<u>Ensamblador</u>	<u>Código máquina (Hexadecimal)</u>
mov ax, 0003	B8 03 00
add ax, 0005	05 05 00

- En el ejemplo anterior el código máquina se representa en hexadecimal por tener una representación más corta.

3.- Clasificación

Según su distancia al hardware

- En la página godbolt.org tenemos una herramienta donde se traduce online un código fuente escrito en C a ensamblador:

The screenshot shows the Compiler Explorer interface. On the left, the C++ source code is displayed:

```
// Type your code here, or
// load an example.
#include <stdio.h>

int square(int num) {
    printf("hola");
    return 0;
}
```

On the right, the generated assembly code for x86-64 gcc 9.2 is shown:

```
.LC0:
    .string "hola"
square(int):
    push    rbp
    mov     rbp, rsp
    sub     rsp, 16
    mov     DWORD PTR [rbp-4], edi
    mov     edi, OFFSET FLAT:.LC0
    mov     eax, 0
    call    printf
    mov     eax, 0
    leave
    ret
```

Compiler options visible include: 11010, ./a.out, .LX0:, lib.f, .text, //, \s+, Intel, Demangle.

3.- Clasificación Según su distancia al hardware

Lenguajes de alto nivel

- Tienden a acercarse al lenguaje humano y se separan del conocimiento interno de la máquina, por eso necesitan traducirse a lenguaje máquina.
- Esta traducción hace que la ejecución sea más lenta que en los lenguajes de bajo nivel pero al no depender del procesador se pueden ejecutar en diferentes ordenadores.
- Ejemplo: Suma de 3+5
$$X = 3 + 5$$

3.- Clasificación

Según su generación

- Pueden clasificarse en 5 generaciones:
 - 1^a generación (1GL): formada por los lenguajes de programación utilizados en los primeros ordenadores: lenguaje máquina y ensamblador.
 - 2^a generación (2GL): formada por el lenguaje macroensamblador que es el lenguaje ensamblador combinado con instrucciones de control y de manejo de datos más complejos. Disponen de macroinstrucciones (macros) como por ejemplo, instrucciones para transferir un bloque de memoria principal a disco.

3.- Clasificación Según su generación

- 2^a generación (2GL): tienen definidas unas instrucciones para realizar operaciones sencillas con datos simples o posiciones de memoria.

Cada modelo de ordenador tiene un lenguaje ensamblador propio distinto de los demás, por lo cual el programa solo puede utilizarse en la máquina para la cual se programó.

Se siguen utilizando para programar los núcleos (*kernel*) de los sistemas operativos y los controladores de algunos dispositivos (*device drivers*).

3.- Clasificación Según su generación

- **3^a generación (3GL)**: formada por la mayor parte de los lenguajes de alto nivel actuales. El código es independiente de la máquina y el lenguaje de programación es parecido al lenguaje humano.
- **4^a generación (4GL)**: formada por lenguajes y entornos diseñados para una tarea o propósito muy específico como acceso a bases de datos, generación de informes, generación de interfaces de usuario, etc. Por ejemplo: SQL, Informix 4GL, Progress 4GL.

3.- Clasificación Según su generación

- 5^a generación (5GL): lenguajes naturales, formada por los lenguajes en los que el programador establece el problema a resolver y las condiciones a cumplir. Mecanismos que permitan la comunicación entre personas y máquinas por medio de lenguajes naturales

Se usa en inteligencia artificial, sistemas basados en conocimiento, sistemas expertos, mecanismos de inferencia o procesamiento de lenguaje natural.
Ejemplos: Prolog, Smalltalk y Lisp.

3.- Clasificación Según su generación

- Ejemplo 1 de código en Prolog:

```
%%  
%% declaraciones  
%%  
padrede('juan', 'maria'). % juan es padre de maria  
padrede('pablo', 'juan'). %pablo es padre de juan  
padrede('pablo', 'marcela'). % pablo es padre de marcela  
padrede('carlos', 'debora'). % carlos es padre de debora  
  
% A es hijo de B si B es padre de A  
hijode(A,B) :- padrede(B,A).  
% A es abuelo de B si A es padre de C y C es padre B  
abuelode(A,B) :-  
    padrede(A,C),  
    padrede(C,B).  
% A y B son hermanos si el padre de A es tambien el padre de B y si A y B no son lo mismo  
hermanode(A,B) :-  
    padrede(C,A) ,  
    padrede(C,B),  
    A \== B.  
  
% A y B son familiares si A es parente de B o A es hijo de B o A es hermano de B  
familiarde(A,B) :-  
    padrede(A,B).  
familiarde(A,B) :-  
    hijode(A,B).  
familiarde(A,B) :-  
    hermanode(A,B).
```

3.- Clasificación Según su generación

- Ejemplo 1 de código en Prolog:

```
%%
%% consultas
%%
% juan es hermano de marcela?
?- hermanode('juan', 'marcela').
yes

% carlos es hermano de juan?
?- hermanode('carlos', 'juan').
no

% pablo es abuelo de maria?
?- abuelode('pablo', 'maria').
yes

% maria es abuela de pablo?
?- abuelode('maria', 'pablo').
no
```

3.- Clasificación

Según su generación

- Ejemplo 2 de código en Prolog:

```
enfermo_de (manuel,gripe) .  
tiene_sintoma (alicia,cansancio) .  
sintoma_de (fiebre,gripe) .  
sintoma_de (tos,gripe) .  
sintoma_de (cansancio,anemia) .  
elimina(vitaminas,cansancio) .  
elimina(aspirinas,fiebre) .  
elimina(jarabe,tos) .  
recetar_a (X,Y) :-enfermo_de (Y,A),alivia(X,A) .  
alivia(X,Y) :-elimina (X,A),sintoma_de (A,Y) .  
  
enfermo_de (X,Y) :-tiene_sintoma (X,Z),sintoma_de (Z,Y) .
```

3.- Clasificación

Según su paradigma de programación

- Un paradigma de programación es una metodología o filosofía de programación a seguir con un núcleo central incuestionable, es decir, los lenguajes que utilizan el mismo paradigma de programación utilizarán los mismos conceptos básicos para programar.
- Pueden clasificarse en 2 grandes grupos:
 - El grupo que sigue el **paradigma imperativo**.
 - El grupo que sigue el **paradigma declarativo**.

3.- Clasificación

Según su paradigma de programación

Paradigma imperativo

- El código está formado por una serie de pasos o instrucciones para realizar una tarea organizando o cambiando valores en memoria. El programador indica cómo obtener el resultado deseado.
- Las instrucciones se ejecutan de forma secuencial. Hasta que no se ejecuta una no se pasa a ejecutar la siguiente.
- Ejemplos: Java, C, C++, PHP, Javascript y Visual Basic.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- Se distingue entre los que siguen la metodología estructurada y los que siguen la metodología orienta a objetos.
- A finales de los 60 nació la metodología estructurada. El teorema de “programa estructurado”, propuesto por Böhm-Jacopini, demuestra que todo programa puede escribirse utilizando solo las siguientes estructuras:
 - Secuencial.
 - Alternativa (basada en una decisión).
 - Repetitiva (bucles).

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- La metodología estructurada evolucionó añadiendo el módulo como componente básico dando lugar a la programación estructurada y modular.
- Los programas estaban formados por módulos o procesos por un lado y datos por el otro.
- Los módulos necesitaban de unos datos de entrada y obtenían otros de salida que a su vez podían ser utilizados por otros módulos. Ejemplo: lenguaje C.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- No debemos medir la complejidad de una aplicación basándonos en el número de líneas de código.
- Hay otros muchos factores que influyen decisivamente como, por ejemplo, la complejidad del algoritmo que se codifica.
- Bill Gates dijo: "Medir la complejidad del software por líneas de código es como medir la complejidad de un avión por su peso".
- Teniendo esto en cuenta, haremos una estimación de la complejidad del software basándonos solo en el número de líneas para ilustrar la importancia de dividir el código en módulos.

3.- Clasificación

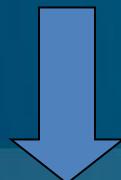
Según su paradigma de programación. Paradigma imperativo

- Los códigos monolíticos presentan el siguiente problema:

Complejidad de un código: $k n^2$

Si $k = 1$

Programa con 10 líneas → Complejidad = 100



Aumenta 10 veces



Programa con 100 líneas



Complejidad = 100



Aumenta 100 veces

Programa con 1000 líneas



Complejidad 10.000

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- Los códigos monolíticos presentan el siguiente problema:

Código con complejidad $k n^2$

Dividimos el código en dos partes y calculamos de nuevo su complejidad.

Complejidad total: $k (n/2)^2 + k (n/2)^2 = \frac{1}{2} k n^2$



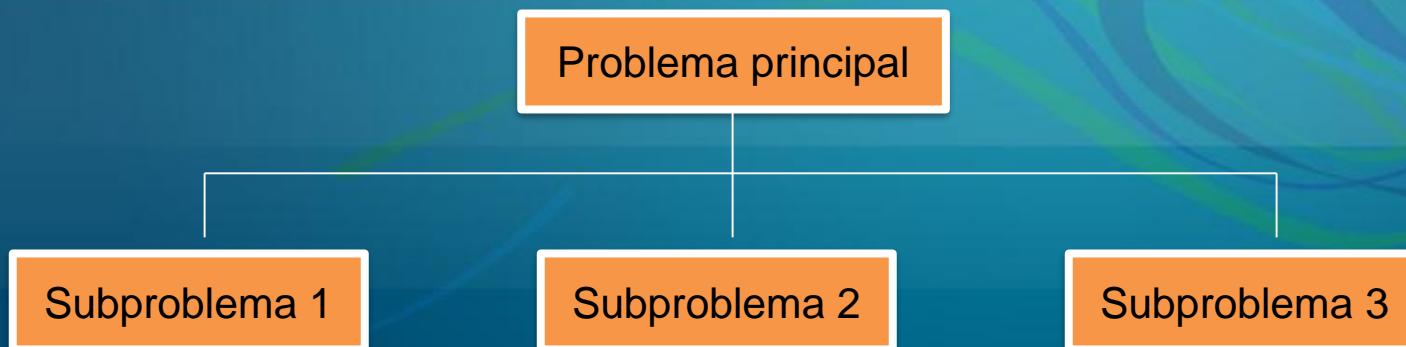
Al dividir el programa en dos, la complejidad se redujo a la mitad

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

Uno de los métodos fundamentales para resolver un problema es dividirlo en problemas más pequeños. Estos problemas pueden a su vez ser divididos repetidamente en problemas más pequeños.

El método de diseño se denomina diseño descendente (Top-Down) debido a que se comienza en la parte superior con un problema general y se diseñan soluciones específicas a sus subproblemas.



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

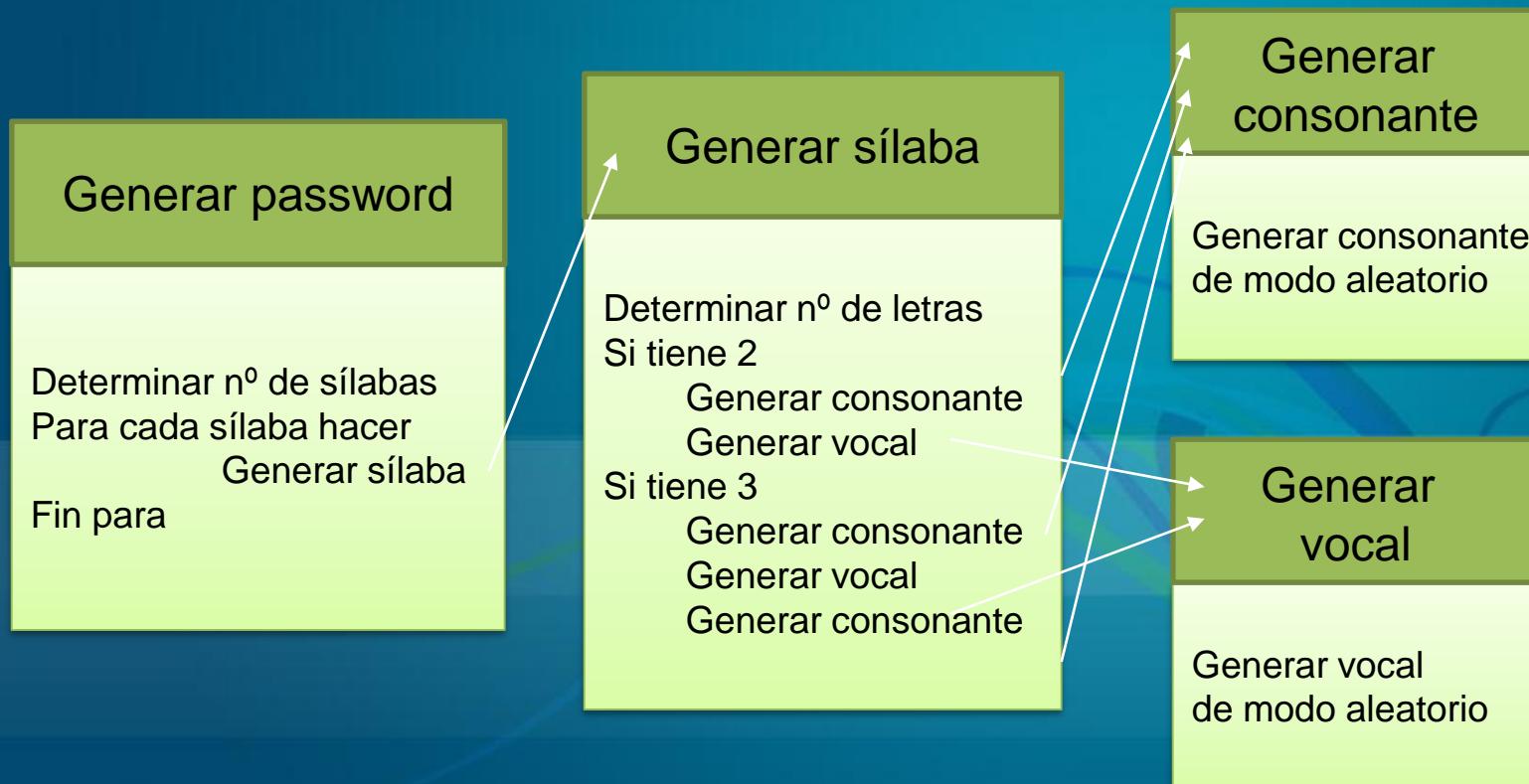
Dicho popular:

"El secreto para progresar es empezar. El secreto para empezar es dividir las tareas complejas y abrumadoras en pequeñas tareas y empezar por la primera."

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

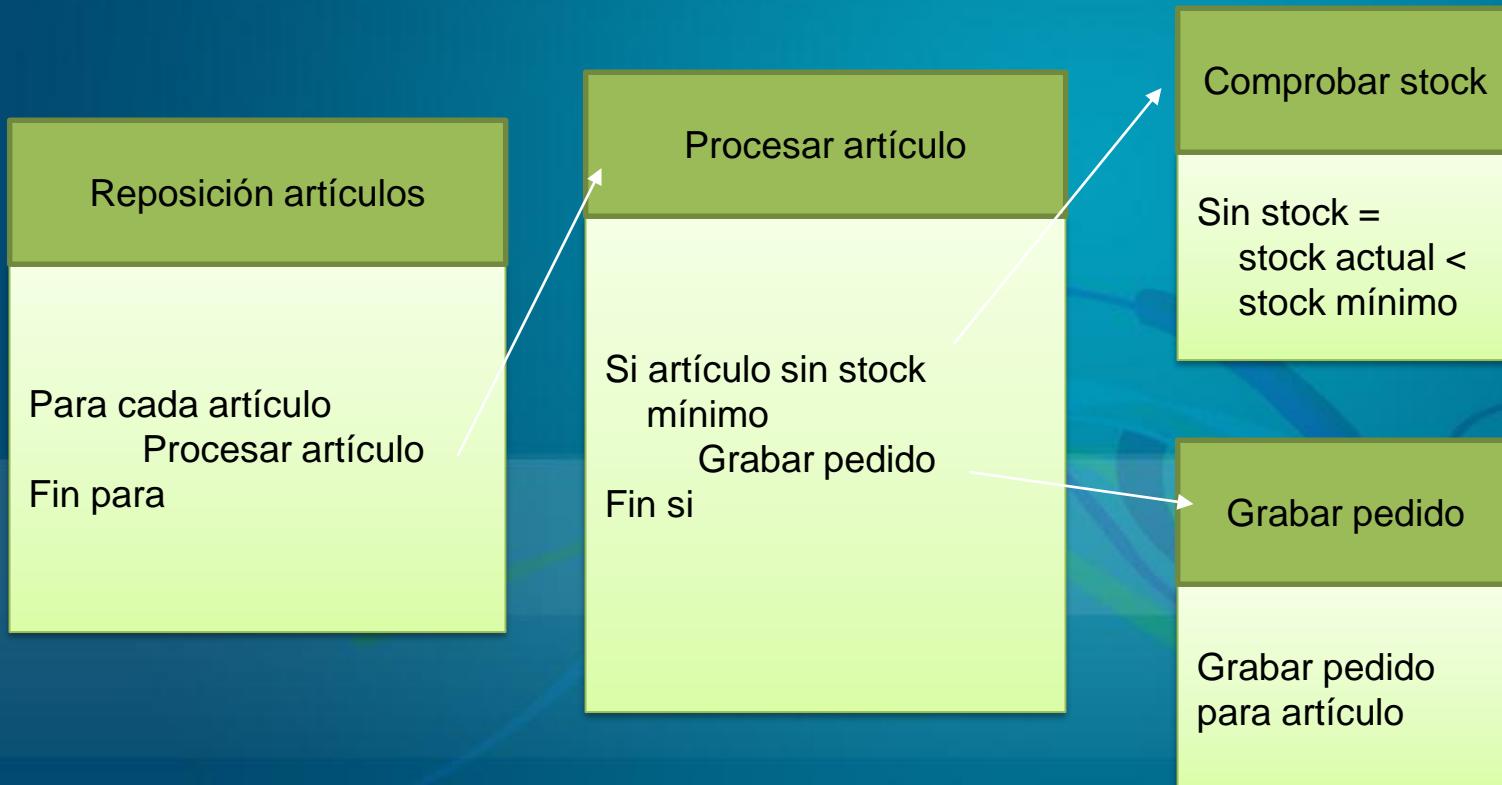
Ejemplo: generar un password aleatorio que sea pronunciable.



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

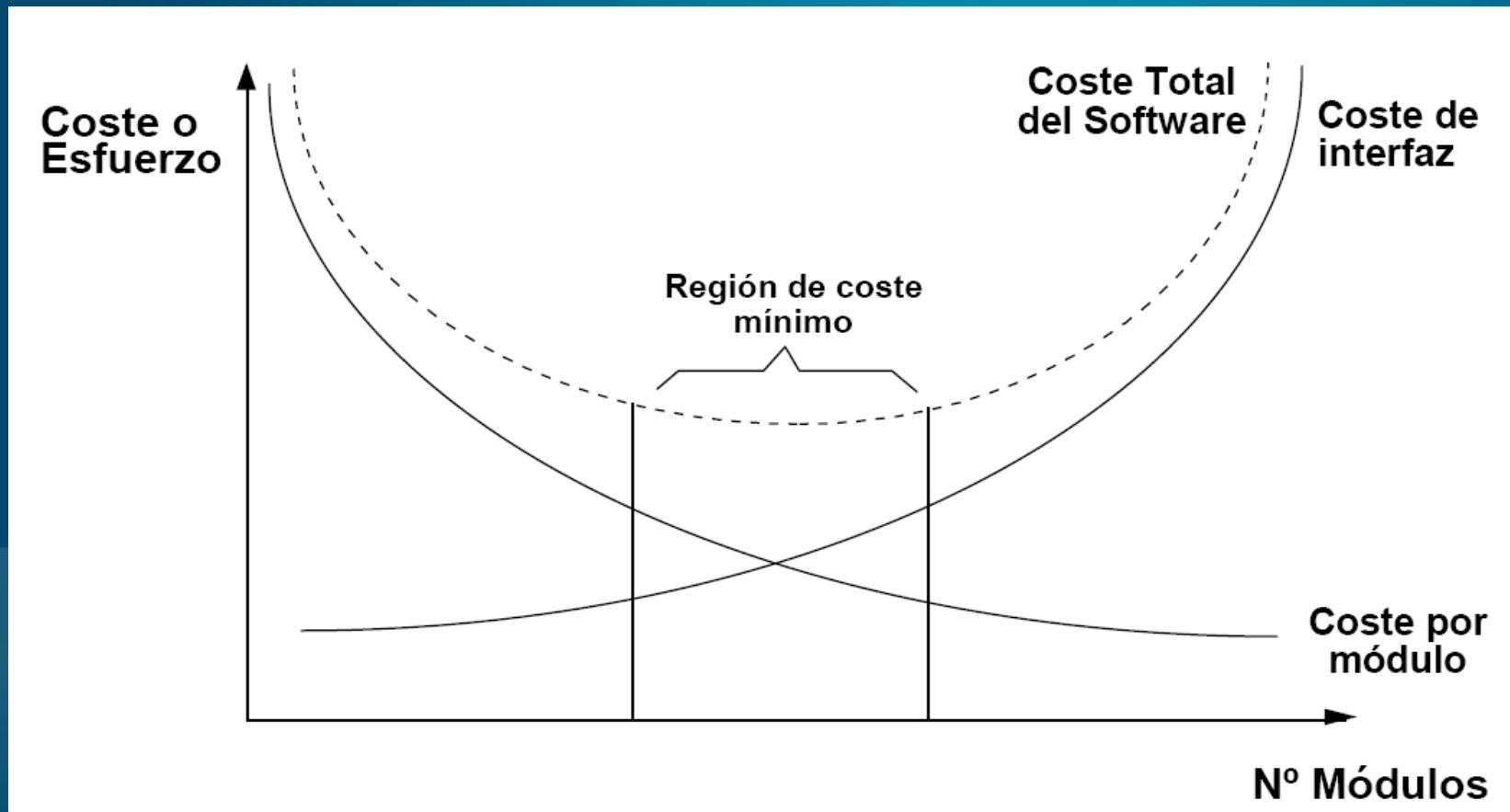
Ejemplo: reposición stock de almacén



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

¿Hasta cuándo conviene seguir dividiendo un módulo?



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- ¿En qué nos debemos basar para realizar la división de un código en distintos módulos?
- Nos basamos en métricas que miden la calidad estructural.
- Cohesión:
 - Indica la relación que existe entre los elementos del mismo módulo.
 - Objetivo: los elementos de un módulo deben tener la máxima relación entre sí.
- Acoplamiento:
 - Es el grado de interdependencia de los módulos.
 - Objetivo: lograr módulos tan independientes como sea posible.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

Por lo tanto:

- Intentaremos que la cohesión sea lo más alta posible.
- Intentaremos que el acoplamiento sea lo más bajo posible.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- En los años 80 apareció la metodología orientada a objetos que considera el objeto como elemento básico.
- Se popularizó a principios de los 90.
- Ejemplos: Java, C++.
- Cada objeto sigue el patrón especificado en una clase. Una clase es un *molde* a partir del que crear las clases.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- Los programas contienen objetos que se relacionan o colaboran con otros objetos de su misma clase o de otras clases.
- La clase está compuesta de **atributos** (datos) y **métodos** (procesos) y pueden tener las siguientes propiedades:

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

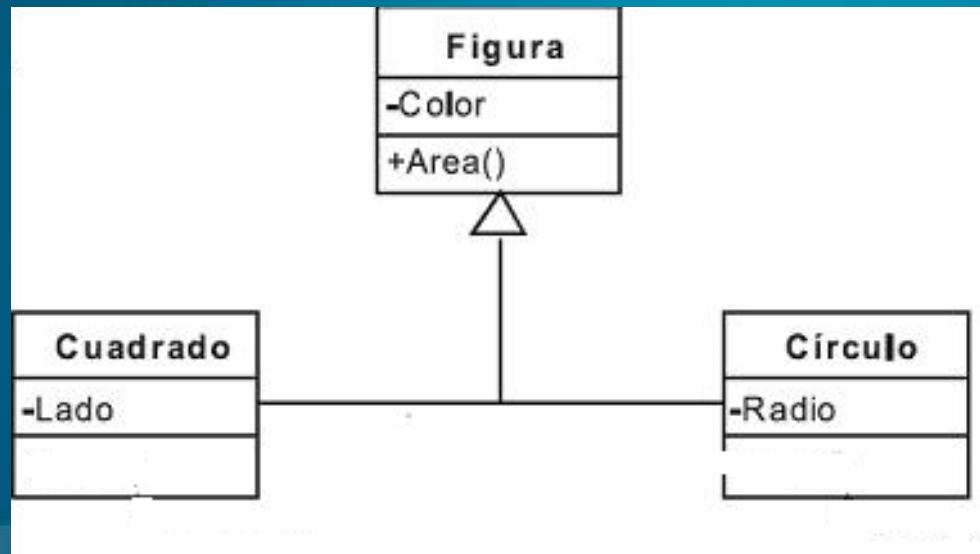
- Herencia: pueden declararse clases hijas que heredan las propiedades y métodos de la clase padre.

Los métodos heredados pueden sobrecargarse, es decir, dentro de la misma clase puede aparecer definido con distinto comportamiento el mismo método con diferente número de parámetros y tipo de los mismos.

3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

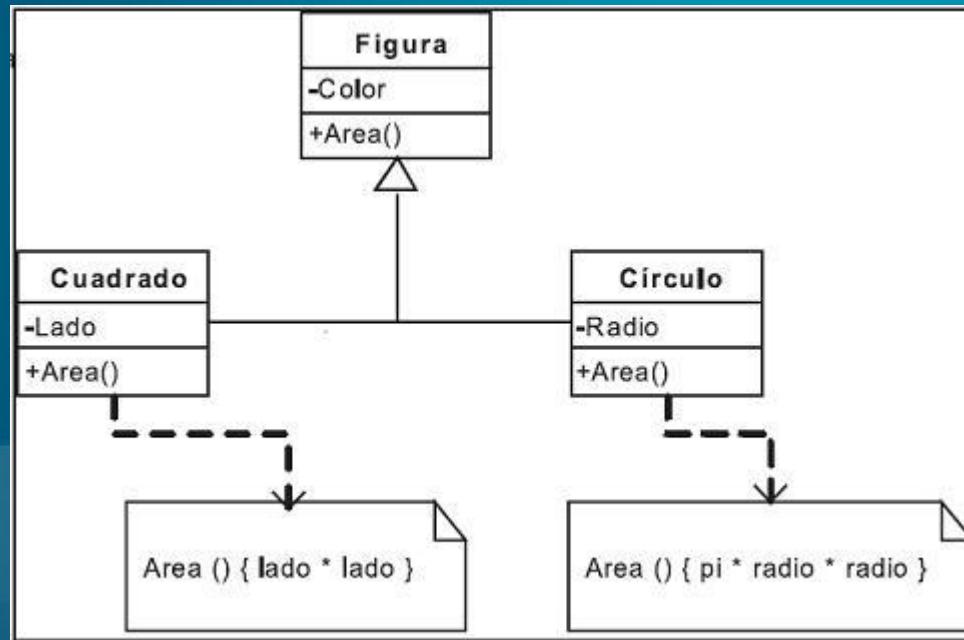
- Herencia



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- **Polimorfismo:** característica que permite que un método se comporte de diferente manera dependiendo del objeto sobre el que se aplica.



3.- Clasificación

Según su paradigma de programación. Paradigma imperativo

- **Encapsulamiento:** Permite ocultar detalles internos de una clase. Las clases encapsulan datos y comportamiento.

3.- Clasificación

Según su paradigma de programación. Paradigma declarativo

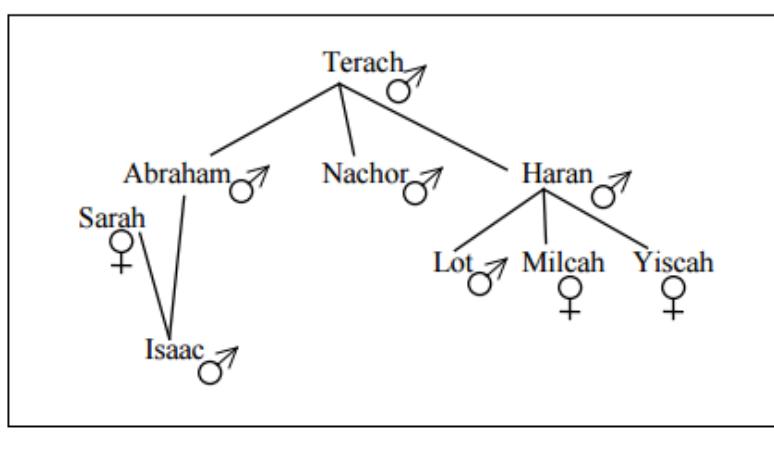
- El código indica qué es lo que se quiere obtener pero no cómo se tiene que obtener.
- Conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.
- El programador tiene que pensar en los resultados que precisa pero no en la secuencia de órdenes para que se lleve a cabo. Ejemplos: Lisp, Prolog o SQL.

3.- Clasificación

Según su paradigma de programación. Paradigma declarativo

- Ejemplo código Prolog.

Conjunto de hechos que constituyen el programa.



Programa 1:

- | | |
|-------------------------------|-------------------------|
| 1. es_padre(terach, abraham). | 9. es_hombre(terach). |
| 2. es_padre(terach, nachor). | 10. es_hombre(abraham). |
| 3. es_padre(terach, haran). | 11. es_hombre(nachor). |
| 4. es_padre(abraham, isaac). | 12. es_hombre(haran). |
| 5. es_padre(haran, lot). | 13. es_hombre(isaac). |
| 6. es_padre(haran, milcah). | 14. es_hombre.lot). |
| 7. es_padre(haran, yiscah). | 15. es_mujer(sarah). |
| 8. es_madre(sarah, isaac). | 16. es_mujer(milcah). |
| | 17. es_mujer(yiscah). |

3.- Clasificación

Según su paradigma de programación. Paradigma declarativo

- Ejemplo código Prolog – Reglas
 - Una regla que define la relación «ser hijo» a partir de las relaciones dadas podría ser:

`es_hijo(X,Y) :- es_padre(Y,X), es_hombre(x)`

Para cualquier X e Y, X es hijo de Y si Y es padre de X y X es hombre.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse

- Se llama código fuente al código escrito en un lenguaje de programación simbólica mediante una herramienta de edición.
- Este código se guarda en un archivo conocido como archivo fuente y tendrá que traducirse a lenguaje máquina para poder ejecutarse.
- La traducción puede hacerse mediante compiladores (traductores que generan código máquina a partir de código fuente) y/o intérpretes (ejecutores paso a paso del código fuente, no se lleva a cabo una traducción en la preejecución).

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse

- Pueden considerarse 3 grupos de lenguajes de programación teniendo en cuenta la forma de traducirse a lenguaje máquina y de ejecutarse:
 - Lenguajes compilados.
 - Interpretados.
 - Máquina virtual o ejecución administrada.
- Algunos lenguajes como Prolog pueden ser considerados como compilados o interpretados ya que dispone de compiladores y de intérpretes.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.

Lenguajes compilados

- Disponen de un compilador o programa que traduce el código fuente a código máquina guardando el resultado en un archivo ejecutable.
- Con ese archivo se puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso por lo que el tiempo de espera entre ejecución y ejecución es ínfimo.
- En tiempo de ejecución puede lanzarse el archivo ejecutable en la plataforma para la que sirve el compilador.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.

Lenguajes compilados

- Características:
 - Existe un archivo ejecutable para la máquina real.
 - El proceso de traducción se hace en tiempo de compilación y una sola vez.
 - La ejecución es muy rápida ya que el ejecutable está en lenguaje máquina.
 - El ejecutable sólo funciona para la plataforma para la que fue creado.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados

- El usuario que ejecuta no tiene que conocer el código fuente, solo tiene el código ejecutable que no es manipulable fácilmente para obtener el código fuente, y como consecuencia el programador tiene el código fuente más protegido.
- El ejecutable no se generará si existen errores léxicos, sintácticos o semánticos.
- Interrumpir la ejecución puede ser difícil para el sistema operativo y puede afectar a la plataforma.

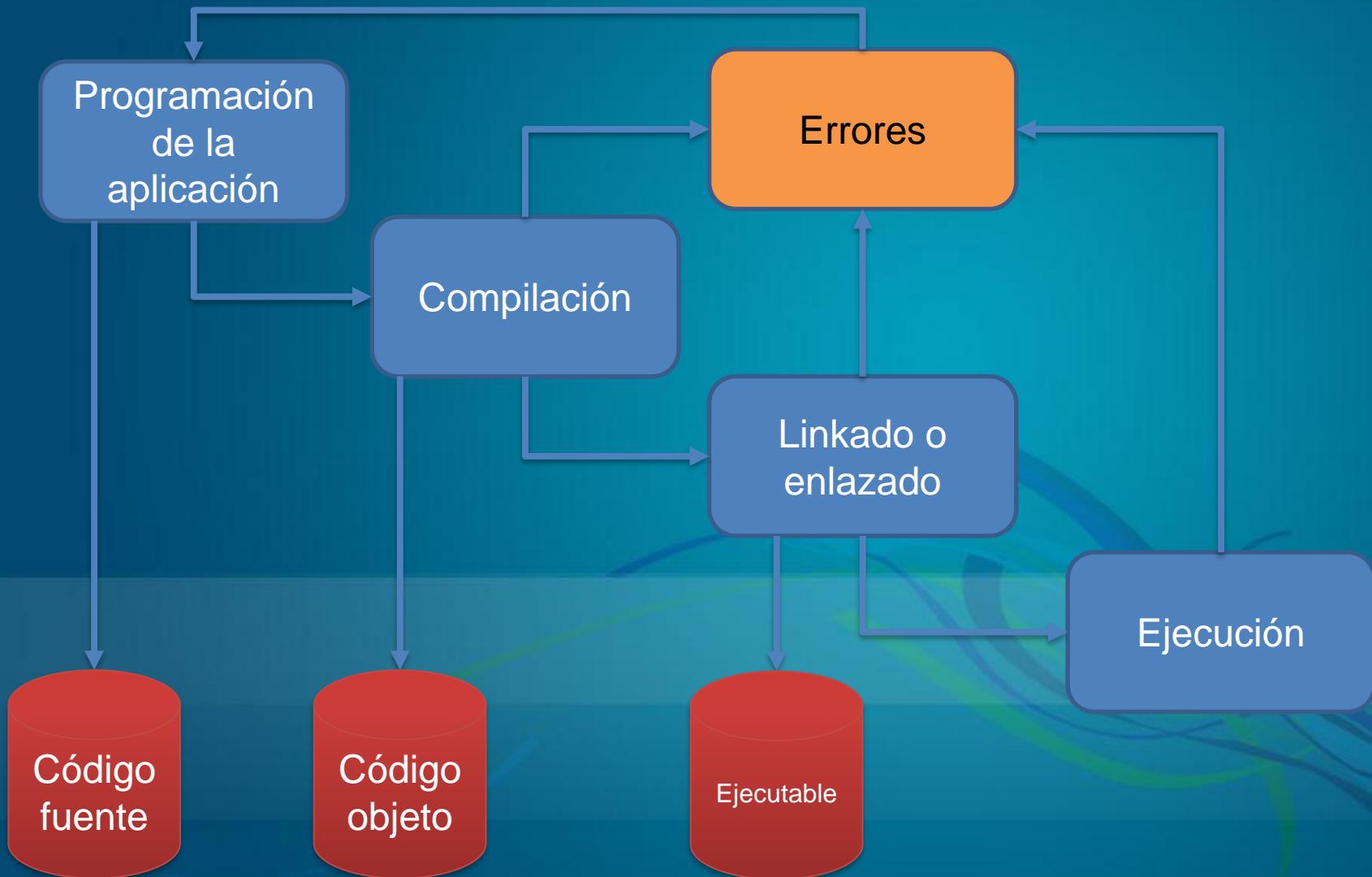
3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados

- La modificación del código fuente implica volver a generar el archivo ejecutable.
- Se necesita un programa denominado linkador que identifique las referencias a nombres externos en el programa objeto y localiza el código correspondiente en las bibliotecas disponibles para obtener el programa ejecutable.

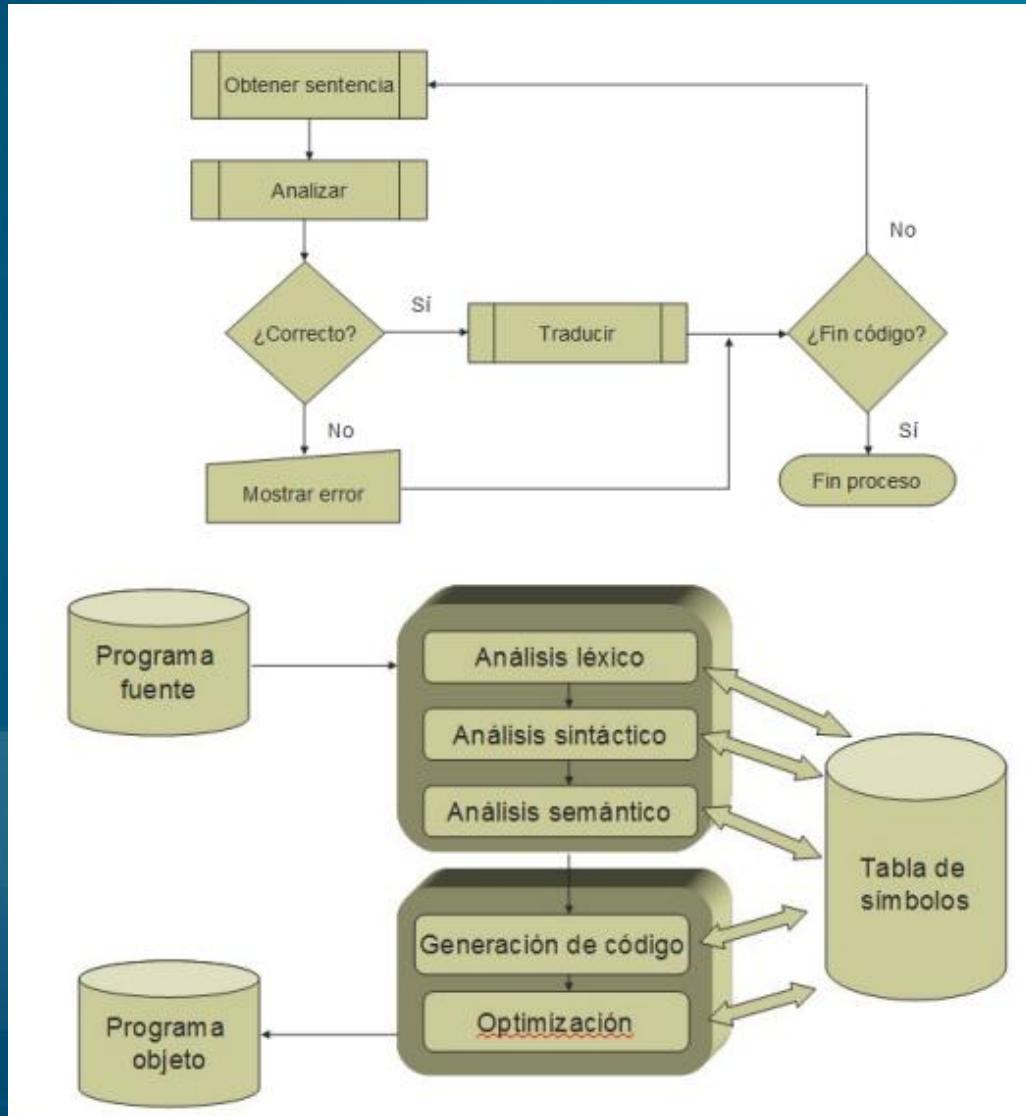
3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados

Lenguajes compilados – Ejemplos

- ADA.
- C.
- C++.
- Objective C.
- Delphi.
- ML.
- Pascal.
- Cobol.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes compilados

Los vemos más en software de escritorio ya que requieren de mayores recursos y de acceso a archivos determinados.

También por el peso mayor que estos suelen tener en sus archivos ejecutables.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.

Lenguajes interpretados

- Ejecutan las instrucciones directamente, sin que se genere código objeto.
- Necesitan de un intérprete o programa en memoria para que en tiempo de ejecución se vaya traduciendo el código fuente a lenguaje máquina.
- Sus instrucciones o más bien el código fuente, escrito por el programador en un lenguaje de alto nivel, es traducido por el intérprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción.
- El proceso se repite cada vez que se ejecuta el programa, el código en cuestión.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes interpretados

- No existe un archivo ejecutable.
- El proceso de traducción se hace cada vez que se ejecuta.
- La ejecución es lenta ya que al proceso de ejecución tiene que sumarse el de traducción.
- El archivo puede ejecutarse en diferentes plataformas siempre que exista intérprete para ellas.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes interpretados

- El usuario utiliza el código fuente para ejecutar el programa.
- Los errores tipo léxico, sintácticos o semánticos pueden aparecer en la ejecución.
- Interrumpir la ejecución solo afecta normalmente al intérprete y no a la plataforma.

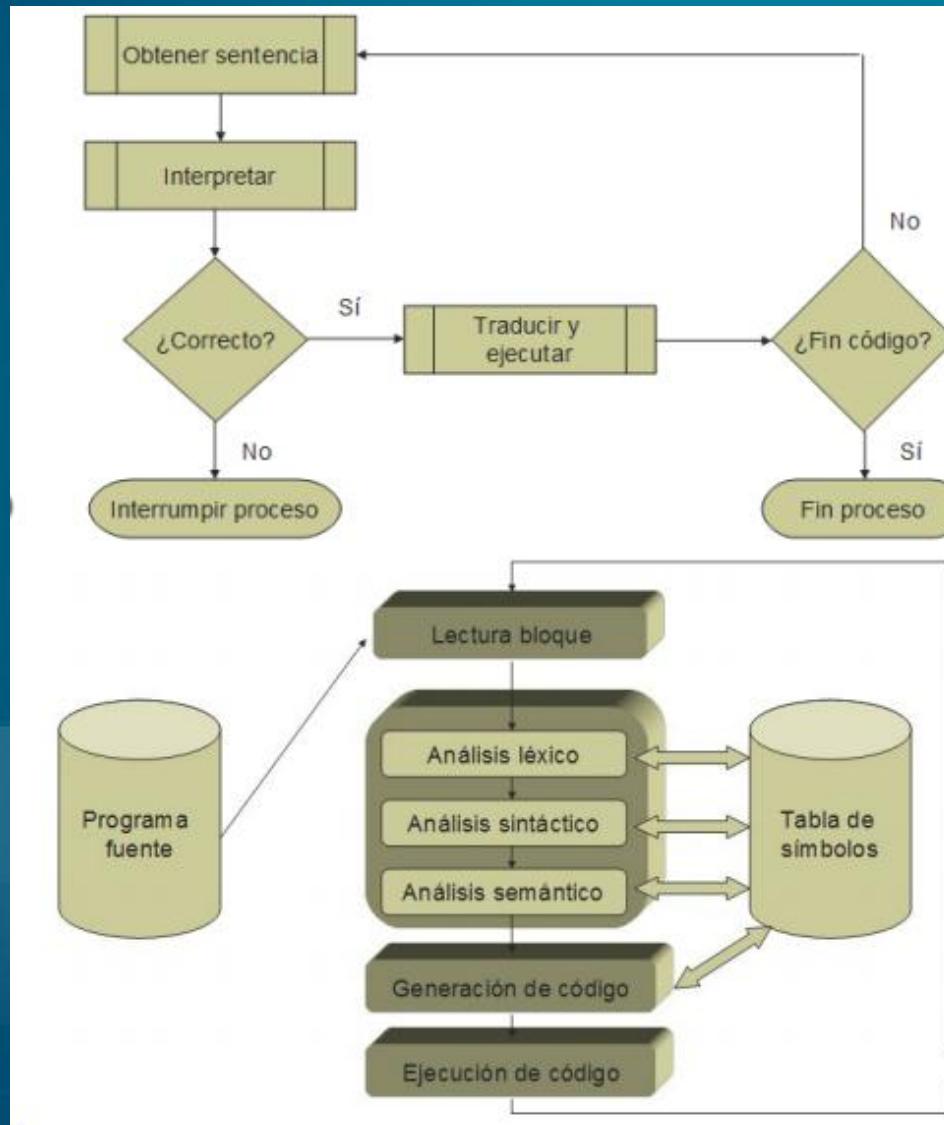
3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes interpretados

- Puede ejecutarse en cualquier plataforma siempre que haya intérprete para ella.
- Permite tipado dinámico de datos. No es necesario inicializar una variable con determinado tipo de dato.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes interpretados



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes interpretados

Lenguajes interpretados – Ejemplos

- Ruby.
- Python.
- PHP.
- JavaScript.
- Matlab.
- Visual Basic 6 y anteriores.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.

Lenguajes interpretados

- Se ven en el desarrollo de aplicaciones o sitios web que van acompañados de frameworks que facilitan en gran medida su programación.
- No es necesario que el usuario final posea, (lenguajes compilados), el compilador instalado en su ordenador para ejecutar el programa o el archivo objeto que este produce. Mayoritariamente necesitan un navegador actualizado y conexión a Internet para acceder y usar aplicaciones en línea.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Los lenguajes de máquina virtual hacen una compilación del código fuente pero, en lugar de compilar a código máquina, compilan a **código intermedio**.
- El código intermedio tiene varias características importantes:
 - Solo se genera si no se encuentran errores en el código fuente.
 - Es independiente de la plataforma.
 - Es muy cercano al lenguaje máquina (muy rápido de traducir, por lo tanto).
 - Una aplicación llamada máquina virtual, que es independiente de cada sistema, es la encargada de convertir el código intermedio al lenguaje máquina de la plataforma donde se quiera ejecutar.

3.- Clasificación

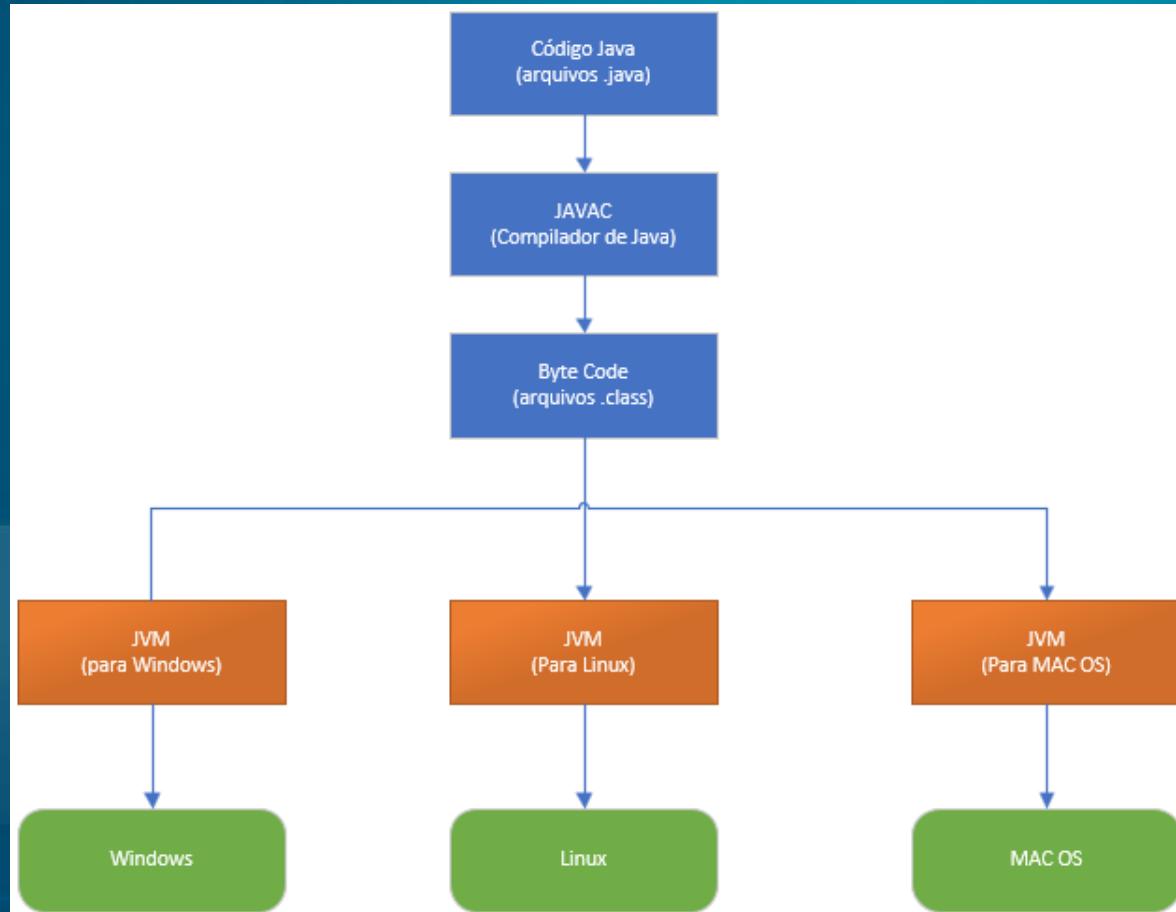
Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- En el caso de lenguajes de máquina virtual como Java ocurre que:
 - En tiempo de compilación el compilador Java genera el código intermedio llamado *bytecode Java* independiente de la plataforma.
 - En tiempo de ejecución, necesita de una máquina virtual Java (JVM) que interprete el *bytecode*. Esta máquina virtual es un SW que simula una máquina real con su sistema operativo.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Con este esquema se logra portabilidad entre plataformas y rapidez de ejecución.



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Este esquema aporta muchas de las ventajas de la compilación y la interpretación, deshaciéndose de algunos inconvenientes. Principalmente:
 - Portabilidad y rapidez: El código intermedio ya está libre de errores sintácticos, y es un código muy sencillo (al estilo del código máquina). Si existe un intérprete para este código en distintas plataformas, el mismo código se puede ejecutar en cada una de ellas.
Su ejecución más rápida, ya que no ha de comprobar la sintaxis.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Este esquema aporta muchas de las ventajas de la compilación y la interpretación, deshaciéndose de algunos inconvenientes. Principalmente:
 - **Estabilidad:** El código intermedio no es ejecutado por una CPU real directamente, sino por una CPU virtual: realmente, por el intérprete de la máquina virtual, que es un programa y no un chip real.
Permite un mayor control sobre este código, facilitando la labor de impedir que un código descontrolado afecte a la estabilidad de la plataforma real.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Imaginemos que disponemos de dos ordenadores: uno con un sistema operativo Windows y un procesador intel de 64 bits y el otro con un sistema operativo Linux y un procesador AMD de 32 bits.
- En el primer ordenador instalamos un compilador de Java y una máquina virtual de Java específicos para Windows 64 bits.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- En el segundo hacemos lo mismo, pero con un compilador y máquina virtual específicos para Linux 32 bits.
- Confeccionamos un programa sencillo (por ejemplo, que escriba "Hola Mundo" por la pantalla) escrito en Java en el primer ordenador y lo compilamos, generando un ejecutable intermedio. Si utilizamos la máquina virtual del primer ordenador para ejecutar ese código intermedio, comprobaremos que el programa escribe, en efecto "Hola Mundo" por la pantalla.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Si llevamos ese ejecutable intermedio tal cual a la segunda máquina, podremos utilizar la máquina virtual instalada allí para ejecutarlo, y comprobaremos que el resultado es exactamente el mismo: "Hola Mundo".

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- En lenguajes de ejecución administrada como los de la plataforma .NET de Microsoft ocurre que:
 - En tiempo de compilación, el compilador genera el código CIL (*Common Intermediate Language*) independiente de la plataforma.

3.- Clasificación

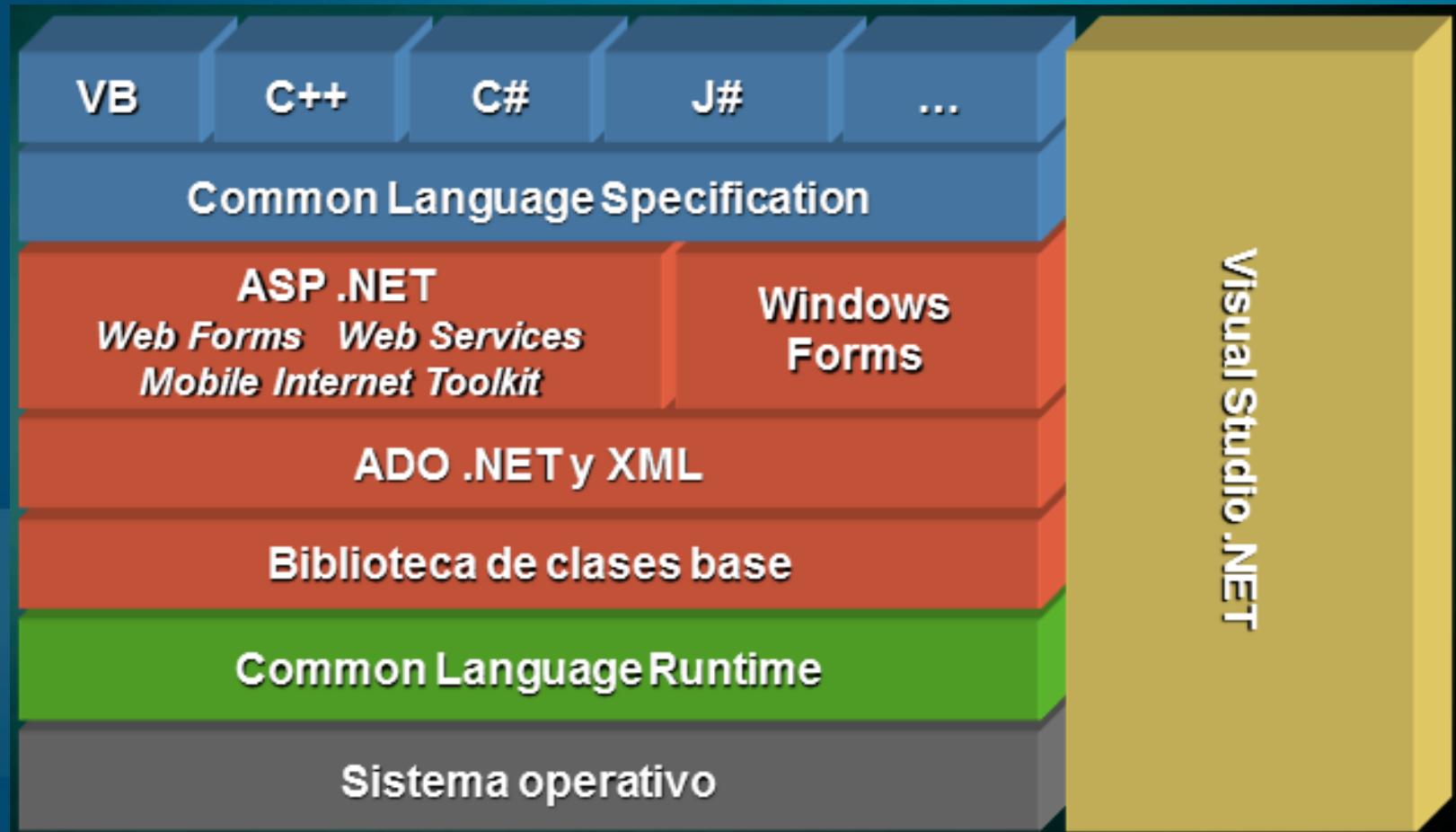
Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Para ejecutar la aplicación es necesario tener en el equipo cliente la versión de .NET Framework (CLR - *Common Language Runtime* - Motor de ejecución) adecuada y se hace la última fase de compilación, en la que el CLR traducirá el código intermedio a código máquina mediante un compilador JIT (*Just In Time*).

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

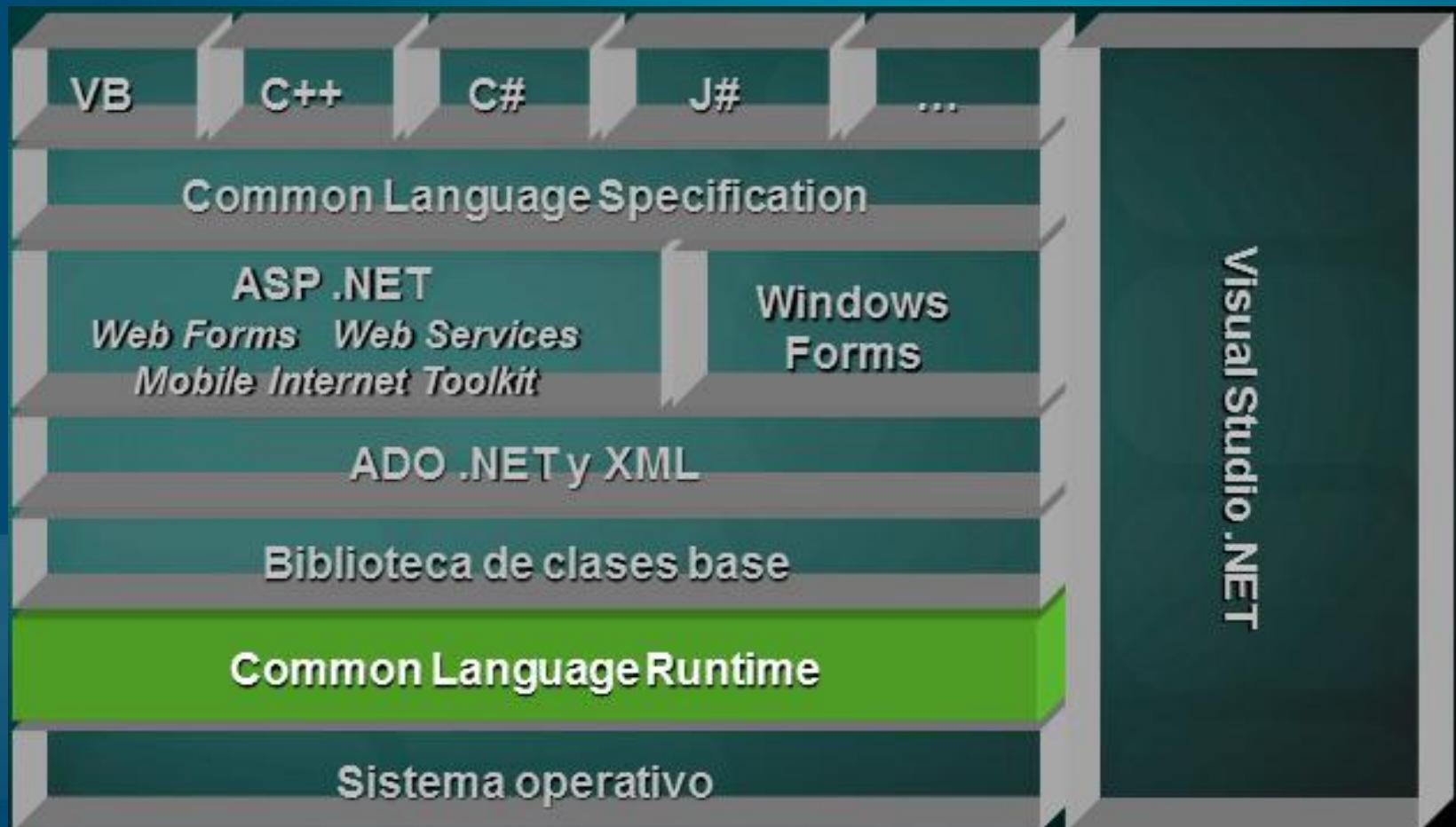
- Caso de Visual Studio .Net



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net - CLR



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

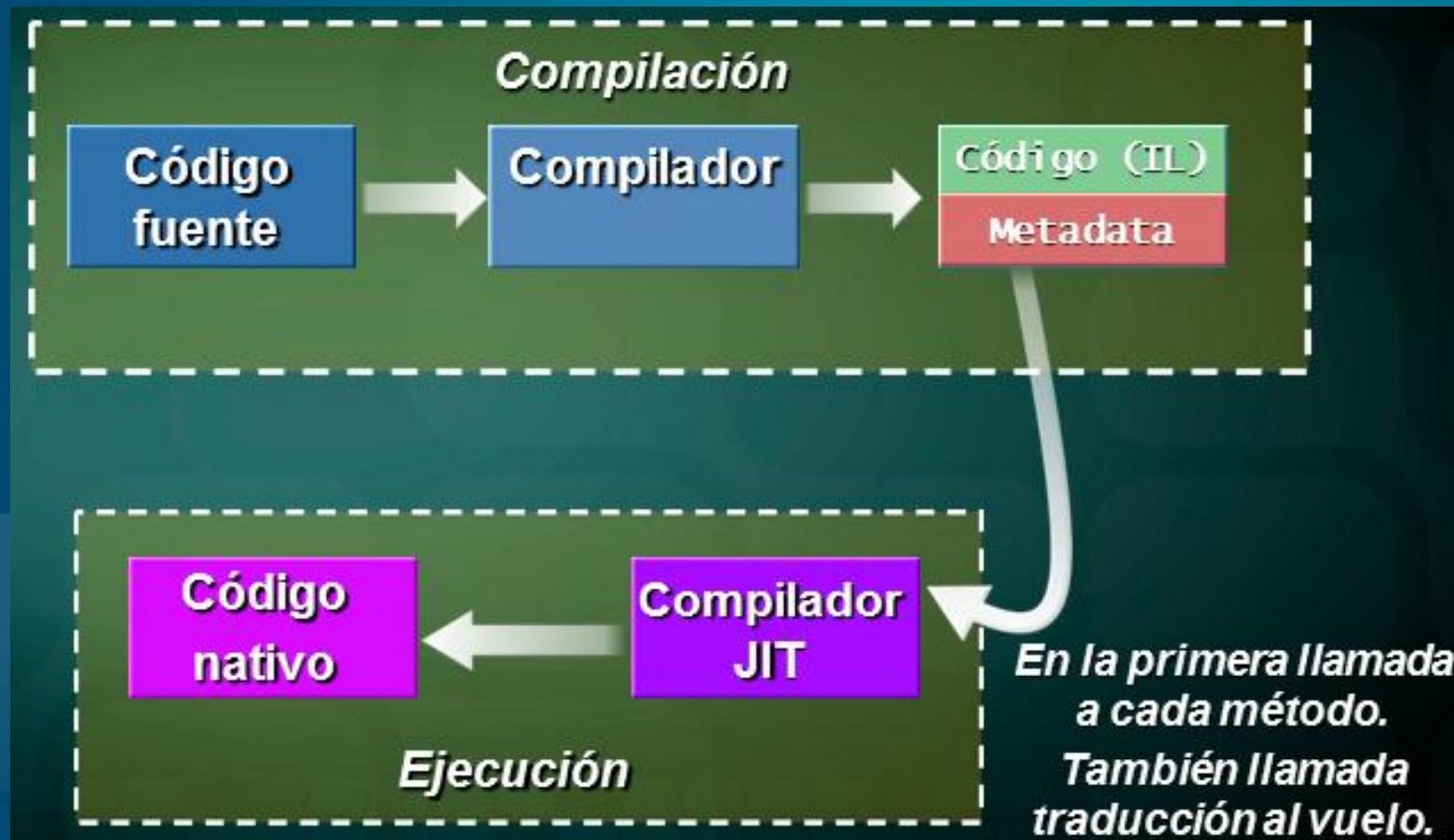
- CLR – Proceso de compilación



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CLR



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CLR
- El ensamblado (unidad mínima de ejecución, distribución, instalación y versionado) contiene lo siguiente:
 - El manifiesto.
 - Contiene información diversa sobre el ensamblado como atributos y dependencias.
 - MSIL (Microsoft Intermediate Language). El código fuente se compila a MSIL.
 - Recursos para el proyecto.



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CLR MSIL

```
.method private hidebysig static void Main(string[ args) cil
managed {
.entrypoint
maxstack 8
L_0000: ldstr "Hola Mundo"
L_0005: call void
          [mscorlib]System.Console::WriteLine(string)
L_000a: ret
}
```

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CLR

Para ejecutar directamente una aplicación .NET mediante el CLR, es necesario los denominados “CLR Hosts”.

Un CLR Host en una aplicación responsable por cargar el CLR en un proceso del sistema operativo, crear los application domains necesarios dentro de ese proceso y ejecutar la aplicación dentro de los application domains.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

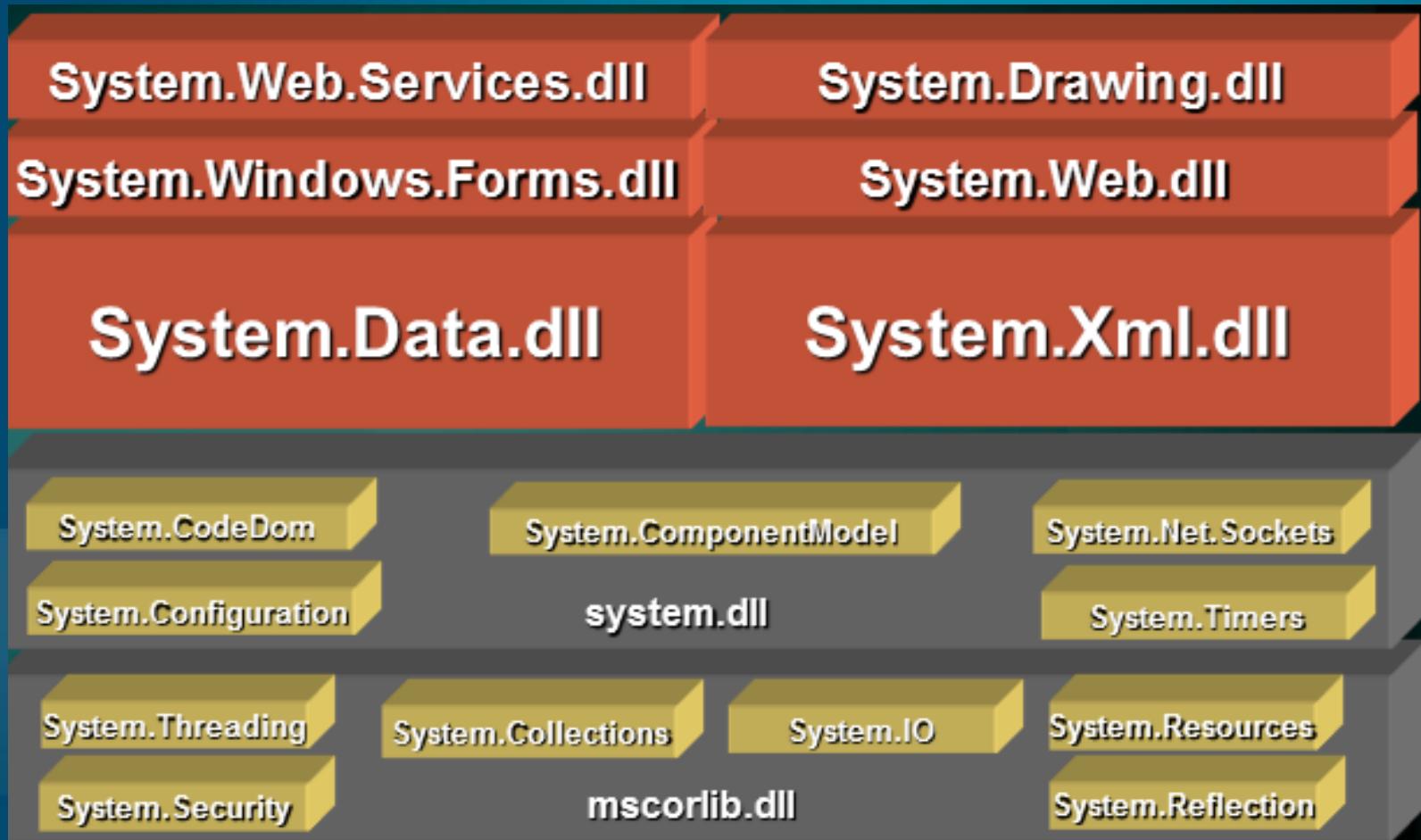
- Caso de Visual Studio .Net – Bibliotecas de clase



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

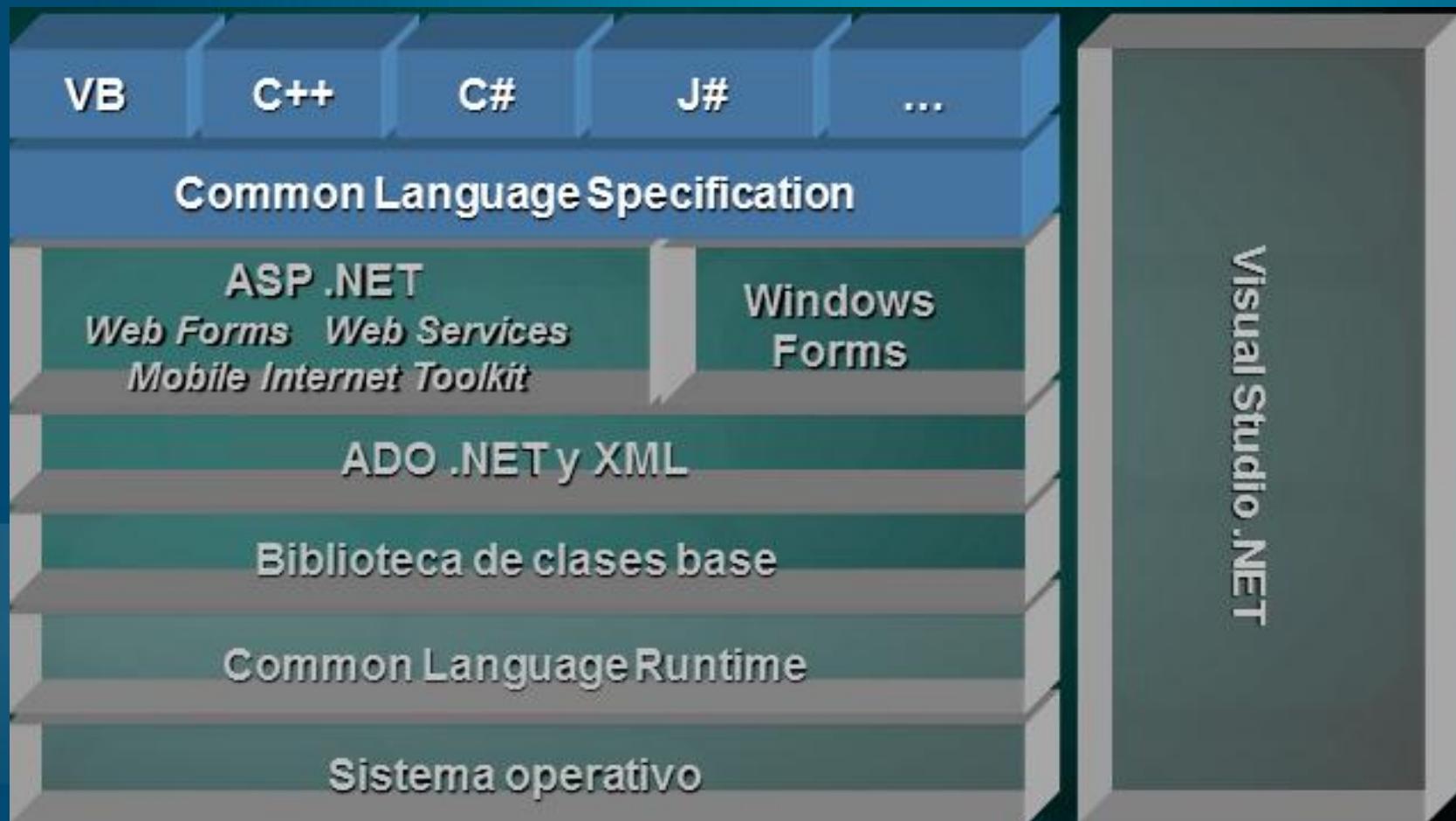
- Caso de Visual Studio .Net – Bibliotecas de clase



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CLS



3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CTS.
- La plataforma .Net cuenta con un sistema común de tipos conocido como CTS (Common Type System).
- En el CTS se recogen todos los tipos de datos y operaciones que es posible utilizar desde un lenguaje .Net.
- No importa si utilizamos Visual Basic .Net, C#, COBOL, Perl o Pascal. El conjunto de datos a utilizar es siempre el mismo.
- El CTS recoge también las operaciones que pueden efectuarse sobre los datos.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Caso de Visual Studio .Net – CTS.
- Existe un subconjunto del CTS compuesto de tipos de datos y operaciones que todos los lenguajes .Net deben implementar para ser compatibles con el resto. Este subconjunto es conocido como **CLS** (Common Language Specification).
- Por ejemplo, Visual Basic no dispone de un tipo de dato entero sin signo. A pesar de ello, Visual Basic .Net es un lenguaje .Net compatible porque los tipos de datos enteros no forman parte del CLS.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Existe un código intermedio que se ejecuta en un software específico pero no es un ejecutable.
- El proceso de traducción inicial se hace antes de la ejecución y el de la traducción final se hace cada vez que se ejecuta.
- La ejecución no es tan rápida como en los lenguajes compilados pero es más rápida que en los lenguajes interpretados.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- El archivo puede ejecutarse en diferentes plataformas siempre que exista el SW específico.
- El usuario no tiene el código fuente, sólo el código intermedio que no es manipulable fácilmente para obtener el código fuente, por lo que el programador tiene código fuente más protegido.

3.- Clasificación

Según la forma de traducirse a lenguaje máquina y ejecutarse.
Lenguajes de máquina virtual o de ejecución administrada

- Los errores de tipo léxico, sintáctico o semántico se detectan en fase de compilación, y si existen no se generará el código intermedio.
- Interrumpir la ejecución sólo afecta normalmente al intérprete y no a la plataforma.
- La modificación del código fuente implica volver a repetir el proceso de compilación.

3.- Clasificación

Según la arquitectura cliente-servidor

La arquitectura cliente-servidor consiste básicamente en un programa cliente que realiza peticiones a un servidor.

Es más útil cuando el cliente o el servidor están comunicados mediante una red, aunque también se puede aplicar cuando están en la misma máquina.

- Se diferencia entre lenguajes que se ejecutan:
 - Del lado del servidor como PHP
 - Del lado del cliente como Javascript.

3.- Clasificación

Según la arquitectura cliente-servidor

- Se puede suponer un navegador cliente y un servidor web con intérprete de PHP y los siguientes pasos:
 - Un usuario desde un navegador cliente pide a un servidor Web una página HTML que tiene código PHP y código Javascript.
 - El servidor Web procesa la petición, interpreta el código PHP, lo ejecuta colocando el resultado de la ejecución en el sitio donde estaba el código PHP y devuelve al cliente una página web con código HTML y Javascript.

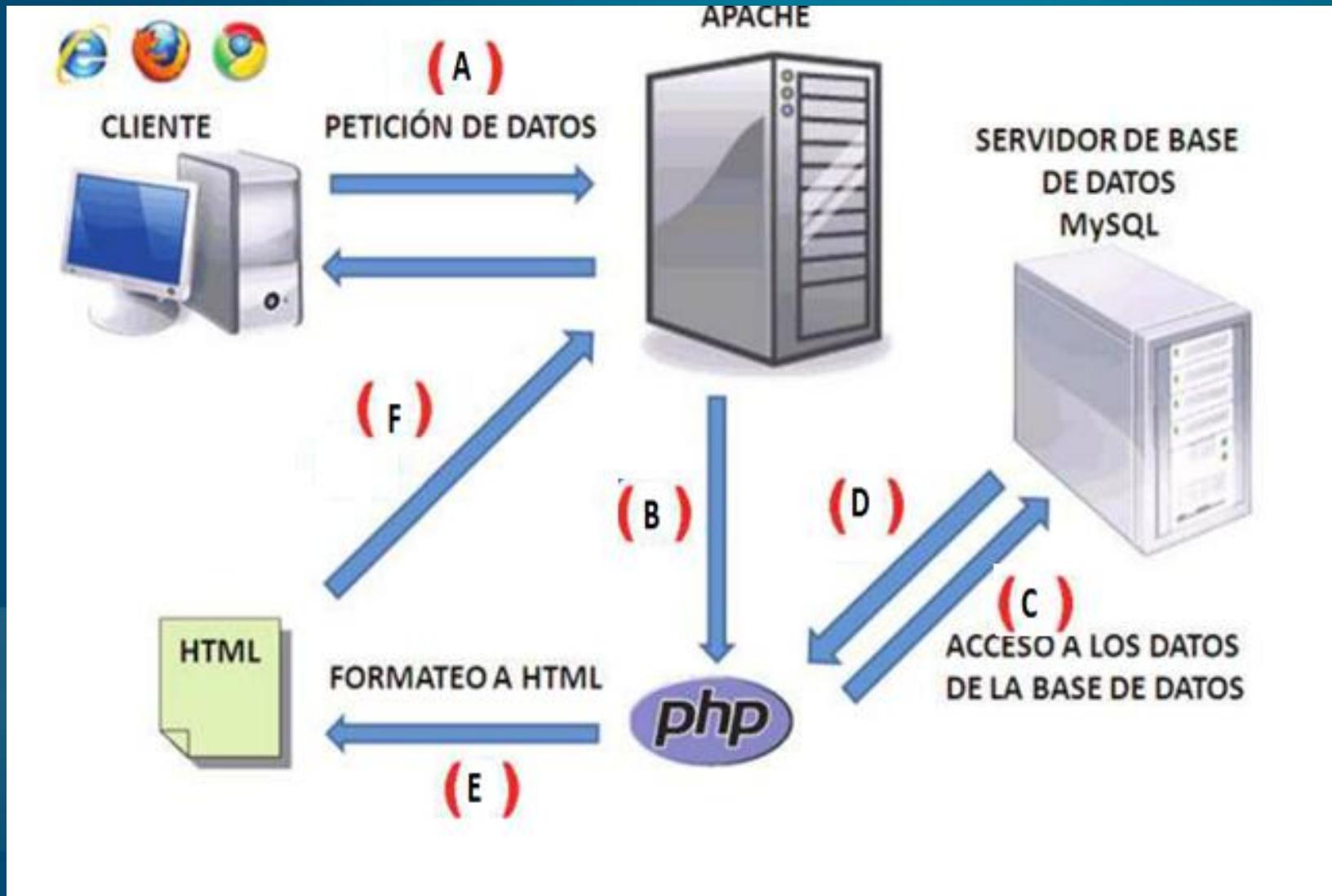
3.- Clasificación

Según la arquitectura cliente-servidor

- El servidor Web debería de ejecutar las órdenes relativas al manejo de una BD en un servidor de BD si el código PHP tuviera ese tipo de instrucciones.
- El navegador cliente interpreta el código Javascript y muestra la página al usuario.
- El usuario puede interactuar con la página y cada vez que hace peticiones al servidor se recarga toda la página con la respuesta del servidor.

3.- Clasificación

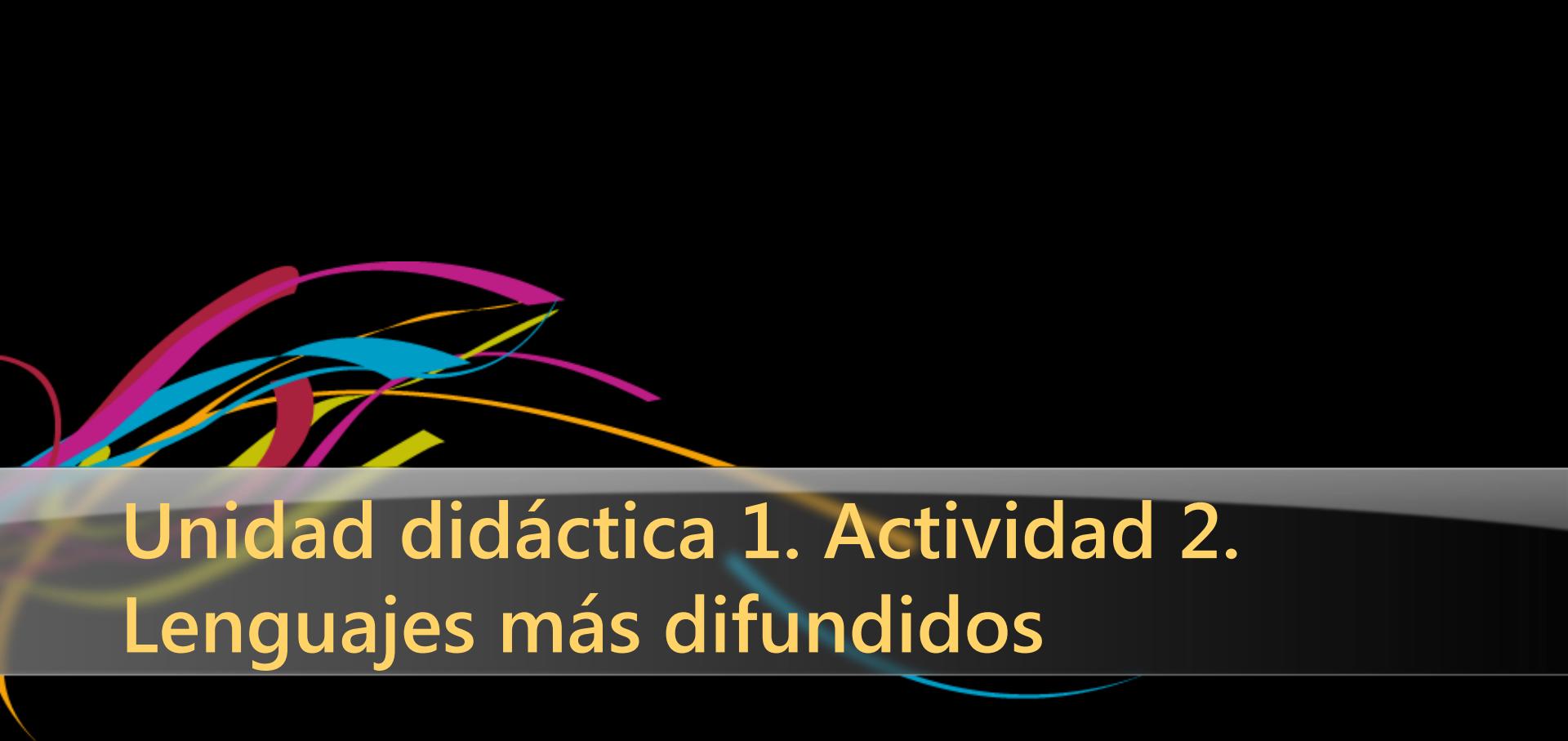
Según la arquitectura cliente-servidor



3.- Clasificación

Según la arquitectura cliente-servidor

- Existe una técnica denominada AJAX (*Asynchronous JavaScript And XML*), que permite que JavaScript procese algún evento iniciado por el usuario haciendo una petición al servidor que este responde con resultado en XML.
- Javascript procesa el resultado XML actualizando secciones de la página sin tener que recargarla totalmente y logrando así una interacción asíncrona entre servidor y cliente.



Unidad didáctica 1. Actividad 2. Lenguajes más difundidos

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

4.- Lenguajes más difundidos

- Pueden consultarse los siguientes índices o clasificaciones:
- Índice Tiobe
[\(http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html\)](http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

Basa la clasificación de los lenguajes en el número de ingenieros cualificados en cada lenguaje en todo el mundo, cursos de lenguajes ofertados, proveedores que trabajan sobre esos lenguajes, búsquedas realizadas en Google, Bing, Yahoo, Wikipedia, Amazon, Youtube e Baidu, y una serie de premisas como que el lenguaje exista como lenguaje de programación en Wikipedia y tenga por los menos 10.000 visitas en Google.

4.- Lenguajes más difundidos

Índice Tiobe

Oct 2020	Oct 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	16.95%	+0.77%
2	1	▼	Java	12.56%	-4.32%
3	3		Python	11.28%	+2.19%
4	4		C++	6.94%	+0.71%
5	5		C#	4.16%	+0.30%
6	6		Visual Basic	3.97%	+0.23%
7	7		JavaScript	2.14%	+0.06%
8	9	▲	PHP	2.09%	+0.18%
9	15	▲	R	1.99%	+0.73%
10	8	▼	SQL	1.57%	-0.37%

Fuente: índice Tiobe. Octubre 2020

4.- Lenguajes más difundidos

- Índice PYPL ou PopularitY of Programming Language index (<http://pypl.github.io/PYPL.html>)

Basa la clasificación de los lenguajes de programación en el análisis de la frecuencia de búsqueda de tutoriales o guías de los lenguajes de programación en Google utilizando Google Trends.

4.- Lenguajes más difundidos

- Índice PYPL ou PopularitY of Programming Language index.

Rank	Change	Language	Share	Trend
1		Python	31.02 %	+2.2 %
2		Java	16.38 %	-2.8 %
3		JavaScript	8.41 %	+0.4 %
4		C#	6.52 %	-0.6 %
5		PHP	5.83 %	-0.4 %
6		C/C++	5.56 %	-0.4 %
7		R	4.26 %	+0.4 %
8		Objective-C	3.48 %	+0.8 %
9		Swift	2.37 %	-0.1 %
10		TypeScript	1.9 %	+0.1 %

Fuente: índice PYPL. Octubre 2020

4.- Lenguajes más difundidos

- La clasificación Redmonk

(<https://redmonk.com/sogrady/2020/07/27/language-rankings-6-20/>)

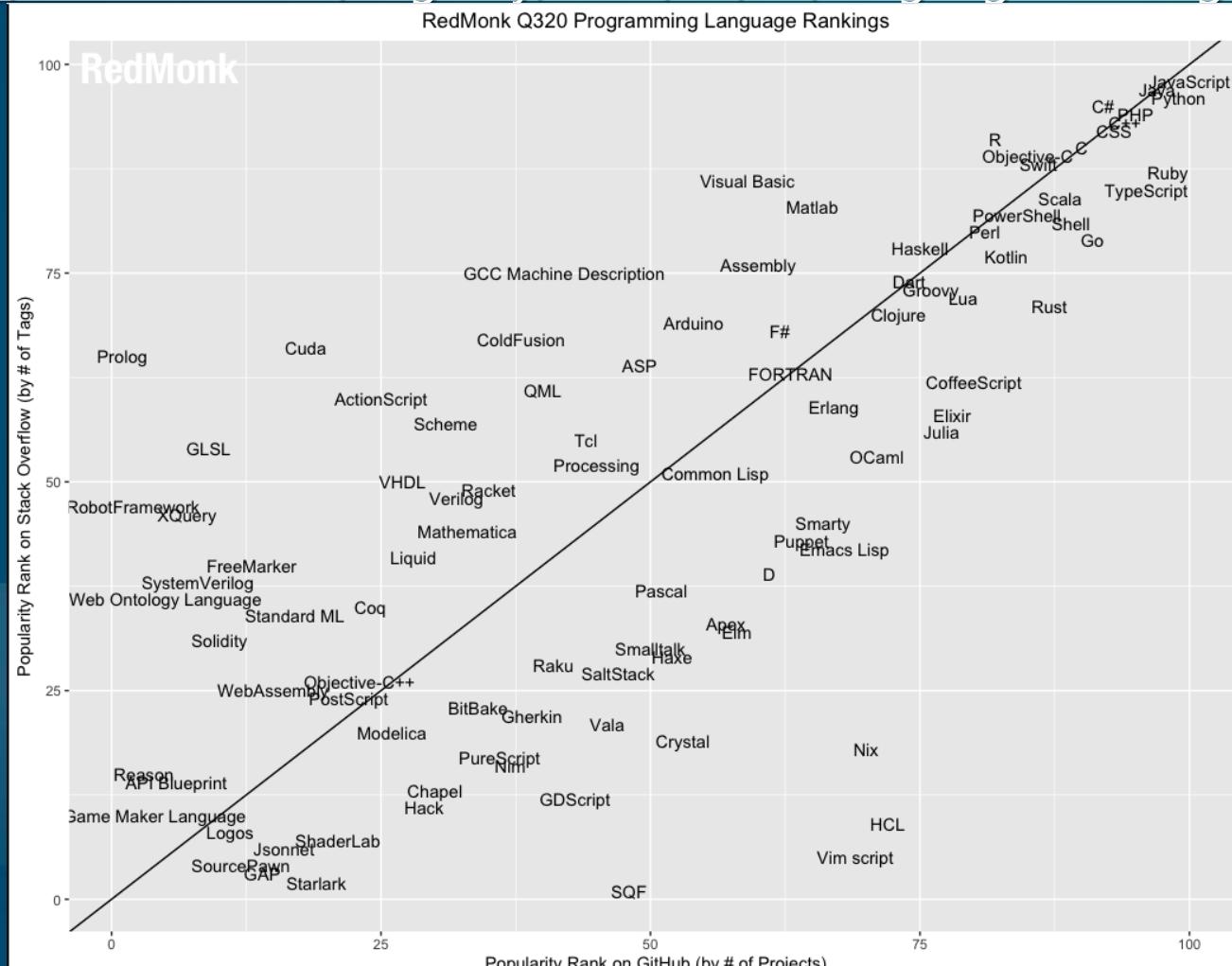
No se basa la clasificación en los buscadores sino en los proyectos albergados en el repositorio GitHub y en las preguntas de la web de *StackOverflow* orientada a programadores.

Incluye lenguajes informáticos y no solo lenguajes de programación.

4.- Lenguajes más difundidos

La clasificación Redmonk

(<http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/>)

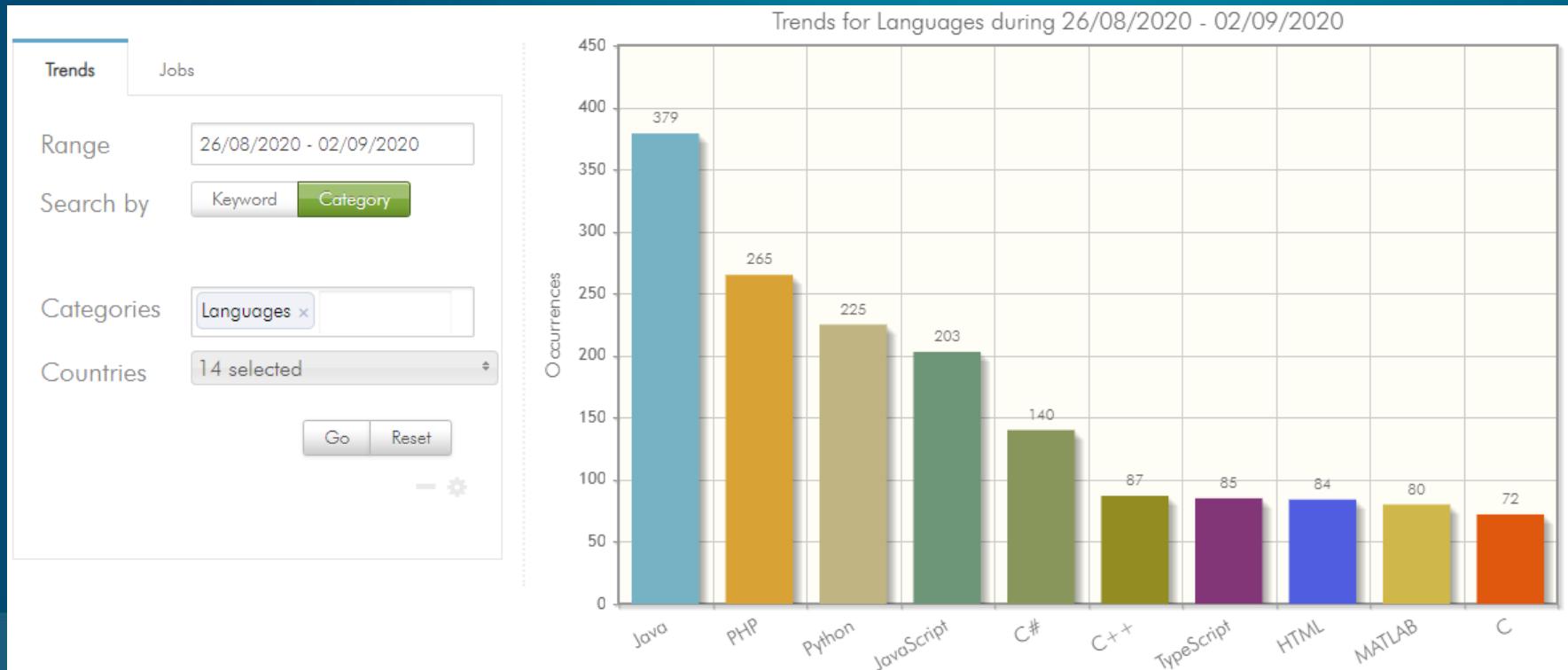


4.- Lenguajes más difundidos

- La clasificación Trendyskills
(<http://trendyskills.com/>)

Se basa en las ofertas de empleo para los lenguajes de programación en España, USA, UK, Alemania, Suecia, Países Bajos, Irlanda, Suiza, Austria, Bélgica, Finlandia, República Checa e Grecia e incluyen lenguajes informáticos y no sólo lenguajes de programación.

4.- Lenguajes más difundidos

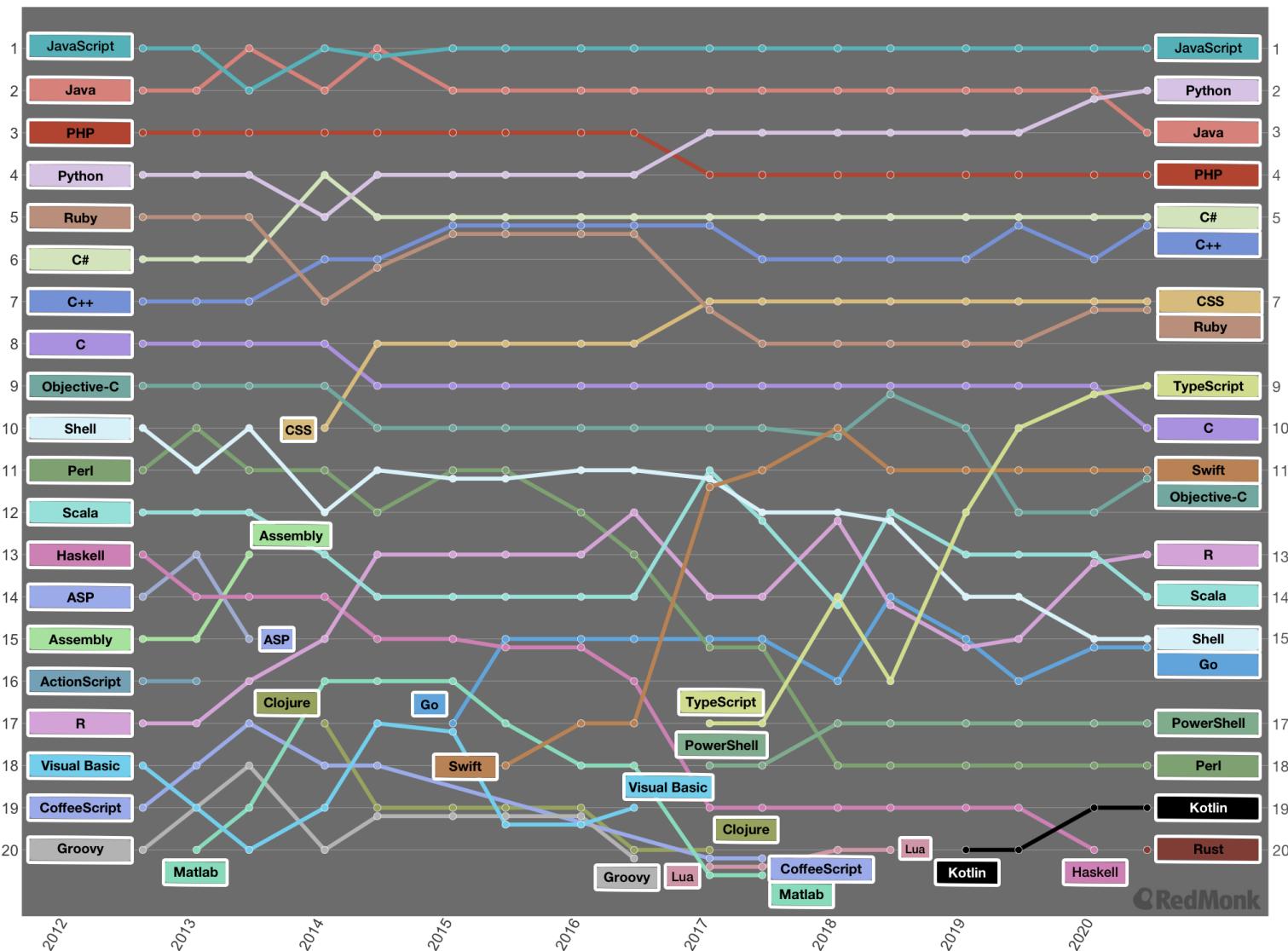


Fuente: índice Trendy skills. Septiembre 2020

4.- Lenguajes más difundidos

RedMonk Language Rankings

September 2012 - June 2020



4.- Lenguajes más difundidos

Lenguaje C

- Es un lenguaje creado por Dennis Ritchie en 1972 para codificar el SO Unix.
- Catalogado como un lenguaje de alto nivel, estructurado y modular, pero con muchas características de bajo nivel como utilizar punteros para hacer referencia a una posición física de memoria RAM.
- Estas características posibilitan trabajar muy cerca de la máquina pero los programas son más complicados y propensos a tener más errores.
- Influyó en el diseño de los lenguajes: C++, Java, PHP, Javascript entre otros.

4.- Lenguajes más difundidos

Lenguaje C

```
/*
 * Programa C que suma os números enteiros do 1 ó 20
 */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
    int i;                      /* contador de números enteiros */
    long int suma;              /* sumador dos números enteiros */
    int final;
    suma = 0;
    for (i = 1; i <=20; i++) {
        suma = suma + i;
    }
    printf("Programa C\nA suma total e: %ld\n", suma);
    printf("\nTeclee calquera numero para finalizar... ");
    scanf("%d",&final);
    fflush(stdin);
    return (EXIT_SUCCESS);
}
```

4.- Lenguajes más difundidos

Lenguaje C++

- Diseñado en 1980 por Bjarne Stroustrup para extender C con mecanismos que permitan POO.

```
/*
 * Programa C++ que suma os números enteiros do 1 ó 20
 */
#include <iostream>
int main(int argc, char**argv) {
    int i;                      /* contador de números enteiros */
    long int suma;              /* sumador dos números enteiros */
    suma = 0;
    for (i = 1; i <=20; i++) {
        suma = suma + i;
    }
    std::cout <<"A suma total e "<< suma<< std::endl;
    return 0;
}
```

4.- Lenguajes más difundidos

Lenguaje Java

- En un lenguaje orientado al objeto desarrollado por Sun Microsystems en 1995.
- Hereda mucha sintaxis de C y C++ pero con un modelo de objetos más simple y elimina las herramientas de bajo nivel que se utilizaban en C.

```
package sumaenteiros;
/*
 * File: Main.java
 * Author: profesor
 * Date: 14/08/2011 12:25:00
 * Obxectivo: visualizar a suma dos 20 primeiros números naturais
 */
public class Main {

    public static void main(String[] args) {
        int suma=0;      /* sumador dos números enteiros */
        for (int i=1;i<=20;i++)
        {
            suma=suma+i;
        }
        System.out.printf("Exemplo Java\n");
        System.out.printf("Suma dos 20 primeiros números naturais =
%d\n",suma);
    }
}
```

4.- Lenguajes más difundidos

Lenguaje C#

- Es un lenguaje orientado al objeto desarrollado Microsoft en el año 2000 como parte de la plataforma .NET.
- Visual C# proporciona un editor de código avanzado, diseñadores de interfaz de usuario y numerosas herramientas para facilitar el desarrollo de aplicaciones en C# y .NET Framework.

```
using System;

namespace exemplo_csharp_consola
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo");
        }
    }
}
```

4.- Lenguajes más difundidos

Lenguaje PHP (Hypertext PreProcessor)

- Es un lenguaje interpretado del lado del servidor que se puede usar para crear cualquier tipo de programa pero que donde tiene más popularidad es en la creación de páginas web dinámicas.
- La ejecución de código PHP en un servidor suele generar código HTML que se envía a los navegadores de los clientes.
- Diseñado por Rasmus Lerdorf en 1995 y desarrollado actualmente por el grupo PHP.

4.- Lenguajes más difundidos

Lenguaje PHP (Hypertext PreProcessor)

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>exemplo de php</title>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
  </head>
  <body>
    <?php
      $sum=0;
      for ($i=1;$i<=20;$i=$i+1)
      {
        $sum=$sum+$i;
      }
      echo "<p>A suma total é: ".$sum."</p>";
    ?>
  </body>
</html>
```

4.- Lenguajes más difundidos

Lenguaje Python

- Es un lenguaje de programación interpretado que permite la POO, con una sintaxis limpia que favorece que el código sea legible.
- Diseñado por Guido Van Rossum en 1991, actualmente subministrado por *Python Software Foundation*.
- Posee licencia de código abierto denominada *Python Software Foundation License 1* compatible con la licencia GNU a partir de la versión 2.1.1

4.- Lenguajes más difundidos

Lenguaje Python

```
# suma os enteiros do 1 ao 20

def sumarEnteiros(n):
    sum=0
    for i in range(1,n+1):
        sum=sum+i

    return sum

A=sumarEnteiros(20)

print "Suma total de enteiros do 1 ao 20 : " + str(A)
```

4.- Lenguajes más difundidos

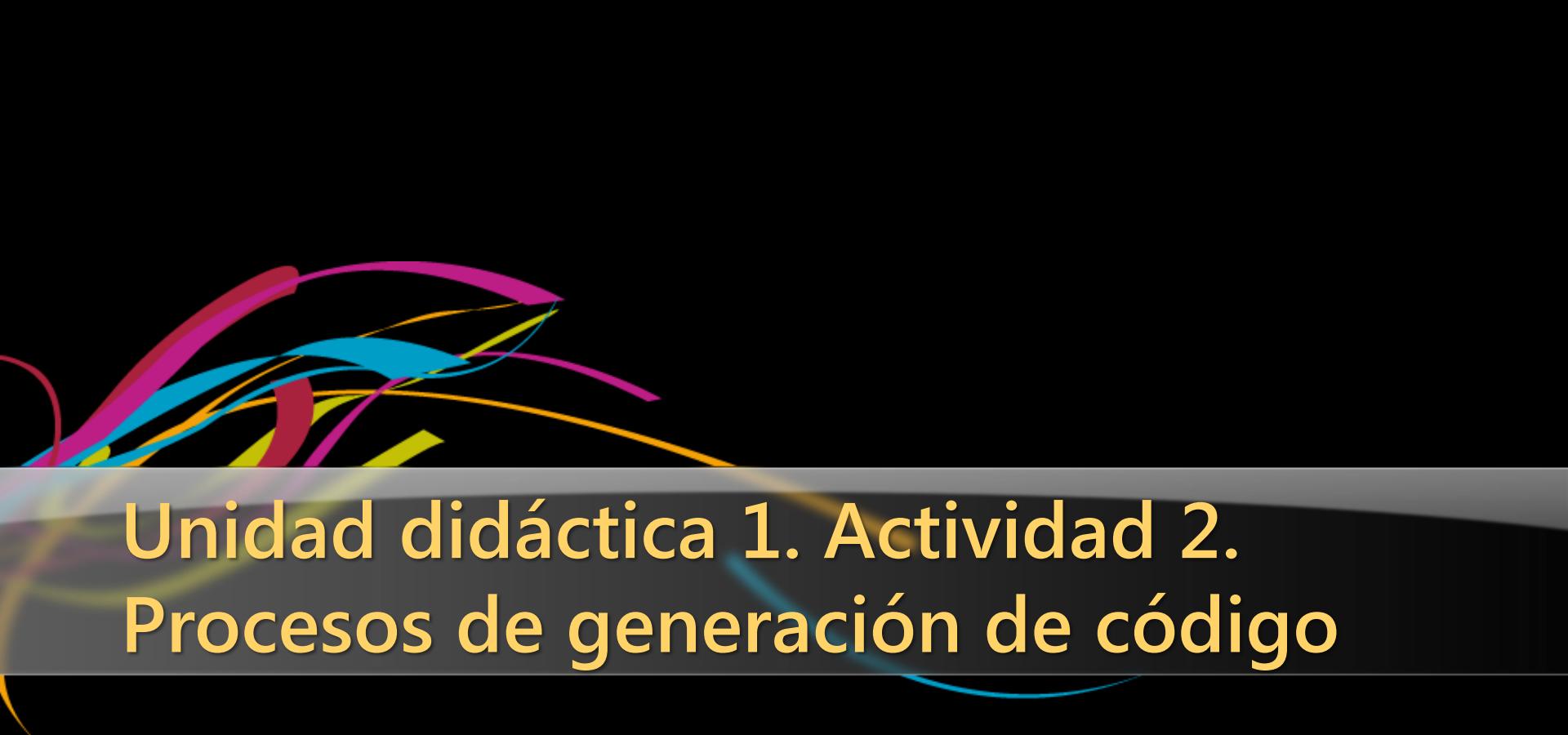
Lenguaje JavaScript

- Es un lenguaje de programación dialecto del estándar ECMAScript.
- Se define como orientado a objetos, basado en prototipos, débilmente tipado (declaración de tipos) y dinámico.
- Se utiliza normalmente en el lado del cliente (*client-side*) y está implementado como parte de un navegador web aunque también existe un JavaScript del lado del servidor (*Server-side JavaScript o SSJS*).

4.- Lenguajes más difundidos

Lenguaje JavaScript

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>exemplo de JavaScript</title>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
  </head>
  <body>
    <h1>Números naturais impares ata 9</h1>
    <script type="text/javascript">
      <!--
      var i;
      for(i=1;i<=10;i+=2)
        document.write(i+" ");
      // -->
    </script>
  </body>
</html>
```



Unidad didáctica 1. Actividad 2. Procesos de generación de código

Ciclo Superior Desarrollo de aplicaciones multiplataforma
IES Muralla Romana

5.- Proceso generación código

Edición

- La generación de código consta de los procesos de edición, compilación y enlace.

- **Edición**

Esta fase consiste en escribir el algoritmo de resolución en un lenguaje de programación mediante un editor de texto o herramienta de edición incluida en un entorno de desarrollo.

El código resultante se llama código fuente y el archivo correspondiente archivo fuente.

5.- Proceso generación código

Compilación

- Consiste en analizar y sintetizar el código fuente mediante un compilador, para obtener, sino se encuentran errores, el código objeto o código intermedio multiplataforma.

Esta fase no se aplicará a los lenguajes interpretados aunque éstos pueden tener herramientas que permitan hacer un análisis léxico y sintáctico antes de pasar a ejecutarse.

5.- Proceso generación código

Compilación. Análisis

Los análisis realizados son:

- **Análisis léxico** lee el archivo fuente carácter por carácter y forma grupos de caracteres (*lexemas*) con un significado léxico mínimo (*tokens*). Elimina los componentes no esenciales del programa fuente, e ignora espacios en blanco, tabuladores, caracteres fuera de línea, comentarios y todo lo que no sea necesario en fases posteriores.

Testigo	Valor
IDENTIFICADOR	ValorX
ASIGNACIÓN	=
IDENTIFICADOR	ValorY
SUMA	+
ENTERO	1
PUNTO_Y_COMA	;

5.- Proceso generación código

Compilación. Análisis

Ejemplos:

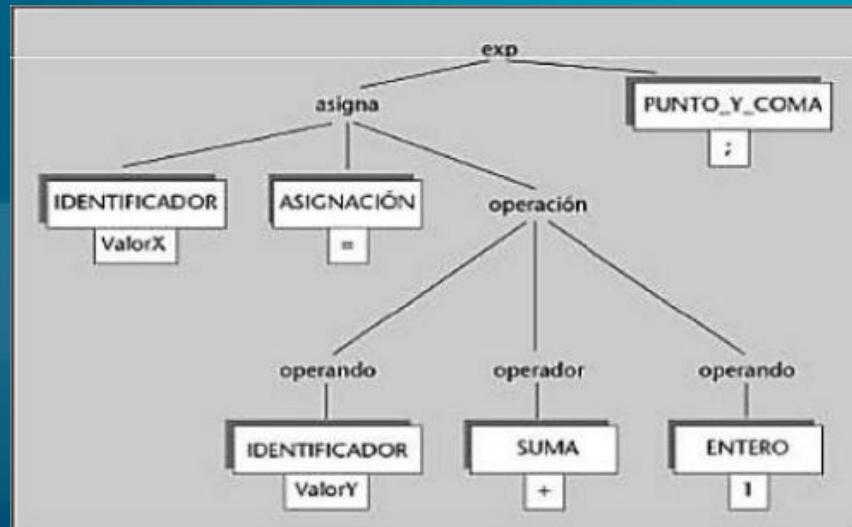
- Números inválidos: un número contiene caracteres inválidos (por ejemplo: 2,13 en lugar de 2.13), no está formando correctamente (por ejemplo, 0.1.33).
- Cadenas incorrectas de caracteres: omisión de comillas.
- Errores de ortografía en palabras reservadas.

5.- Proceso generación código

Compilación. Análisis

- Análisis sintáctico utiliza los tokens encontrados por el analizador léxico y comprueba si llegan en el orden correcto proporcionado por la gramática que define el lenguaje fuente.

La salida suele ser un árbol sintáctico con la estructura sintáctica del programa fuente.



5.- Proceso generación código

Compilación. Análisis

- Análisis sintáctico: el analizador sintáctico espera un símbolo que no corresponde al que se acaba de leer.

Ejemplos

- Paréntesis o corchetes omitidos, por ejemplo:
 $x := y * (1 + z;$
- Operadores u operando omitidos, por ejemplo:
 $x := y (1 + z);$

5.- Proceso generación código

Compilación. Análisis

- Análisis semántico se ocupa de comprobar el significado de las sentencias.

Puede haber sentencias sintácticamente correctas pero que no puedan ejecutar por no tener ningún sentido.

Comprueba, entre otros, que las variables utilizadas estén declaradas, la coherencia entre el tipo de datos de una variable y el valor almacenado, o la comparación de un número y tipo de parámetros entre la definición y una llamada a un método.

5.- Proceso generación código

Compilación. Análisis

- Análisis semántico
 - Ejemplos
 - *Identificadores no definidos;*
 - *Operadores y operandos incompatibles.*

5.- Proceso generación código

Síntesis

Permite

- La generación de código intermedio independiente de la máquina.
- Algunos lenguajes compilados como C pasan antes por una fase de preprocessamiento en las que se lleva a cabo operaciones como sustituir las constantes por el valor o incluir archivos de cabecera.
- Otros lenguajes como Java generan Bytecode Java que podrá ser ejecutado en una JVM y otras como C# genera el código CIL que será ejecutado en el entorno CLR.

5.- Proceso generación código

Síntesis

- La traducción del código intermedio anterior al código máquina para obtener el código objeto. Esta traducción lleva consigo también una optimización de código.
- Este nuevo código aún no está listo para ejecutarse directamente.

5.- Proceso generación código

Enlace

Consiste en enlazar mediante un programa enlazador el archivo objeto obtenido en la compilación con módulos objetos externos para obtener, sino se encuentran errores, el archivo ejecutable.

El archivo obtenido en la compilación puede tener referencias a códigos objeto externos que forman parte de bibliotecas externas estáticas o dinámicas:

- Si la biblioteca es **estática**, el enlazador añade códigos objeto de las bibliotecas al archivo objeto, por lo que el archivo ejecutable resultante aumenta de tamaño con relación al archivo objeto, pero no necesita nada más que el SO para ejecutarse.

5.- Proceso generación código

Enlace

- Si la biblioteca es dinámica (*Dynamic Link Library - DLL en Windows, Shared objects en Linux*), el enlazador añade referencias a la biblioteca, por lo que el archivo ejecutable resultante apenas aumenta de tamaño con relación al archivo objeto, pero la biblioteca dinámica tiene que estar accesible cuando el archivo ejecutable se ejecute.

5.- Proceso generación código

Ejecución

- La ejecución necesita de herramientas diferentes dependiendo de si el lenguaje es interpretado, compilado o de máquina virtual o ejecución administrada.
- Si el lenguaje es interpretado será necesario el archivo fuente y el intérprete para que este vaya traduciendo cada instrucción del archivo fuente a lenguaje máquina (análisis y síntesis) y ejecutándola. Ej. Python
- Si el lenguaje es compilado será necesario el archivo ejecutable, y en algunos casos también se necesitan bibliotecas dinámicas. Ej. C

5.- Proceso generación código

Ejecución

- Si el lenguaje necesita de máquina virtual, será necesario tener el código intermedio y la máquina virtual para que vaya traduciendo el código intermedio a lenguaje máquina y ejecutándolo.
- Ej. Java o los programas para la plataforma Android.