Técnico en

DESARROLLO DE SOFTWARE

Lógica de Programación I

Lógica de Programación I

Unidad I

1. Conceptos de programación

Algoritmo - es una secuencia ordenada de instrucciones finitas que deben seguirse para resolver un problema.

Es importante tener en cuenta que las instrucciones deben darse de forma ordenada, ya que de lo contrario podremos obtener un resultado no deseado.

Por ejemplo: Algoritmo para cocinar un pastel.

1. Pre calentar el horno.
2. Mezclar los ingredientes.
3. Engrasar el molde.
4. Colocar la mezcla en el molde.
5. Hornear la mezcla.
6. Sacar el pastel del horno.

Si tenemos esta propuesta de algoritmo para cocinar un pastel, los pasos pueden ir variando dependiendo del tipo de pastel o de cualquier agente externo (ej. que no se tengan todos los ingredientes); pero existen pasos de este algoritmo que no deben variar,

por ejemplo: no podemos sacar el pastel del horno sin antes haberlo horneado, y es por eso que decimos que la secuencia de instrucciones debe estar ordenada.

Programa

Es la implementación de un algoritmo de tal forma que la computadora entienda. Para la implementación de los algoritmos se utilizan lenguajes de programación.

Programación

Es el proceso mediante el cual se lleva a cabo la creación de código fuente de un programa mediante un lenguaje definido.

Lenguaje de Programación

Un lenguaje de programación provee las abstracciones, principios de organización y las estructuras de control para escribir programas.

2. Tipos de Lenguajes de Programación

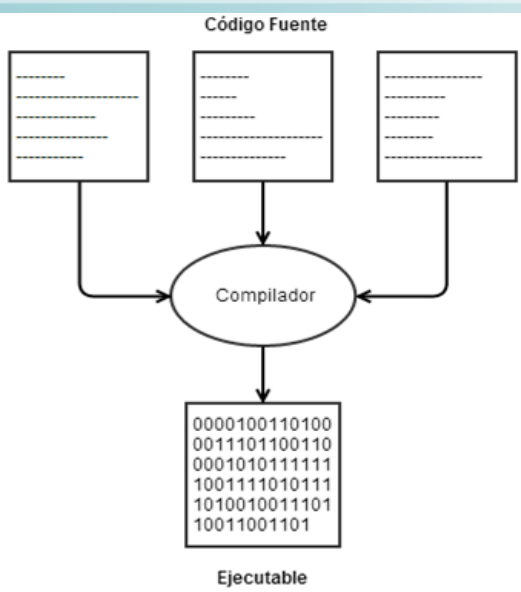
Solo hay un tipo de lenguaje de programación que las computadoras pueden entender e interpretar: el código binario, también llamado código máquina. Este es el nivel más bajo de los lenguajes en los que se puede escribir un programa. Todos los demás lenguajes se pueden distinguir como de alto nivel o bajo nivel de acuerdo a que tanto se asemejan al código máquina.

Para ejecutar un programa en la computadora se necesita convertir el conjunto de instrucciones escritas en lenguaje de programación (código fuente) a código máquina. Existen dos formas principales de hacer esto: utilizando compilación o interpretación del código fuente.

Lenguajes compilados

Compilar consiste en leer el código fuente línea por línea y traducir cada instrucción de alto nivel por varias instrucciones en código máquina que realizan lo que la instrucción de alto nivel declaró. Al final se obtendrá un conjunto de instrucciones en código máquina que son grabadas en un archivo que contiene una estructura interna que un determinado sistema operativo es capaz de entender, este archivo es llamado “ejecutable”. El tipo de programa que se encarga de realizar estas traducciones se denomina **compilador**.

Lenguajes de programación compilados: C, C++, Java, Objective C y otros.



Lenguajes interpretados

Interpretar es algo totalmente distinto de compilar pero el resultado es el mismo. El programa interprete reside en la memoria y directamente traduce y ejecuta el programa de alto nivel sin traducción preliminar a código máquina.

Lenguajes de programación interpretados: PHP (se interpreta del lado del servidor), JavaScript y otros.

Ventajas y Desventajas

	Lenguaje compilado	Lenguaje interpretado
Ventajas	- Listo para ejecutarse - A menudo más rápido	- Multiplataforma - Más fácil de probar

	- El código fuente es privado	- Fácil de detectar errores
Desventajas	- No es multiplataforma - Inflexible - Se necesita un paso extra al compilar	- A menudo más lento - El código fuente es público - Se necesita un intérprete

Lenguajes Híbridos

También existe un tipo de lenguaje híbrido que utiliza compilación e interpretación para aprovechar las ventajas de ambas. La similitud con el lenguaje compilado es que traduce el código fuente y genera un ejecutable, pero no lo traduce a código máquina sino a un pseudocódigo denominado código-p.

El código-p es bastante simple pero más poderoso que el código máquina. Al ejecutar el programa se interpreta el código-p. El lenguaje híbrido es casi tan rápido como el lenguaje compilado pero conserva la flexibilidad del lenguaje interpretado.

Lenguajes híbridos: Python, Perl y otros.

Escribir código fuente

El código fuente de los lenguajes de programación está escrito en texto plano. Los editores de texto simple funcionan muy bien para escribir código fuente de cualquier lenguaje de programación por ejemplo el Bloc de Notas en Windows o TextEdit en Mac.

Para empezar a aprender a programar no hay ningún lenguaje determinado. En este curso queremos que no solo se queden con los conceptos sino que también los lleven a la práctica para reforzar el aprendizaje de la lógica de la programación. Elegimos JavaScript como su primer lenguaje de programación porque es popular, relevante, flexible y no se necesita instalar nada para ejecutarlo, por lo que se puede utilizar en cualquier sistema operativo.

Escribiendo código fuente en JavaScript

JavaScript surgió por la necesidad de ampliar las posibilidades de HTML. Por lo que tenemos que crear un archivo HTML que invoque nuestro programa en JavaScript.

Para crear este archivo abrimos el editor de texto (bloc de notas, TextEdit, etc.) y copiamos lo siguiente:

```
<html>

 <head>

  <title>Ejercicio</title>

  <script src="ejercicio.js"></script>
```

```
</head>

</html>
```

Esto indica que nuestro archivo de JavaScript se llamará ejercicio.js y debe estar en el mismo directorio que el archivo HTML.

Debemos guardar este archivo como ejercicio.html, hay que verificar que el editor de texto lo haya guardado con la extensión html para que se pueda ejecutar correctamente. Estos archivos los pueden descargar del Material de Apoyo.

Para crear el programa en JavaScript, abrimos el editor de texto y escribimos nuestro código fuente. Por ejemplo:

```
alert("Hola");
```

Debemos guardar este archivo como ejercicio.js

Ahora solo nos queda ejecutar el programa abriendo en nuestro navegador el archivo ejercicio.html

Comentarios

Los comentarios son textos que nos ayudan a describir lo que hace un fragmento del programa y son ignorados al ejecutarse un programa. En JavaScript hay dos tipos de comentarios.

Comentarios pequeños: Cualquier texto entre // y el final de la línea.

Comentarios grandes: Cualquier texto entre /* y */ pueden ocupar múltiples líneas.

Ejemplos:

```
// Comentario pequeño

/* Comentario grande */

/* Comentario

   muy

   grande */
```

3. Conceptos básicos

Los datos que se van a procesar deben almacenarse en celdas o casillas de memoria para su posterior utilización. Estas celdas de memoria tienen un nombre o etiqueta que permite su identificación. El valor que se almacena en las celdas de memoria puede ser constante o variable.

3.1. Variables y Constantes

Variables

Son datos que se almacenan en celdas de memorias y pueden cambiar su valor durante la ejecución del programa.

- Identificación de variables

Un identificador es un nombre que describe concretamente el valor de la variable. Por ejemplo si el valor que vamos a guardar es un numero de teléfono no tiene sentido que el identificador por ejemplo sea *pato*, lo ideal es que el identificador sea *telefono*.

Cada lenguaje de programación tiene algunas reglas para los identificadores de las variables. Por ejemplo en JavaScript son las siguientes:

- Debe empezar con letras, $ o _
- Después del primer carácter puede contener letras, números $ o _
- Distingue entre mayúsculas y minúsculas

- Evitar palabras reservadas

Correcto	**Incorrecto**
totalCompras	totalCompras!
dosis	2is
_num_copias2	_num copias2

- *Palabras reservadas*

En todos los lenguajes de programación hay ciertas palabras clave que no pueden utilizarse como identificadores en los programas.

Por ejemplo en JavaScript no se puede utilizar ninguna de estas palabras claves como identificadores: break, case, catch, continue, debugger, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with

Hay otras palabras reservadas para el futuro que también hay que evitar: abstract, boolean, byte, char, class, const, debugger, double, enum, export, extends, final, float, goto, implements, import, int, interface, long, native, package, private, protected, public, short, static, super, synchronized, throws, transient, volatile

- Declaración de variables

Antes de utilizar una variable, hay que declararla. Por ejemplo en JavaScript las variables se declaran con la palabra clave **var.**

```
var nombre;

var total;
```

También se pueden declarar las variables con la misma palabra clave.

```
var nombre, total;
```

- Almacenamiento de valores

Para almacenar un valor en una variable, se asocia un identificador con un valor.

Ejemplo en JavaScript:

```
total = 5;
```

También se puede combinar la declaración con el almacenamiento de valores.

```
var asunto = “Hola”;

var color = “Rojo”, figura = “Circulo”;
```

Mientras no se almacene un valor en una variable declarada, su valor sera *undefined* (indefinido).

Ejemplos en JavaScript

Desplegando datos

Para desplegar los datos vamos a utilizar el comando alert, el cual mostrara una alerta con los datos o variables. Por ejemplo:

```
alert(“Hola mundo!”);
```

```
var total = 54;

alert(total);
```

Solicitando datos de entrada

Para solicitarle un dato al usuario vamos a utilizar el comando prompt. De una vez vamos a guardar el dato en una variable para poder utilizarlo dentro del programa. Por ejemplo:

```
var email = prompt ("Ingrese su email");

alert(email);
```

Constantes

Son datos que se almacenan en celdas de memorias, pero no cambian su valor durante la ejecución del programa.

- En JavaScript funciona de manera similar que las variables pero se diferencia a la hora de declarar las constantes. Las constantes se declaran con la palabra clave **const** y de una vez se debe almacenar su valor**.**

```
const pi = 3.1416;
```

3.2. Tipos de datos

Un tipo de dato es una clasificación que define los posibles valores y las operaciones que se pueden realizar. Los tipos de datos simples son los siguientes:

Numéricos

Un tipo de datos numérico representa un subconjunto finito de números. Algunos lenguajes de programación hacen distinción entre números enteros y números decimales. Para cada lenguaje de programación existe un rango de los números que la computadora puede representar, algunos soportan números negativos otros no.

JavaScript no hace distinción entre enteros y decimales. Puede representar todos los números enteros entre −9007199254740992 (−2^53) y 9007199254740992 (2^53). Cuando un número aparece directamente en JavaScript se le denomina literal numérico. Ejemplo:

```
1;

2.20;

340232;
```

Para escribir un número negativo se le antepone al número un guión. Ejemplo:

```
-2;

-423;
```

Los números decimales también pueden utilizar notación científica. Se debe escribir el número seguido de la letra e y el exponente. Ejemplo:

```
3.45e12; // 3.45 x 10^12

2.38E-5; // 2.38 x 10^-5
```

Ejemplo almacenamiento de valor tipo numérico en variable.

```
var precio = 4.34;
```

Operaciones numéricas

Se pueden realizar las típicas operaciones aritméticas: suma, resta, multiplicación y división. Dependiendo del lenguaje de programación se pueden realizar otro tipo de operaciones aritméticas avanzadas que veremos más adelante. Las operaciones aritméticas también se pueden realizar con variables que almacenan datos del tipo numérico.

Operaciones aritméticas en JavaScript

Operación aritmética	Operador	Ejemplo
Suma	+	3 + 4
Resta	-	10 - 5
Multiplicación	*	9 * 5
División	/	27 / 3
Módulo	%	5 % 2

Precedencia en operaciones numéricas

La precedencia de operaciones es un conjunto de reglas que determina en qué orden se van a ejecutar las operaciones cuando se evalúa una expresión. Los operadores con mayor precedencia se van a ejecutar antes que las de menor precedencia.

Precedencia	Operador	Descripción
4	()	Operador de agrupamiento
3	*, /, %	Multiplicación, división y módulo
2	+, -	Suma y resta
1	=	Asignación

Ejemplo en JavaScript

```
var resultado = 34 * (27 / 3 + 2);
```

En la expresión anterior hay cinco operadores =, *, (), /, +

1. De primero se realiza la operación que está encerrada entre los paréntesis porque tiene mayor precedencia.
2. Luego dentro de los paréntesis la que tiene mayor precedencia es la división, el resultado de 27 / 3 es 9.
3. Luego se realiza la suma de 9 + 2 el resultado es 11.
4. Ya que se tiene el resultado del paréntesis se procede a multiplicar por 34 el resultado es 374.
5. Al final se almacena el resultado 374 en la variable resultado.

Operaciones numéricas complejas

Cada lenguaje de programación además de las operaciones aritméticas básicas soporta varias operaciones numéricas complejas. Por ejemplo: raíz cuadrada, logaritmos, etc.

Operaciones numéricas complejas en JavaScript

En JavaScript se utiliza la clase nativa Math. Les mostraremos algunos operadores complejos en la siguiente tabla.

Operación compleja	Operador	Ejemplo
Raíz cuadrada	Math.sqrt(x)	Math.sqrt(9);
Raíz cúbica	Math.cbrt(x)	Math.cbrt(8);
Potenciación	Math.pow(base,exponente) ;	Math.pow(4,3);
Valor absoluto	Math.abs(x);	Math.abs(-3);
Redondear hacia arriba	Math.ceil(x);	Math.ceil(3.4);
Redondear hacia abajo	Math.floor(x);	Math.floor(3.4);
Redondear: redondea al valor entero más cercano (si la parte decimal es menor a .5 se redondea	Math.round(x);	Math.round(3.4);

hacia atras, sino hacia adelante).		

Operaciones comparativas

Las operaciones comparativas son operadores en su mayoría binarios nos permiten comparar datos del tipo numérico devolviéndonos verdadero o falso según sea el caso. Las operaciones comparativas también se pueden realizar con variables que almacenan datos del tipo numérico.

Operaciones comparativas en JavaScript

Operación comparativa	Operador	Ejemplo

Igualdad	==	20 == 20 // Verdadero
		23 == 50 // Falso

No igualdad	!=	4 != 3 // Verdadero 86 != 86 // Falso
Mayor que	>	6 > 3 // Verdadero 3 > 6 // Falso
Mayor o igual que	>=	5 >= 5 // Verdadero 2 >= 8 // Falso

Menor que	<	3 < 9 // Verdadero 9 < 3 // Falso
Menor o igual que	<=	40 <= 56 // Verdadero 56 <= 40 // Falso

Error común: No hay que confundir = (sirve para asignar valores) con == (igualdad).

Lógicos

Un tipo de dato lógico o booleano representa verdadero o falso, encendido o apagado, sí o no. Hay sólo dos posibles valores de este tipo de dato.

En JavaScript utiliza las palabras reservadas **true** y **false** para verdadero y falso. También se puede utilizar 1 y 0 para verdadero y falso. Undefined también se considera falso.

```
var verdadero = true;

var falso = 0;
```

Los valores booleanos generalmente provienen del resultado de las operaciones comparativas como en el caso del tipo de datos numérico.

```
var verdadero = 5 >= 2;

var falso = 3 < 3;
```

Este tipo de dato es muy utilizado en las estructuras de control que veremos más adelante.

Operaciones lógicas (and, or, not)

Estos operadores comúnmente son utilizados con los operadores comparativos para cumplir múltiples condiciones.

- AND

  Funciona de la misma manera que la conjunción lógica. El resultado es verdadero si ambas expresiones son verdaderas.

- OR

  Funciona de la misma manera que la disyunción lógica. El resultado es verdadero si alguna expresión es verdadera.

- NOT

  Funciona de la misma manera que la negación. El resultado invierte el valor lógico.

Operaciones lógicas en JavaScript

Operación lógica	Operador	Ejemplo
and	&&	true && true // Verdadero true && false // Falso false && true // Falso false && false // Falso
or	||	true || true // Verdadero true || false // Verdadero false || true // Verdadero false || false // Falso
not	!	!true // Falso !false // Verdadero

Caracteres

El tipo de dato carácter se utiliza para representar un símbolo de un código predeterminado. Normalmente se utiliza el código ASCII el cual está formado por 128 caracteres en los que se incluyen letras, números, signos de puntuación, espacio y otros.

Cadena de caracteres

Una cadena de caracteres (string) es una secuencia ordenada de caracteres. En algunos lenguajes de programación es diferente la declaración de un carácter a la de una cadena de caracteres.

En **JavaScript** un literal de cadenas de caracteres o un carácter se declara entre comillas dobles o comillas simples.

```
var asunto = “Confirmacion de pedido”;

var nombre = ‘Maria Rodriguez’;
```

De modo que para desplegar una comilla simple se debe declarar entre comillas dobles y para desplegar una comilla doble se debe declarar entre comillas simples.

```
var referencia = ‘Se un uso un reportaje de “Prensa Libre”’;

var significado = “Niveo significa ‘de nieve o parecido a ella’”;
```

Secuencias de escape en strings

En algunos lenguajes de programación se utiliza la diagonal invertida (\) para representar caracteres que no se pueden representar de otra forma. Por ejemplo para representar una nueva línea se utiliza la secuencia de escape **\n** y para representar tabulación se utiliza la secuencia de escape \t.

Como en el ejemplo anterior de JavaScript para desplegar comillas dobles, si queremos obtener el mismo resultado podemos utilizar la secuencia de escape \" dentro de comillas dobles.

```
var referencia = "Se un uso un reportaje de \"Prensa Libre\"";
```

Funciona de igual manera para las comillas simples utilizando la secuencia de escape \' en el literal.

```
var significado = 'Niveo significa \'de nieve o parecido a ella\';
```

Concatenación

La operación concatenar sirve para unir varias cadenas de texto o caracteres.

En JavaScript con el tipo de dato cadenas de caracteres (string) se utiliza el símbolo + para la concatenación. Se pueden concatenar literales de cadenas de caracteres con variables de este tipo de dato. Por ejemplo:

```
var nombre = “Juan”;

var asunto = “Bienvenido ” + nombre;

var mensaje = “Estimado “ + nombre + “ Bienvenido al curso”;

var email = asunto + mensaje;
```

Otros tipos de datos

En los lenguajes de programación existen otros tipos de datos más complejos.

- Arreglos

  Es un tipo de dato que almacena una colección ordenada de valores. Según el lenguaje de programación estos valores pueden ser del mismo tipo de datos o de múltiples tipos de datos.

  En JavaScript se pueden almacenar valores de múltiples tipos de datos. Los valores del arreglo se escriben entre llaves separados entre comas. Por ejemplo:

```
var coloresPrimarios = [“Rojo”, “Amarillo”, “Azul”];

var precios = [0.59, 2.10, 23, 5];
```

3.3. Estructuras de Control

Es una serie de declaraciones, instrucciones y condiciones necesarias para determinar el flujo dentro de un programa.

1. Condicionales

Para la toma de decisiones dentro de los programas es necesario utilizar instrucciones condicionales. En la mayoría de lenguajes de programación se tienen las siguientes instrucciones:

a. If

Es una instrucción utilizada para ejecutar una porción de código si y sólo si la condición especificada es verdadera. Ejemplo:

```
var edad = 18;

if (edad >= 18 ) {

  alert("Es mayor de edad");

}
```

El ejemplo anterior debería mostrar el mensaje "Es mayor de edad"; en cambio si le colocan al valor de la variable edad un número menor a 18, este no va a hacer nada.

b. If … else

Es una instrucción utilizada para ejecutar una porción de código si la condición especificada es verdadera, pero también cuenta con otra instrucción para el caso donde la condición es falsa. Ejemplo:

```
var edad = 18;

if (edad >= 18 ) {

   alert("Es mayor de edad");

} else {

   alert("Es menor de edad");

}
```

El ejemplo anterior debería mostrar el mensaje "Es mayor de edad"; en cambio si le colocan al valor de la variable edad un número menor a 18, entonces debería mostrar el mensaje "Es menor de edad".

c. If … else if … else

La instrucción "else if" es utilizada para ejecutar una porción de código en caso de que la primera condición sea falsa y se quiera validar una o más condiciones. Ejemplo:

```
var num = 18;

if (num < 10 ) {

   alert("Es menor a 10");
```

```
} else if (num >= 10 && num < 20) {
   alert("Es mayor o igual a 10 y menor a 20");
} else if (num  >= 20 && num < 50) {
   alert("Es mayor o igual a 20 y menor a 50");
} else {
   alert("Es mayor o igual a 50");
}
```

El ejemplo anterior debería mostrar el mensaje “Es mayor o igual a 10 y menor a 20”; y dependiendo del valor que tenga la variable “num”, ese mensaje va a mostrar.

d. Switch case

Esta instrucción es utilizada cuando se tienen varios bloques de código y se quiere escoger entre uno de ellos dependiendo del valor que tenga una variable en específico. Ejemplo:

```
var mes = 5;
switch(mes) {
case 1:
  alert("Enero");
```

```
    break;
case 2:
    alert("Febrero");
    break;
case 3:
    alert("Marzo");
    break;
case 4:
    alert("Abril");
    break;
case 5:
    alert("Mayo");
    break;
case 6:
    alert("Junio");
    break;
```

```
case 7:
  alert("Julio");
  break;
case 8:
  alert("Agosto");
  break;
case 9:
  alert("Septiembre");
  break;
case 10:
  alert("Octubre");
  break;
case 11:
  alert("Noviembre");
  break;
case 12:
```

```
  alert("Diciembre");

  break;

default:

  alert("No es un mes válido");

  break;

}
```

La instrucción “break” que se muestra en el ejemplo anterior se utiliza para detener la ejecución de un bloque de código y continuar con la siguiente estructura, bucle, ciclo o código del programa.

Si en el ejemplo anterior no hubiéramos colocado la instrucción break al finalizar el bloque de código, se hubiera ejecutado cada uno de los bloques de código de las distintas opciones que se tenían disponibles en el switch.

2. **Iterativas**

Se utilizan para repetir un bloque de instrucciones N veces, donde N >= 0. Para efectos de ejemplo, utilizaremos las dos estructuras de control iterativas que tiene JavaScript:

a. For

Ejecuta una porción de código un número de veces.

Ejemplo:

```
var num = 10;

var resultado = 0;

for (var i=1; i<=num; i++)

{

  resultado += i;

}

alert(resultado);
```

El ejemplo anterior realiza la suma de los números de 1 a 10. El valor de la variable resultado varía dependiendo del valor de la variable i de la siguiente forma:

cuando i es 1, resultado = 1;

cuando i es 2, resultado = 3;

cuando i es 3, resultado = 6;

cuando i es 4, resultado = 10;

cuando i es 5, resultado = 15;

cuando i es 6, resultado = 21;

cuando i es 7, resultado = 28;

cuando i es 8, resultado = 36;

cuando i es 9, resultado = 45;

cuando i es 10, resultado = 55;

b. While

Ejecuta una porción de código si una condición se cumple.

Ejemplo:

```
var num = 10;

var resultado = 0;

var i = 1;

while (i <= num) {

  resultado += i;

  i++;

}

alert(resultado);
```

El ejemplo anterior tiene la misma funcionalidad que el ejemplo realizado para "*for*", pero adaptado a estructura de "*while*".

En este ejemplo el valor de resultado va cambiando al igual que en el ejemplo de "*for*", pero aqui se hizo la definición de la variable "*i*" fuera del bucle; además se hizo una operación más dentro del bloque de código: "*i*++", la cual le va sumando 1 a la variable "*i*".

Referencias

- http://www.play-hookey.com/computers/language_levels.html
- http://www.bottomupcs.com/chapter06.html
- http://oreilly.com/JavaScript/excerpts/learning-JavaScript/JavaScript-datatypes-variables.html
- http://www.w3schools.com/js/
- John C. Mitchell. Concepts In Programming Languages, Cambridge University Press, 2003.

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educacionales del curso.