Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Vacaciones de Junio de 2022
Catedrático: Ing. Mario Bautista
Auxiliar: Moises Gonzalez Fuentes

LFScript

Fase 1

Objetivos

Objetivo general

Implementar una aplicación donde se apliquen los conceptos adquiridos en el curso de Organización de Lenguajes y Compiladores 1.

Objetivos específicos

- Aplicar la fase de análisis léxico, sintáctico y semántico para la implementación de un intérprete.
- Construir un árbol de sintaxis abstracta (AST) para la ejecución de código de alto nivel.
- Conocer el patrón de diseño, intérprete.
- Detectar y reportar errores léxicos, sintácticos, semánticos y de ejecución.

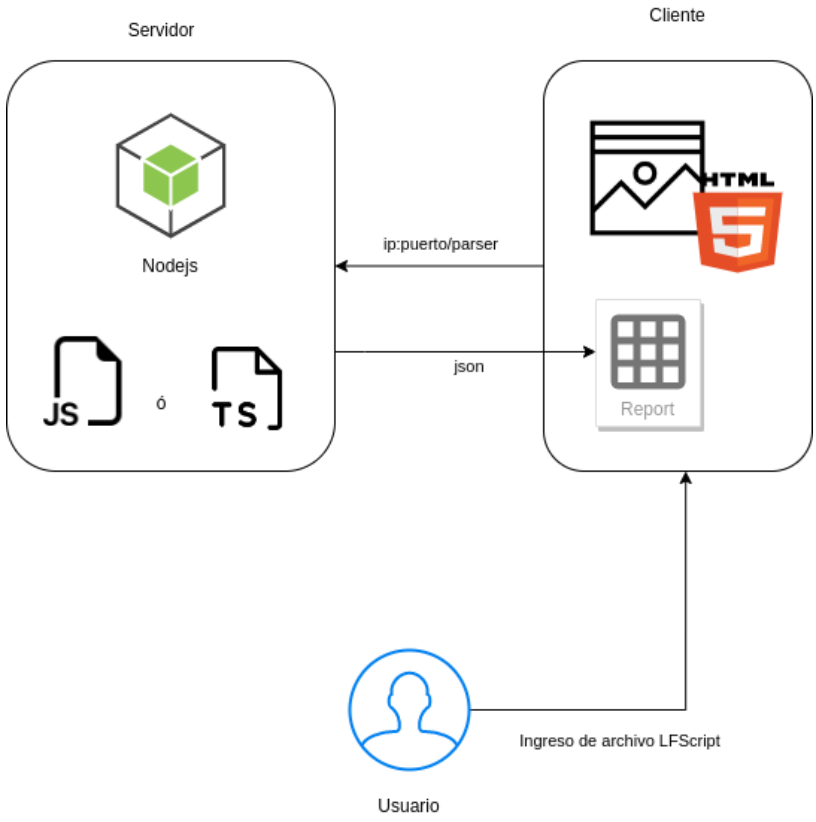
Descripción de la solución

El estudiante tiene que implementar un *intérprete* que ejecute instrucciones de alto nivel definidas en el lenguaje **LFScript**, el cual se definirá más adelante. Para la implementación se tiene que crear un *AST*, el cual servirá para la ejecución de las instrucciones del lenguaje.

Por medio de un servidor, se procesarán las solicitudes hechas al intérprete. Y las salidas serán visibles en una página web.

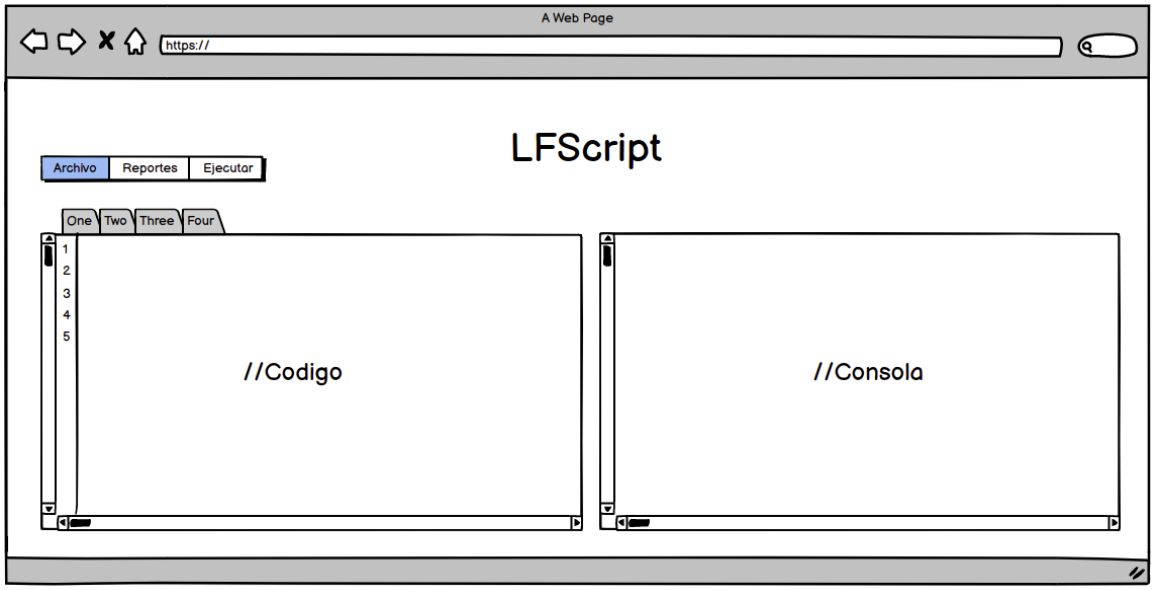
Flujo de la aplicación

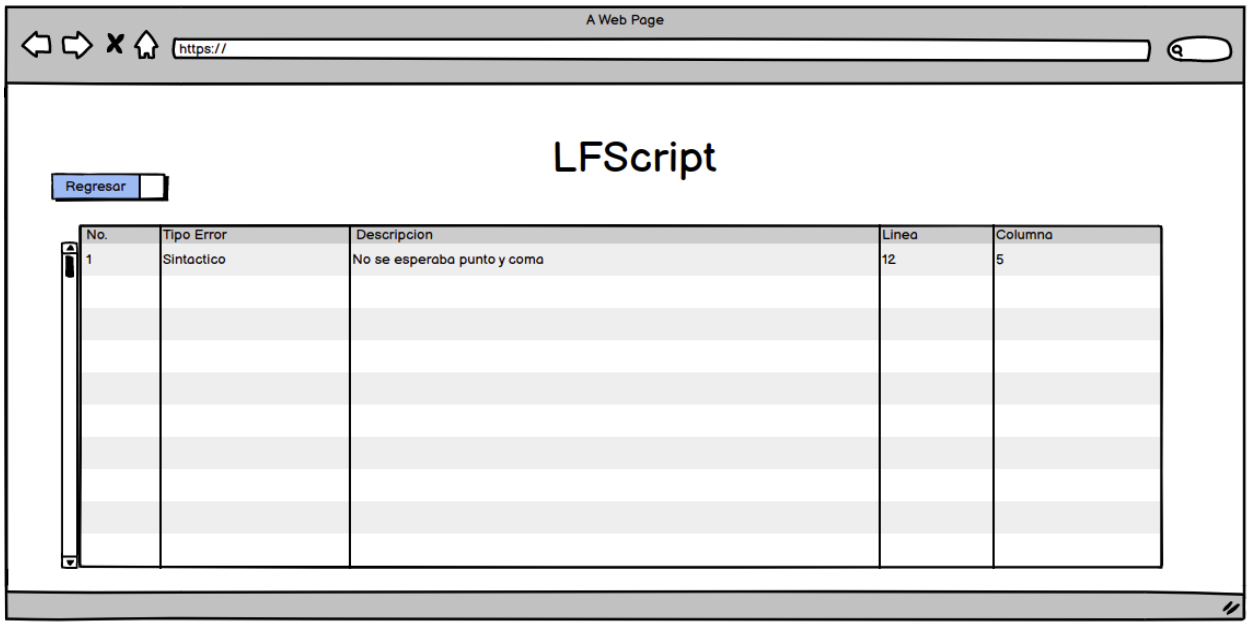
- El usuario abre la página web e ingresa el código fuente. O puede cargar el archivo con extensión **[.LF]**
- La página web hace la solicitud al servidor.
- El servidor analiza la solicitud y realiza el análisis léxico, sintáctico y semántico.
- El servidor retorna la información por medio de un JSON.
- El usuario observa las salidas de una forma amigable.



Interfaz gráfica

La aplicación deberá de contar con una interfaz gráfica amigable, la cual permitirá interactuar con el intérprete, manipular archivos de entrada y generar reportes, cada uno de sus elementos se describe a continuación:





Características del editor

- Tendrá múltiples pestañas.
- Debe de mostrar la fila y columna actual del cursor dentro de una pestaña.

Menú de archivos

- Abrir: permitirá cargar el contenido de un archivo a una pestaña dentro del editor.
- Guardar: guardará en un archivo la cadena de texto que se encuentre en la pestaña actual.
- Guardar como: guardará en un nuevo archivo la cadena de texto que se encuentre en la pestaña actual.
- Crear: creará una nueva pestaña dentro del editor, la pestaña no tendrá contenido.

Menú de ejecutar

- Ejecutar: se tomará la cadena que se encuentre en la pestaña actual y se hará la solicitud al servidor para que realice el proceso de interpretación.

Menú de reportes

- Reporte AST: en este reporte se visualiza una imagen con el AST que se genera al analizar la cadena de entrada de la pestaña actual con el código **LFScript**.
- Reporte de errores: en este reporte se mostrarán todos los errores que ocurran durante la interpretación. El reporte debe de ser generado en un archivo HTML en forma de tabla.
- Reporte TS: en este reporte se mostrarán todas las tablas de símbolos que se generaron con el código **LFScript.**

Área de salida

El editor debe de contar con una consola en la cual se mostrará la salida del código **LFScript**.

Definición del lenguaje

Archivos LFScript

Los archivos con código LFScript tendrá extensión **[.LF]**, el contenido de cada archivo se define a continuación:

- Puede venir la definición de una o más clases dentro del archivo.
- El código **LFScript** no será sensible a las mayúsculas.

Comentarios

Comentario de una línea

Los comentarios de una línea iniciarán con "//" y terminarán con un salto de línea.

```
1
2
3   //este es un comentario de una linea
4
```

Comentarios de varias líneas

Los comentarios de varias líneas iniciarán con "/*" y terminarán con "*/".

```
3   /*******************
4   ******** olc1 ******
5   *******************/
6
```

Tipos de datos

El lenguaje LFScript es un lenguaje con tipado, esto quiere decir que se conoce el tipo de dato de una variable desde el momento en que fue declarada. También tiene un tipado estático, lo que indica que una variable o expresión puede tener solo un tipo durante toda la ejecución.

Tipos de datos primitivos

Tipo de dato	Ejemplo
int	100, 024, 1, - 450, -5
double	0.2455, -6.245, -4.0
char	'a', 'f', ' ', '%'
boolean	true, false
String	"OLC1", "", "20", "- 4.45", "%%%#", "luisa", "{cadena}"

Caracteres de escape

Los caracteres de escape son caracteres que se definen de forma especial dentro de una cadena de texto (String). A continuación, se definen los caracteres que se tienen que manejar.

Carácter de escape	Descripción
\"	Comillas dobles
****	Barra invertida
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal

Expresiones

Signos de agrupación

Los signos de agrupación se utilizarán para dar orden y cierta precedencia a las operaciones. Se utilizarán los paréntesis : (10 + 1) * 3

Literales

Los literales hacen referencia a las expresiones donde su tipo de dato es primitivo.

Literal	Ejemplo
Entero	100, - 42, 6, ...
Decimal	-0.5482, 5.1247, - 0.0, …
Booleano	true, false
Carácter	‘’, ‘(‘, “$’, …
Cadena	“Compiladores 1”, “”, “100”, “,#,”, …

Incremento

Esta expresión devuelve el valor actual de la variable y posteriormente aumenta en uno su valor. Esto es exclusivamente para identificadores. Es utilizado como expresión e instrucción.

```
int a = 100;
a++;
double b = a++ * 4;
```

```
let x= ++z;
```

Decremento

Esta expresión devuelve el valor actual de la variable y posteriormente disminuye en uno su valor. Esto es exclusivamente para identificadores. Es utilizado como expresión e instrucción.

```
int a = 100;
a--;
double b = a-- * 4;
```

```
let x= --z;
```

Expresiones aritméticas

A continuación, se presenta el sistema de tipos de las operaciones aritméticas. Para los casos donde se opere con un tipo de dato **char** se tomará el valor del código ASCII del carácter.

Suma

El operador será el signo más [+]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
~~int~~	~~int~~	~~int~~
~~int~~	~~double~~	~~double~~
~~int~~	~~char~~	~~int~~
~~int~~	~~String~~	~~String~~
~~double~~	~~int~~	~~double~~
~~double~~	~~double~~	~~double~~
~~double~~	~~char~~	~~double~~
~~double~~	~~String~~	~~String~~

char	int	int
char	double	double
char	char	int
char	String	String
String	int	String
String	double	String
String	char	String
String	String	String
String	Boolean	String

Resta

El operador será el signo menos [-]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double
int	char	int
double	int	double
double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

Multiplicación

El operador será el signo [*]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double
int	char	int
double	int	double
double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

División

El operador será el signo[/]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double

int	char	int
double	int	double

double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

Potencia

El operador será el signo [**]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	double
int	double	double
int	char	double
double	int	double
double	double	double
double	char	double
char	int	double
char	double	double
char	char	double

Modulo

El operador será el signo [%]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	double
int	double	double
int	char	double
double	int	double
double	double	double
double	char	double
char	int	double
char	double	double
char	char	double

Expresiones relacionales

A continuación, se presenta el sistema de tipos de las operaciones relacionales. El resultado de estas operaciones será un valor de tipo booleano (true o false).

Mayor que, Menor que, Mayor o igual que, Menor o igual que

Los operadores de estas operaciones serán los siguiente [>, <, >=, <=]. A continuación, se presenta la tabla de tipos de esta operación

Operando 1	Operando 2
int	int
int	double
int	char
double	int
double	double
double	char
char	int
char	double
char	char

Igual que, diferente que

Los operadores de estas operaciones serán los siguientes [==, !=]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2
int	int
int	double
int	char
double	int
double	double
double	char
char	int
char	double
char	char
String	String
Boolean	Boolean

Expresiones lógicas

A continuación, se presenta la tabla de verdad de las operaciones lógicas. El resultado de estas operaciones será un valor de tipo booleano (true o false). Los operadores que reciben estas expresiones tienen que ser de tipo booleano.

Or

El operador será definido por los caracteres [||]. Cabe mencionar que esta operación se realiza en modo de **corto circuito**, lo que quiere decir que sí el primer operando es verdadero ya no se evalúa el segundo operando porque ya se sabe que la condición es verdadera.

And

El operador será definido por los caracteres [&&]. Cabe mencionar que esta operación se realiza en modo de **corto circuito**, lo que quiere decir que sí el primer operando es falso ya no se evalúa el segundo operando porque ya se sabe que la condición es falsa.

Xor

El operador será definido por los caracteres [^].

Not

El operador será definido por los caracteres [!].

X	Y	X || Y	X && Y	X ^ Y	!X
false	**false**	false	false	false	true
false	**true**	true	false	true	true
true	**false**	true	false	true	false
true	**true**	true	true	false	false

Declaración de variables

Esta instrucción permitirá guardar en tabla de símbolos una variable con su respectivo valor. El valor de esta variable puede cambiar durante la ejecución del código. El identificador debe iniciar con una letra y después aceptar más letras o guión bajo.

```
int contador = 100;
Nodo raiz= new Nodo(100);
String nombre = "compi"+"1";
```

```
double x,y,x= 12.12;
```

Declaración de variables constantes

Esta instrucción permitirá guardar en tabla de símbolos una variable con su respectivo valor. La diferencia es que el valor de esta variable nunca cambiará de valor durante la ejecución del código. Para diferenciarlo se usará la palabra reservada "**Const**". La condición para el identificador es el mismo que el anterior.

```
const int contador = 100;
const String nombre = "compi"+"1";
```

```
const double x,y,x= 12.12;
```

Asignación de variables

Esta instrucción permitirá modificar el valor de una variable que fue previamente declarada. Cuando se usa una variable "const" retorna un error semántico, ya que esa variable no puede modificarse.

```
int contador= suma(a,b);
contador= contador--;
double x= 12.13;
x=99.99+1-9;
```

Sentencias de control

Sentencia IF

Esta instrucción contendrá una lista de condiciones las cuales serán evaluadas para ver qué condición es la que se cumple, de cumplirse con una condición se ejecutará el bloque de instrucciones de la condición (existe una forma resumida donde únicamente ejecuta una instrucción). En esta instrucción se puede definir un caso que se ejecute cuando ninguna de

las condiciones se cumple, este caso es el **ELSE**. A continuación, un ejemplo de su uso.

```
if(a==10 && true){
    //instrucciones
}else if(a==11 && true){
    //instrucciones
}else if(a==12 && true){
    //instrucciones
}else{
    //instrucciones
}
```

```
if(raiz!= null){
    //instruccion
}
```

```
if (a==20) //una instrucciones ;

if (a==20) //una instruccion ;
else if (a==21) //una instruccion ;
else //una instruccion ;
```

Sentencia SWITCH

Esta instrucción recibirá una expresión de control y una lista de casos (opciones). Esta sentencia comparará el valor de la expresión de control con el valor del caso, si estos valores coinciden se ejecuta el bloque de instrucciones de ese caso. Si dentro del caso que se ejecuta no se encuentra una instrucción break se ejecutarán el resto de casos que se encuentren debajo sin realizar la comparación entre la expresión de control y el valor del caso, sí se definiera un caso **default** y este está debajo del caso que se cumple también se ejecuta.

```
swith(num){
    case 1:
        //instrucciones;
        break;
    case 2:
        //instrucciones;
    case 3:
        //instrucciones;
        break;
    default:
        //instrucciones;
        break;
    case 5:
        //instrucciones;
        break;
}
```

Dentro de esta sentencia también se puede definir un caso por defecto, el cuál será ejecutado cuando ninguno de los casos coincida con el valor de la expresión de control. Este caso por defecto puede venir en cualquier parte de la instrucción Switch, no necesariamente al final.

Sentencia FOR

Esta sentencia estará conformada por lo siguiente:

- Inicialización: se podrá declarar cualquier tipo de variable o realizar una **asignación**.
- Condición: será la expresión que se verificará en cada iteración para decidir si se ejecuta el bloque de instrucciones o no.
- Actualización: esta parte permite actualizar la variable de control del ciclo. Se puede definir una asignación de variable, incremento o decremento.

```
int y=10;

for(int i=0; i<10; i++){
    //instruccioines
}

for(int i=0; i<10; i=i+1){
    //instruccioines
}

for(y=0; i<10; y=y+1){
    //instruccioines
}
```

Sentencia WHILE

Esta instrucción ejecutará un conjunto de instrucciones mientras el resultado de su expresión de control sea verdadero.

```
while(true){
    //intrucciones
}
```

```
while( a!=56 || b<15 ){
    //intrucciones
}
```

Sentencia DO-WHILE

Esta instrucción al igual que el WHILE ejecutará un conjunto de instrucciones mientras el

resultado de su expresión de control sea verdadero, la diferencia es que esta instrucción ejecuta su bloque de instrucciones una vez sin evaluar la condición, la condición se empieza a evaluar a partir de la segunda iteración.

```
do{
    //intrucciones
}while(true);
```

```
do{
    //intrucciones
}while( a!=56 || b<15 );
```

Sentencia BREAK

Esta instrucción permite alterar el flujo de ejecución del programa, la sentencia BREAK tendrá los siguientes usos:

- Terminar la ejecución del ciclo más cercano(While, Do-While).
- Terminar la ejecución de una instrucción SWITCH.

```
do{
    //intrucciones
    if (a=="olc"){
        break;
        //codigo inaccesible
    }
}while(true);
```

Nota: se debe de verificar que esta instrucción se encuentre dentro de un ciclo o dentro de un SWITCH, de lo contrario sería un error semántico.

Sentencia CONTINUE

Esta instrucción permite alterar el flujo de ejecución del programa, la sentencia CONTINUE tendrá los siguientes usos:

- Pasar a la siguiente iteración del ciclo más cercano(For).

```
for(int i=0; i<10; i++){
    //instruccioines
    continue;
    //instruccioines no accesible
}
```

```
for(int i=0; i<10; i++){
    //instruccioines
    if(...){
        continue;
    }
    //instruccioines no accesible
}
```

Nota: se debe de verificar que esta instrucción se encuentre dentro de un ciclo FOR, de lo contrario sería un error semántico.

Métodos y funciones

Método

Se puede definir a un método como un conjunto de instrucciones dentro de un bloque al cual se le define un nombre y el cual puede ser invocado desde otra parte del programa. Los métodos no retornan valores al momento de invocarlos como lo hacen las funciones.

```
void metodo1(int a, string b){
    //instrucciones
}

void metodo2(){
    //instrucciones
}
```

Función

Se puede definir a una función como un conjunto de instrucciones dentro de un bloque al cual se le define un nombre y la cual puede ser invocada desde otra parte del programa. A

diferencia de los métodos las funciones siempre devuelven un valor al momento de invocarlas. Si no tiene una sentencia de retorno con expresión, será un error semántico.

```
string metodo1(int a, string b){
    //instrucciones
    return ...
}

int metodo2(){
    //instrucciones
    return ...
}
```

Parámetros de un método o función

Los parámetros son variables que se definen para un método o función, al realizar una llamada se deben de enviar valores a los parámetros definidos. Cabe resaltar que una función puede o no tener parámetros.

Llamada a métodos y funciones

Una llamada es una invocación a ejecutar las instrucciones definidas dentro de un método o función. Se usará la palabra reservada "**call**" cuando sea una Instrucción. Cuando se retoma como una expresión se omite la palabra reservada.

```
call metodo1();
call metodo2(1,"olc");
call metodo3(1,"olc",null);
```

Sentencia return

Esta instrucción permite alterar el flujo de ejecución del programa, existen dos formas de utilizar esta instrucción:

- **return**: para terminar de ejecutar un método. Se debe de validar el posible error semántico donde la instrucción venga dentro de una función y no dentro de un método.
- **return <EXPRESIÓN>**: para terminar de ejecutar una función. Se debe de validar el posible error semántico donde la instrucción venga dentro de un método y no dentro de una función.

Otra condiciones de métodos

Permite definir métodos con el mismo nombre, pero que tiene diferente cantidad de parámetros o que tiene la misma cantidad, pero el tipo de los parámetros es diferente y lo diferenciarán de los demás. Los métodos pueden estar en **cualquier posición del código** y ser ejecutados, para ello se recomienda reconocerlos en la primera pasada.

```
call metodo1(2022);

void metodo1(int a){
    //instruccion
}

void metodo1(doble b){
    //instruccion
}

//mismo nombre pero los parametros lo diferencian
```

Bloque de instrucciones

Es un conjunto de instrucciones dentro de llaves y tiene su propia tabla de símbolos.

```
{
    //instrucciones
    {
        //instrucciones
    }
    //instrucciones

}
```

Funciones nativas

Estas son funciones propias del lenguaje LFScript.

Println

Esta función recibe de parámetro una expresión e imprime en la consola de la aplicación el resultado, al final del valor de la expresión se agrega un salto de línea. Puede tener o no una expresión a imprimir.

```
println(metodo1(1,2));
println(1+1);
println(age); //variable
println();
```

Print

Esta función recibe de parámetro una expresión e imprime en la consola de la aplicación el resultado, a diferencia de **Println** esta NO agrega un salto de línea al final del valor de la expresión.

```
print(metodo1(1,2));
print(1+1);
print(age); //variable
print();
```

Typeof

Retorna el tipo de dato de una variable, o una expresión. Como una cadena de caracteres(string), a continuación un ejemplo.

```
string tipo_dato = typeof("hola");
println(tipo_dato); //string

int x=0;
string tipo_dato = typeof(x);

println(typeof(x)); //int
```

Precedencia y asociatividad de operadores

La precedencia dentro de la tabla está definida de menor a mayor.

Nivel de precedencia	Operador	Asociatividad
1	||	Izquierda
2	&&	Izquierda
3	^	Izquierda
4	>, <, >=, <=, ==, !=	No asociativo
5	+, -	Izquierda
6	*, /, %	Izquierda
7	!	Derecha

Nota: de tener algún problema con la precedencia definida se adjunta un enlace donde pueden consultar otra tabla de precedencia y asociatividad:
https://introcs.cs.princeton.edu/java/11precedence/

Restricciones del proyecto

- La aplicación deberá de desarrollarse utilizando el lenguaje **JavaScript o TS**.
- Se debe respetar la arquitectura del proyecto (Flujo del proyecto).
- Las herramientas para realizar los analizadores será **Jison**.
- El proyecto se realizará de manera **individual** si se detecta algún tipo de copia el laboratorio quedará anulado.
- La calificación será sobre la interfaz gráfica.
- Utilizar controlador de versiones Git.

Forma de entrega del proyecto

Entregables

- Código fuente de la aplicación.
- Archivo con la gramática utilizada en el proyecto.
- Link de repositorio remoto **privado** GitHub [OLC1-F1-carné]

Fecha de entrega

Sábado 18 de junio de 2022, hasta las 23:59