Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2020
Ing. Mario Bautista, Ing. Manuel Castillo, Ing. Kevin Lajpop
Auxiliares: Luis Mazariegos, Huriel Gómez, Fernando Paz, José Puac

Translator in Docker

Objetivos

General

1. Aplicar los conocimientos aprendidos sobre el analizador léxico y sintáctico vistos en clase y laboratorio, para la implementación de un traductor como servicio.

Específico

1. Creación de un traductor utilizando conocimientos de programación básicos, sin el uso de herramientas.
2. Creación de un traductor utilizando herramientas para análisis léxico y sintáctico.
3. Se logre comprender el analizador tanto a mano como con el uso de herramientas.

Descripción

Realizar la traducción de un lenguaje a otro puede ser algo tedioso sobre todo cuando estamos trabajando sobre sistemas legacy y con los cuales queremos replicar la funcionalidad en otro lenguaje de programación que es más reciente o popular y del cual encontramos mayor información y soporte técnico.

Para darle solución tenemos aquellos transpiradores que se dedican a traducir de un lenguaje de programación a otro con el mismo nivel de dificultad para su análisis. Pero estos están destinados a obtener un lenguaje en específico y traducirlo a únicamente otro de salida.

Como estudiante de la carrera de ciencias y sistemas se le pide que implemente la solución para crear un traductor de lenguajes en el cual tendrá como entrada un programa en un lenguaje fuente del cual tendrá que dar como resultado el equivalente al programa pero en 2 lenguajes distintos. La idea central de esto es que al realizar la traducción podamos dar más de una opción como salida y que el usuario final pueda elegir si desea descargar únicamente una de las 2 opciones o incluso ambas según su necesidad.

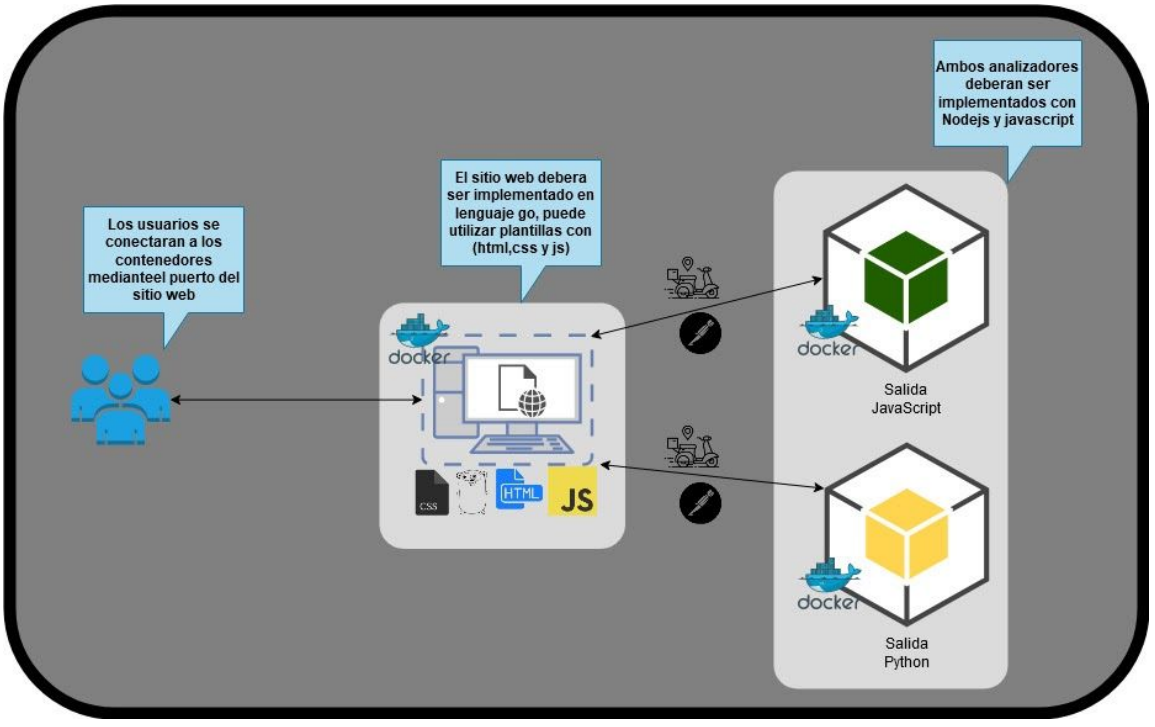
Para entrar en contexto debemos definir qué lenguaje será el permitido para realizar el análisis y cuáles son los lenguajes de salida. Para ello definimos la siguiente tabla:

Lenguaje	Funcion
Java	Lenguaje fuente, este lenguaje definirá la sintaxis de los archivos de entrada que deberá analizar tanto con un analizador léxico como sintáctico.
JavaScript	Lenguaje objeto, este lenguaje será una de las dos posibilidades de traducción, esta será una de las salidas esperadas como equivalente a java.
Python	Lenguaje objeto, este lenguaje será la otra opción de traducción de igual manera es una salida que se espera sea equivalente a java.

Nota: la sintaxis permitida para cada uno de los lenguajes se detalla a continuación, la forma de la traducción será de manera respectiva, esto indica que se definirá la sintaxis de java para un conjunto de sentencias y estas de igual manera serán detalladas para los lenguajes de salida.

Por ejemplo, si en java se define la sintaxis de un ciclo for, también se definirá la sintaxis del ciclo for en javascript y python. Y esta será la guía a tomar en cuenta para el proceso de traducción.

Flujo del programa



Tomar en cuenta lo que se establece en el flujo del programa, se deberá implementar 3 contenedores para simular que están en distintas computadoras. Los lenguajes a analizar son los establecidos anteriormente y los lenguajes donde deberán programar son los que se detallan en el flujo de trabajo y al final del enunciado.

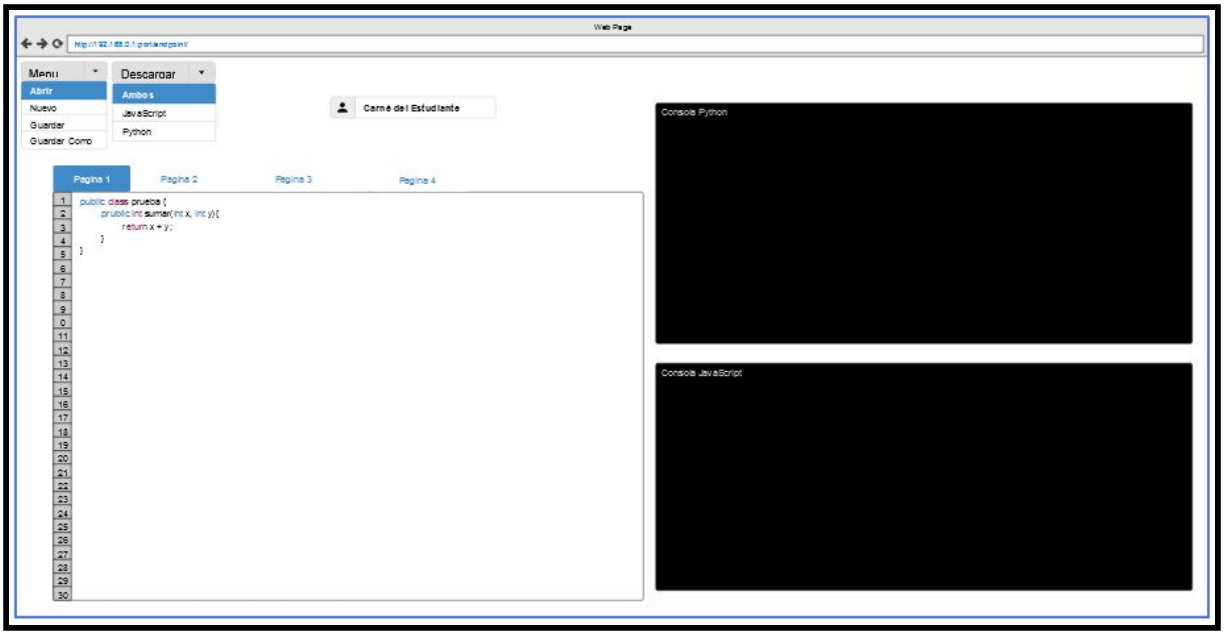
Interfaz Gráfica

La interfaz gráfica solo aplica para el sitio web, esta será realizada mediante plantillas html las cuales serán levantadas mediante el lenguaje únicamente, no se permite el uso de otro servidor http como apache o nginx para hacer público el sitio desde el contenedor.

Queda a criterio del estudiante el puerto a utilizar para hacer público el sitio.

La interfaz debe contar con los siguiente:

1. Un menú en el cual se muestre la opción de cargar un archivo de entrada el cual funcionara como el lenguaje de entrada, este archivo será un java.
2. Deberá contar con un botón para guardar el cual descargara el archivo actual.
3. Listado de pestañas para tener más de un documento visible en el editor.
4. Pintar palabras reservadas, se permite el uso de herramientas para realizar dicha acción.
5. Dos consolas de salida una para cada uno de los traductores, en el cual marcará los errores obtenidos del archivo de entrada.
6. Al terminar de analizar y de traducir el archivo de entrada se descargan los archivos de salida por medio de un botón en el cual se podrá elegir descargar uno en específico o ambos.
7. Las dos consolas tendrán como restricción que no sean editables, para evitar cambios en las salidas de los errores.
8. Dentro de la interfaz deberá ser visible el número de carné del estudiante.



Sentencias permitidas

Todas las sentencias permitidas dentro de los 3 lenguajes serán:

- Clases
- Interfaces
- Métodos y funciones (Parámetros)
- Sentencias de repetición
    1. For
    2. While
    3. Do while
- Sentencias de control
    1. If
- Sentencias break, continue, return
- Expresiones
    1. Lógicas (and,or,not,xor)
    2. Relacionales (igual, mayor, menor, mayor igual, menor igual, distinto)
    3. Aritméticas (suma, resta, multiplicación, división, adición ++, sustracción --, negativos)
    4. Paréntesis
- Tipos de datos
    1. Entero
    2. Booleano
    3. Flotante
    4. String
    5. Carácter
- Comentarios
    1. Unilinea
    2. Multilínea
- Declaración y asignación de variables
- Llamadas a métodos
- Método y función por defecto
    1. Main
    2. Print

A continuación se detalla la sintaxis a utilizar tanto para el lenguaje de entrada como para los lenguajes de salida esperados.

Sintaxis del lenguaje Java

A continuación se detalla la sintaxis permitida para la definición e implementación de cada una de las sentencias permitidas dentro de java, así como un ejemplo de su uso.

- **Clases**

La definición de la sintaxis de una clase en java está dada por un modificador, en este caso solo será válido el modificador 'public' además del nombre identificador de la clase. Dentro de una clase es permitido declarar variables globales al igual que métodos y/o funciones.

public class IdentificadorClase { // Declaraciones de atributos y métodos }	public class ejemplo1 { // Todo tipo de variables globales y métodos // o funciones }

- **Interfaces**

Al igual que las clases una interfaz tiene el modificador 'public' y un identificador de la interfaz. Estas tendrán dentro de ellas solamente la definición de funciones, mas no la implementación.

public interface IdentificadorInterfaz { // Definición de funciones }	public interface interfaz1 { // Definición de funciones }

- **Métodos y funciones (Parámetros)**

La diferencia entre los métodos y funciones es el valor de retorno, dentro de la sintaxis en java podremos utilizar distintos tipos de datos para el retorno de las funciones, por lo cual tenemos la lista de tipos de datos que se detallan al final y el tipo void que indica que el método no devuelve un valor.

Debemos tomar en cuenta que una función puede ser declarada e implementada, así como simplemente declarada, el uso de las funciones o métodos que son declarados únicamente será dentro de las interfaces. Dentro de una clase debemos implementar todas las funciones.

public [TIPO / void] nombreFuncion (PARÁMETROS) { // Bloque de instrucciones }	public void suma(int x, int y){ // Bloque de instrucciones }
public [TIPO / void] nombreFuncion (PARÁMETROS);	public void suma(int x, int y);

TIPO: se refiere a los tipos de datos permitidos explicados a continuación.

PARÁMETROS: Es el listado de valores permitidos enviados únicamente por valor y no por referencia a la función o método. La sintaxis se detalla a continuación:

[TIPO id, TIPO id , ...]	int x, double y, char z, String s

- **Sentencias de repetición**

Las sentencias de repetición que se utilizaran dentro del lenguaje se limitan a las 3 descritas a continuación.

For:

El ciclo de repetición for está destinado a repetir un bloque de instrucciones 'n' veces, para ello es necesario conocer la cantidad 'n' que se ejecutará dicho bloque de sentencias. La sintaxis a utilizar es:

for(DEC; EXP; EXP) { // Bloque de Instrucciones }	for(int x = 0; x <= a+4; x++){ // Bloque de Instrucciones } for(double x = 20; x > 5*4; x --){ // Bloque de Instrucciones }

DEC: referencia a la sintaxis de una declaración.
EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

While:

El bloque de repetición while permite ejecutar una cantidad de veces un bloque de sentencias, este está limitado por medio de una condición la cual indica si se sigue ejecutando o se termina el bloque de ejecución.

while (EXP) { // Bloque de sentencias }	boolean prueba = true; while (prueba) { System.out.println("Esto lo verás una vez"); prueba = false; }

EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

Do while:

La sentencia de repetición do-while al igual que while ejecuta un bloque de sentencias mediante una condición de salida, la diferencia es que este se ejecuta al menos una vez. La sintaxis a utilizar dentro de java es la siguiente:

do { // Bloque de sentencias } while(EXP);	int contador = 0 ; do{ System.out.println ("Contador" + (contador + 1)); contador ++; } while (contador<10);

EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

- **Sentencias de control**

<u>If:</u>
La sentencia de control if denota una condición que define el camino de la ejecución, la sentencia de control if puede estar construida mediante el bloque then, else if y else, aunque no es obligatorio que este se componga de todos esos bloques siempre, tomando en cuenta lo anterior se define la sintaxis a utilizar dentro de java para la sentencia if:

if (EXP) { // Bloque de sentencias then }[else if (EXP){ // Bloque de sentencias else if } . . . . else{ // Bloque de sentencias else }]	if (a > 5){ // Bloque de sentencias then } else if (a < 5){ // Bloque de sentencias else if }else{ // Bloque de sentencias else }

Se subraya la parte opcional de un if teniendo en cuenta que puede componerse únicamente de una lista de else if o un else o incluso ambos. Tomar en cuenta que dentro de un if únicamente puede existir un else que lo acompañe.

- **Sentencias break, continue, return**

Las sentencias break, continue, return estas son utilizadas para generar saltos dentro de la ejecución y flujo del programa. La importancia de cada una radica en cuando es correcto su uso, por ejemplo, una sentencia break y continue únicamente la podemos utilizar dentro de un ciclo de repetición, si se usa pero no existe un ciclo de repetición como entorno padre este es un error, de igual manera una sentencia return se utiliza para terminar la ejecución de un metodo o funcion y el retorno dependerá del tipo definido en la función.

A continuación se detalla la sintaxis correcta para cada una de las sentencias:

break;	break;
continue;	continue;
return [EXP];	return; return 4+ 5; return “Cadena”; return var;

- **Expresiones**

La expresiones son todo tipo de operación a nivel aritmético, lógico y relacional, a continuación se detalla cada uno de los tipos de expresiones que serán válidas dentro del lenguaje y las posibles operaciones que se podrán realizar.

Lógicas (and,or,not,xor):

Estas retornan a un valor booleano y reciben como nodos hijos valores booleanos que son retornados de operaciones relacionales como comparaciones o de otras operaciones booleanas.

AND	&&	a && b	binario
OR	||	a || b	binario
NOT	!	! a	unario
XOR	^	a ^ b	binario

Tomar en cuenta que a y b representan expresiones.

Relacionales (igual, mayor, menor, mayor igual, menor igual, distinto):

Este tipo de operaciones realiza comparaciones que retornan un valor booleano, como nodos hijos tienen operaciones aritméticas o valores puntuales.

>	a > b
<	a < b
>=	a >= b
<=	a <= b
==	a == b
!=	a != b

Tomar en cuenta que a y b representan expresiones.

Aritméticas (suma, resta, multiplicación, división, adición ++, sustracción --, negativos):
La operaciones aritméticas válidas dentro del lenguaje java serán utilizadas dentro de las expresiones, estas deberán tener la siguiente sintaxis para ser válidas:

Suma	EXP + EXP	4 + a
Resta	EXP - EXP	4 - a
Multiplicación	EXP * EXP	c * 6
División	EXP / EXP	q / c c != 0
Adición ++	EXP ++	a ++
Sustracción --	EXP --	v --
Negativos -	-EXP	-7

Es importante tomar en cuenta la precedencia y asociatividad de las operaciones.

Paréntesis:
El uso de paréntesis marca la precedencia de los operadores al momento de realizar una operación, esto define de manera explícita la forma en la que se evaluarán dichas operaciones, no importando el tipo de operaciones.

(7 + 6) * 5	Primero se realizará la suma y luego la multiplicación. Aun sabiendo que la precedencia no va en ese sentido.

- **Tipos de datos**

Dentro de la sintaxis de java y el lenguaje se tendrán distintos tipos de datos estos son los definidos anteriormente como TIPO. Entre los cuales podemos encontrar:

Entero	int
Booleano	boolean (true, false)
Flotante	double
String	String
Carácter	char

- **Comentarios**

Dentro del lenguaje serán permitidos 2 tipos de comentarios, estos son unilinea y multilínea, para esto se define la sintaxis correcta de cada uno de ellos.

Unilinea	// Comentario unilinea que termina al encontrar un salto de línea
Multilínea	/* * * Este tipo de comentario * soporta más de una línea * */

- **Declaración y asignación de variables**

Dentro del lenguaje se podrán realizar declaraciones y asignaciones de variables a continuación se describen las sintaxis permitidas para cada acción:

Declaración:

Para la declaración de variables debemos tomar que puede ser una lista de valores del mismo tipo y estos pueden ser asignados o no.

TIPO [id = EXP / id , …];	int x=0, y, z=9; boolean ft = true; double c = 1.2;

Asignación:

La sintaxis para la asignación de variables en java será:

id = EXP;	x = true && false;

- **Llamadas a métodos**

Dentro del lenguaje java se podrán realizar llamadas a métodos o funciones, esto se realizará dentro de otros métodos o funciones y la sintaxis para las llamadas es por medio del nombre de la función y una lista de parámetros enviados por valor.

nombre_funcion([PARAMETROS]);	mifuncion(x, “cadena”, true);

- **Método y función por defecto**

Main:

El método main es el primero en ejecutarse dentro de un proyecto en java, este tiene una sintaxis definida la cual se define a continuación, tomar en cuenta que este siempre será

público y de tipo void, además de eso tiene la palabra reservada static que será una palabra reservada utilizada únicamente para este método.

```
public static void main(String[] args) {
        //Instrucciones
}
```

Print:

El lenguaje java define una sintaxis para el método print, esta se define a continuación y puede recibir expresiones como parámetro. Tomar en cuenta que el método print puede agregar un salto de línea al final o no esto dependerá de la sintaxis utilizada, ambos son válidos dentro del lenguaje.

System.out.println(EXP);	System.out.println(“texto con salto de línea”);
System.out.print(EXP);	System.out.print(“texto sin salto de línea”);

Sintaxis del lenguaje JavaScript

A continuación se detalla la sintaxis permitida para la definición e implementación de cada una de las sentencias permitidas dentro de javascript, así como un ejemplo de su uso.

- **Clases**

Utilice la palabra clave class para crear una clase y siempre agregue el constructor() método.

El método constructor se llama cada vez que se inicializa el objeto de clase.

class name [extends] { // Contenido de la clase }	class Car { constructor(brand) { this.carname = brand; } } mycar = new Car("Ford");

- **Interfaces**

NOTA: JavaScript no implementa Interfaces

- **Métodos y funciones (Parámetros)**

Funciones

Una función de JavaScript es un bloque de código diseñado para realizar una tarea en particular.

Una función de JavaScript se ejecuta cuando "algo" la invoca (la llama).

function *name*(*parameter1, parameter2, parameter3*) { *// code to be executed* }	var x = myFunction(4, 3); // Function is called, return value will end up in x function myFunction(a, b) { return a * b; // Function returns the product of a and b }
Una función de JavaScript se define con la functionpalabra clave, seguida de un nombre , seguido de paréntesis () . Los nombres de funciones pueden contener letras, dígitos, subrayados y signos de dólar (las mismas reglas que las variables).	Cuando JavaScript llega a una declaración return, la función dejará de ejecutarse. Si la función fue invocada desde una declaración, JavaScript "regresará" para ejecutar el código después de la declaración de invocación.

Los paréntesis pueden incluir nombres de parámetros separados por comas: (parámetro1, parámetro2, ...) El código a ejecutar, por la función, se coloca entre llaves: {} Los parámetros de la función se enumeran entre paréntesis () en la definición de la función. Los argumentos de la función son los valores que recibe la función cuando se invoca. Dentro de la función, los argumentos (los parámetros) se comportan como variables locales.	Las funciones a menudo calculan un valor de retorno

Métodos

Los métodos siguen la misma lógica que las propiedades, la diferencia es que son funciones y se definen como funciones. Llamar a un método es similar a acceder a una propiedad, pero se agrega () al final del nombre del método, posiblemente con argumentos.

En JavaScript los métodos son objetos como lo es una función normal y se vinculan a un objeto como lo hace una propiedad, lo que significa que se pueden invocar desde "fuera de su contexto"

Los métodos JavaScript son acciones que se pueden realizar en objetos. Un método de JavaScript es una propiedad que contiene una definición de función . En una definición de función, se thisrefiere al "propietario" de la función. *objectName.methodName()*	var person = { firstName: "John", lastName : "Doe", id : 5566, fullName : function() { return this.firstName + " " + this.lastName; } }; name = person.fullName();

- **Sentencias de repetición**

Las sentencias de repetición que se utilizaran dentro del lenguaje se limitan a las 3 descritas a continuación.

For:

Los ciclos pueden ejecutar un bloque de código varias veces.

for (instruccion 1; declaracion 2; instruccion 3) { // code block to be executed }	for (i = 0; i < 5; i++) { // code block to be executed } var i; for (i = 0; i < 5; i++) { // code block to be executed }
La instrucción 1 se ejecuta (una vez) antes de la ejecución del bloque de código. La declaración 2 define la condición para ejecutar el bloque de código. La instrucción 3 se ejecuta (cada vez) después de que se haya ejecutado el bloque de código.	En el ejemplo anterior, puede leer: La declaración 1 establece una variable antes de que comience el ciclo (var i = 0). La declaración 2 define la condición para que se ejecute el ciclo (i debe ser menor que 5). La declaración 3 aumenta un valor (i ++) cada vez que se ejecuta el bloque de código en el ciclo.

While:

El ciclo while recorre un bloque de código siempre que se cumpla una condición especificada.

while (*condition*) { *// code block to be executed* }	while (i < 10) { text += "The number is " + i; i++; }
condition: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.	En el siguiente ejemplo, el código del ciclo se ejecutará una y otra vez, siempre que una variable (i) sea menor que 10:

Do while:

El ciclo do/while es una variante del ciclo while. Este ciclo ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, luego repetirá el ciclo siempre que la condición sea verdadera.

do { *// code block to be executed* }	do { text += "The number is " + i; i++;

while (*condition*);	} while (i < 10);
condition: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.	En el siguiente ejemplo, el código del ciclo se ejecutará una y otra vez, siempre que una variable (i) sea menor que 10

- **Sentencias de control**

if (condición) sentencia1 [else sentencia2]	**Condición** Una expresión que puede ser evaluada como verdadera o falsa. **Sentencia1** Sentencia que se ejecutará si condición es evaluada como verdadera. Puede ser cualquier sentencia, incluyendo otras sentenccias if anidadas. Para ejecutar múltiples sentencias, use una sentencia block ({ ... }) para agruparlas. **Sentencia2** Sentencia que se ejecutará si condición se evalúa como falsa, y exista una cláusula else. Puede ser cualquier sentencia, incluyendo sentencias block y otras sentencias if anidadas.
Multiples sentencias if...else pueden ser anidadas para crear una cláusula else if: if (*condición1*) *sentencia1* else if (*condición2*) *sentencia2* else if (*condición3*) *sentencia3* ... else *sentenciaN*	**Ejemplo** if (condición) { sentencia1 }else if (Condicion){ Sentencia2 } else { sentencia3 }

- **Sentencias break, continue, return**

- Break

La declaración break también se puede utilizar para saltar de un bucle.

La declaración break rompe el ciclo y continúa ejecutando el código después del ciclo (si lo hay):

- Continue

La declaración continue rompe una iteración (en el ciclo), si ocurre una condición específica, y continúa con la siguiente iteración en el ciclo.

- Return

Cuando JavaScript llega a una declaración return, la función dejará de ejecutarse.

Si la función fue invocada desde una declaración, JavaScript "regresará" para ejecutar el código después de la declaración de invocación.

- **Expresiones**

La expresiones son todo tipo de operación a nivel aritmético, lógico y relacional, a continuación se detalla cada uno de los tipos de expresiones que serán válidas dentro del lenguaje y las posibles operaciones que se podrán realizar.

Lógicas (and,or,not,xor):

Los operadores lógicos se utilizan para determinar la lógica entre variables o valores.

Dado eso x = 6y y = 3, la siguiente tabla explica los operadores lógicos:

Operator	**Description**	**Example**
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Relacionales (igual, mayor, menor, mayor igual, menor igual, distinto):

Los operadores de comparación se utilizan en declaraciones lógicas para determinar la igualdad o diferencia entre variables o valores.

Dado x = 5, la siguiente tabla explica los operadores de comparación:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Aritméticas (suma, resta, multiplicación, división, adición ++, sustracción --, negativos):

Operadores aritméticos realizan operaciones aritméticas con números (literales o variables).

Operator	Description	Example
+	Suma	var x = 100 + 50; var x = a + b;
-	Resta	var x = 5; var y = 2;

		var z = x - y;
*	Multiplicacion	var x = 5; var y = 2; var z = x * y;
**	Exponenciacion	var x = 5; var z = x ** 2; // result is 25
/	Division	var x = 5; var y = 2; var z = x / y;
%	Modular	var x = 5; var y = 2; var z = x % y;
++	Incremento	var x = 5; x++; var z = x;
--	Decremento	var x = 5; x--; var z = x;

Paréntesis:
La sintaxis para la traducción de las operaciones entre paréntesis es la misma que en java.

- **Tipos de datos**

Un valor primitivo es un valor que no tiene propiedades ni métodos.
Un tipo de datos primitivo son datos que tienen un valor primitivo.

JavaScript define 5 tipos de tipos de datos primitivos:

- string
- number
- boolean
- null
- undefined

Value	**Type**	**Comment**
"Hello"	string	"Hello" is always "Hello"

3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

- **Comentarios**

Comentarios de una sola línea

- Los comentarios de una sola línea comienzan con //.
- Cualquier texto entre //y el final de la línea será ignorado por JavaScript (no se ejecutará).
- Este ejemplo utiliza un comentario de una sola línea antes de cada línea de código:

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Comentarios de varias líneas

Los comentarios de varias líneas comienzan con /*y terminan con */.

Cualquier texto entre /*y */será ignorado por JavaScript.

Este ejemplo utiliza un comentario de varias líneas (un bloque de comentarios) para explicar el código:

```
/*

The code below will change

the heading with id = "myH"

and the paragraph with id = "myP"

in my web page:

*/

document.getElementById("myH").innerHTML = "My First Page";

document.getElementById("myP").innerHTML = "My first paragraph.";
```

- **Declaración y asignación de variables**

1. **var** declara una variable de scope global o local para la función sin importar el ámbito de bloque. Permite hoisting.

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
```

2. **let** declara una variable de scope global, local para la función o de bloque. Es reasignable y no permite hoisting.

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

3. **const** declara una variable de scope global, local para la función o de bloque. No es reasignable, pero es mutable. No permite hoisting.

```
var x = 10;
// Here x is 10
{
  const x = 2;
  // Here x is 2
}
// Here x is 10
```

En general, let sería todo lo que se necesita dentro de un bloque, función o ámbito global. const sería para variables que no van sufrir una reasignación. var se puede relegar para cuando necesitemos hacer hoisting, vamos, casi nunca.

Print:

El console objeto proporciona acceso a la consola de depuración del navegador (por ejemplo, la Consola web en Firefox). Los detalles de cómo funciona varían de un navegador a otro, pero hay un conjunto de características de facto que normalmente se proporcionan.

```
console.log(obj1 [, obj2, ..., objN]);
console.log(msg [, subst1, ..., substN]);
```

obj1 ... objN

Una lista de objetos JavaScript para mostrar. Las representaciones en texto de cada uno de los objetos se agregan y muestran juntas (al final una tras otra), en el orden listado.

msg

Un texto (mensaje) conteniendo cero o más sustituciones de cadenas (sustituciones de strings).

subst1 ... substN

Objetos JavaScript con la sustitución a reemplazar dentro del texto (msg). Esto brinda control adicional en el formato de salida del texto.

Sintaxis del lenguaje Python

A continuación se detalla la sintaxis del lenguaje Python, con sus respectivos ejemplos para la traducción requerida.

- **Clases**

La definición de la sintaxis de una clase en python inicia con la palabra reservada “class” seguida del identificador de la clase. Dentro de una clase es permitido declarar variables globales al igual que métodos y/o funciones.

class IdentificadorClase : # Declaraciones de atributos y métodos	class ejemplo1: # Todo tipo de variables globales y métodos # o funciones

- **Interfaces**

Las interfaces en java son traducidas a Python como una clase normal.

class IdentificadorInterfaz: // Definición de funciones	class interfaz1: // Definición de funciones

- **Métodos y funciones (Parámetros)**

La declaración de métodos y funciones en python no tienen ninguna diferencia al no existir tipos explícitos.

Para una función simplemente declarada se deberá agregar en su traducción un punto y coma al final.

def nombreFuncion (PARÁMETROS) : // Bloque de instrucciones	self suma(int x, int y): // Bloque de instrucciones
self nombreFuncion (PARÁMETROS);	self operacion(int x, int y);

PARÁMETROS: Es el listado de valores permitidos enviados al hacer la llamada a una función o método. La sintaxis se detalla a continuación:

[id1, id2 , …, idn]	x, y, z, s

- **Sentencias de repetición**

Las sentencias de repetición que se utilizaran dentro del lenguaje se limitan a las 3 descritas a continuación.

For:

La sintaxis a utilizar dentro de Python para un For es la siguiente:

for id in range (EXP, EXP): // Bloque de Instrucciones	for x in range (0, a+4): // Bloque de Instrucciones for x in range (20, 5*4): // Bloque de Instrucciones

EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

While:

La sintaxis a utilizar dentro de Python para un While es la siguiente:

while (EXP) { // Bloque de sentencias }	prueba = True while prueba : print("Esto lo verás una vez"), prueba = False; while a < 10 : print("el valor de a es: ", a)

EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

Do while:

La sintaxis a utilizar dentro de Python para un Do While es la siguiente:

do { // Bloque de sentencias } while(EXP);	contador = 0 while True: print(“Contador”, (contador+1)) contador += 1 if contador < 10 : break

EXP: referencia a la sintaxis de una expresión, el conjunto de expresiones se detalla posteriormente.

- **Sentencias de control**

If:
La sintaxis a utilizar dentro de Python para un If... elif… else es la siguiente:

if EXP : // Bloque de sentencias then [elif EXP : // Bloque de sentencias else if . . . . else: // Bloque de sentencias else]	if a > 5 : // Bloque de sentencias then elif a < 5 : // Bloque de sentencias else if else : // Bloque de sentencias else

Se subraya la parte opcional de un if teniendo en cuenta que puede componerse únicamente de una lista de else if o un else o incluso ambos.

- **Sentencias break, continue, return**

Las sentencias break, continue, return estas son utilizadas para generar saltos dentro de la ejecución y flujo del programa.

A continuación se detalla la sintaxis correcta para cada una de las sentencias:

break	break
continue	continue
return [EXP]	return return 4+ 5 return “Cadena” return var

- **Expresiones**

La expresiones son todo tipo de operación a nivel aritmético, lógico y relacional, a continuación se detalla cada uno de los tipos de expresiones que serán válidas dentro del lenguaje y las posibles operaciones que se podrán realizar.

Lógicas (and,or,not,xor):
La sintaxis para una operación lógica se detalla a continuación, donde a y b representan otras expresiones:

AND	and	a and b	binario
OR	or	a or b	binario
NOT	not	not a	unario
XOR	xor	a xor b	binario

Relacionales (igual, mayor, menor, mayor igual, menor igual, distinto):
La sintaxis para una operación relacional se detalla a continuación, donde a y b representan otras expresiones:

>	a > b
<	a < b
>=	a >= b
<=	a <= b
==	a == b
!=	a != b

Aritméticas (suma, resta, multiplicación, división, adición ++, sustracción --, negativos):
La sintaxis para una operación aritmética se detalla a continuación, donde a y b representan otras expresiones:

Suma	EXP + EXP	4 + a
Resta	EXP - EXP	4 - a
Multiplicación	EXP * EXP	c * 6
División	EXP / EXP	q / c c != 0
Adición ++	EXP += 1	a += 1
Sustracción --	EXP --	v -= 1
Negativos -	-EXP	-7

Es importante tomar en cuenta la precedencia y asociatividad de las operaciones.

Paréntesis:
La sintaxis para la traducción de las operaciones entre paréntesis es la misma que en java.

- **Tipos de datos**

Debido a que en Python no se declara un tipo explicitamente, cad uno de los tipos se deben de sustituir por la palabra reservada var. Sintaxis para la los tipos en Python:

Entero	var
Booleano	var (True, False)
Flotante	var
String	var
Carácter	var

- **Comentarios**

Dentro del lenguaje Python serán permitidos 2 tipos de comentarios, estos son unilinea y multilínea, para esto se define la sintaxis correcta de cada uno de ellos.

Unilinea	# Comentario unilinea que termina al encontrar un salto de línea
Multilínea	‘’’ Este tipo de comentario soporta más de una línea ‘’’

- **Declaración y asignación de variables**

La sintaxis para la declaración de variables dentro del lenguaje Python se describen a continuación:

Declaración:

Para la declaración de variables debemos tomar que puede ser una lista de valores del mismo tipo y estos pueden ser asignados o no.

var [id = EXP / id , …];	var x=0, y, z=9 var ft = True var c = 1.2;

Asignación:

La sintaxis para la asignación de variables en java será:

id = EXP;	x = True and False;

- **Llamadas a métodos**

Dentro del lenguaje Python se podrán realizar llamadas a métodos o funciones. La sintaxis para las llamadas es por medio del nombre de la función y una lista de parámetros.

nombre_funcion([PARAMETROS])	mifuncion(x, “cadena”, True)

- **Método y función por defecto**

Main:

La sintaxis que debe implementar en la traducción del main es la siguiente:

```
def main( ):
        //Instrucciones


if __name__ = “__main__”:
        main()
```

Print:

La traducción de java al lenguaje Python define una sintaxis para el método print:

print(EXP); print(EXP , end="");	print("texto con salto de línea"); print("texto sin salto de línea", end="");

Reportes

Reporte de errores

Los analizadores deben estar en la capacidad de recuperarse de errores léxicos y sintácticos.

Errores léxicos:

Se deben visualizar los caracteres que se encuentren en el archivo de entrada que NO pertenezcan al lenguaje. Estos errores se deben de descartar para así no entorpecer al análisis sintáctico.

Errores sintácticos:

Se deben visualizar los errores sintácticos que tenga el archivo, para recuperarse en el análisis semántico, se debe de utilizar la **recuperación de errores en modo pánico, utilizando como carácter el punto y coma (;)**.

No.	Tipo Error	Fila	Columna	Descripción
1	Léxico	12	6	El carácter '¬' no pertenece al lenguaje.
2	Léxico	14	8	El carácter '@' no pertenece al lenguaje.
2	Sintáctico	45	10	Se encontró 'ID' y se esperaba: ' } '.

De existir errores deberán ser visualizados al momento de terminar el análisis en alguna consola y también en una tabla en la interfaz gráfica de parte del cliente.

Lista de Tokens Generada

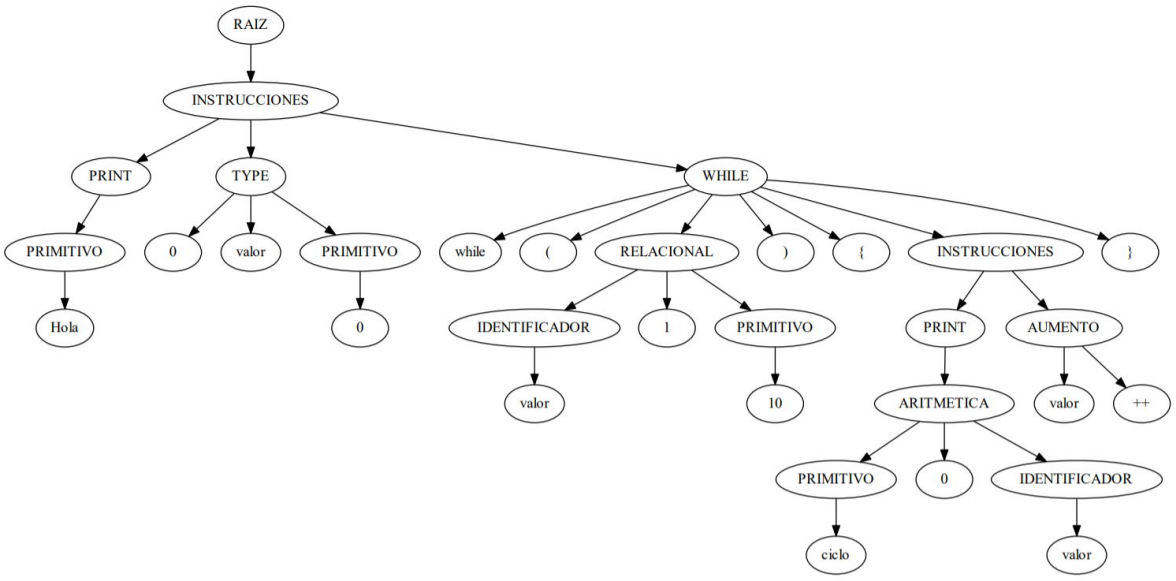
Se deberá generar un reporte de los tokens reconocidos del archivo de entrada.

No.	Fila	Columna	Tipo	Descripción
1	12	6	Identificador	'var1'
2	14	8	Decimal	'10.5985'
2	45	10	Coma	','

Árbol de Análisis Sintáctico

El árbol de análisis sintáctico nos ayuda a visualizar de una forma gráfica cómo fue realizado el análisis sintáctico, generando nodos de las producciones por las cuales fue pasando el analizador sintáctico.

El árbol deberá de ser generado con la herramienta Graphviz (se recomienda generarlo en formato PDF)



Requerimientos

Para poder calificar es necesario que como mínimo se cuente con:

1. El cliente con el sitio web para poder visualizar los archivos de entrada y las consolas. Además de los botones que permiten la funcionalidad del programa.
2. Como mínimo un traductor completo esto incluye todos los reportes correspondientes a dicha traducción.
3. Implementación en contenedores, ya que se darán ejemplos en laboratorio de su uso por lo cual no habrá excusa para no utilizarlos.

Entregables

Para tener derecho a calificación deberá presentar lo siguiente:

1. Código fuente de la solución, incluye código utilizado en ambos traductores y en el sitio web.
2. Manual técnico y de usuario redactados en formato MARKDOWN.
3. Publicación de las imágenes de docker con los servicios instalados en docker hub.
4. Repositorio git (Github, Gitlab, BitBucket) con nombre OLC1_Proyecto2_#carné.

Consideraciones

1. Trabajar de manera Individual.
2. El traductor a python deberá realizarlo mediante un analizador lexico y sintactico de su conocimiento sin el uso de herramientas.
3. El traductor de java a javascript deberá realizarlo mediante el uso de la herramienta jison.
4. Tendrá 30 minutos para la calificación, una vez terminado este tiempo se colocará la nota que se haya obtenido al momento.
5. Las copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ciencias y Sistemas los involucrados.
6. Lenguaje de programación Golang para el sitio web, puede utilizar plantillas html.
7. Los dos traductores deberán ser implementados mediante el uso de nodejs y javascript, es permitido el uso de typescript.
8. El sistema operativo y el IDE son libres.
9. Para la implementación de los contenedores deberá utilizar docker y las imágenes de ubuntu como base para el proyecto (se darán ejemplos del uso de los contenedores dentro del laboratorio desde el inicio para evitar que esto represente un contratiempo).
10. Los archivos de entradas serán proporcionados el día de la calificación.
11. No habrá prórroga y no se recibirán entregas tarde.
12. El proyecto debe contener su número de carné. ej: [OLC 1]P2_#carné.
13. Fecha de entrega: Viernes 30 de octubre de 2020 antes de las 11:59 PM.

Nota: Tomar en cuenta que la entrega será mediante Uedi y que al concluir el limite de tiempo de entrega este ya no dejará subir el entregable.