

# Manual Técnico

# Practica 2

---

Asunción Mariana Sic Sor **201504051**

20 Abril, 2020



## Índice

<b>Descripción del Sistema</b>	<b>1</b>
<b>Navegación</b>	<b>5</b>
<b>Mapa del Sistema</b>	<b>18</b>

## Descripción del Sistema

## Objeto

- Que el usuario pueda hacer uso de este programa.
- Describir a detalle el programa.
- Facilidad en traducir de un lenguaje de programación a otro.
- Que el usuario pueda traducir código en lenguaje c# a lenguaje python.
- Que el usuario pueda incluir cadenas en HTML en lenguaje c# y éstas puedan ser traducidas de una estructura JSON.

## Alcance

- Este programa está dirigido a programadores.

## Funcionalidad

En este programa se puede traducir de un lenguaje c# a lenguaje python, las sentencias que se pueden traducir se describen a continuación:

1. Comentario de una sola línea.
2. Comentarios multilínea.
3. Los tipos de datos (TIPO) admitidos son:
  - a. Entero (int)
  - b. Decimal o Doble (double)
  - c. Carácter (char)
  - d. Booleano (bool)
  - e. Cadena (string)
4. Variables
  - a. Declaración
    - i. Declaración por medio de una lista de variables
    - ii. Declaración simple (TIPO id;)
  - b. Asignación
    - i. Asignación simple (TIPO id = VALOR;)
5. Declaración de Métodos: Método es aquel de tipo “VOID”, es decir, no retorna ningún valor, éste puede llevar o no parámetros.
6. Declaración de Funciones: Función es aquella que, a diferencia del método, sí retorna algún valor y también puede llevar o no parámetros.
7. Declaración del método principal (main): Es aquella que viene una sola vez y en la que no lleva parámetros ni existe un valor para retorno.
8. Sentencias de control
  - a. IF: En ella también se puede declarar una lista de ELSE IF y/o su respectivo ELSE.

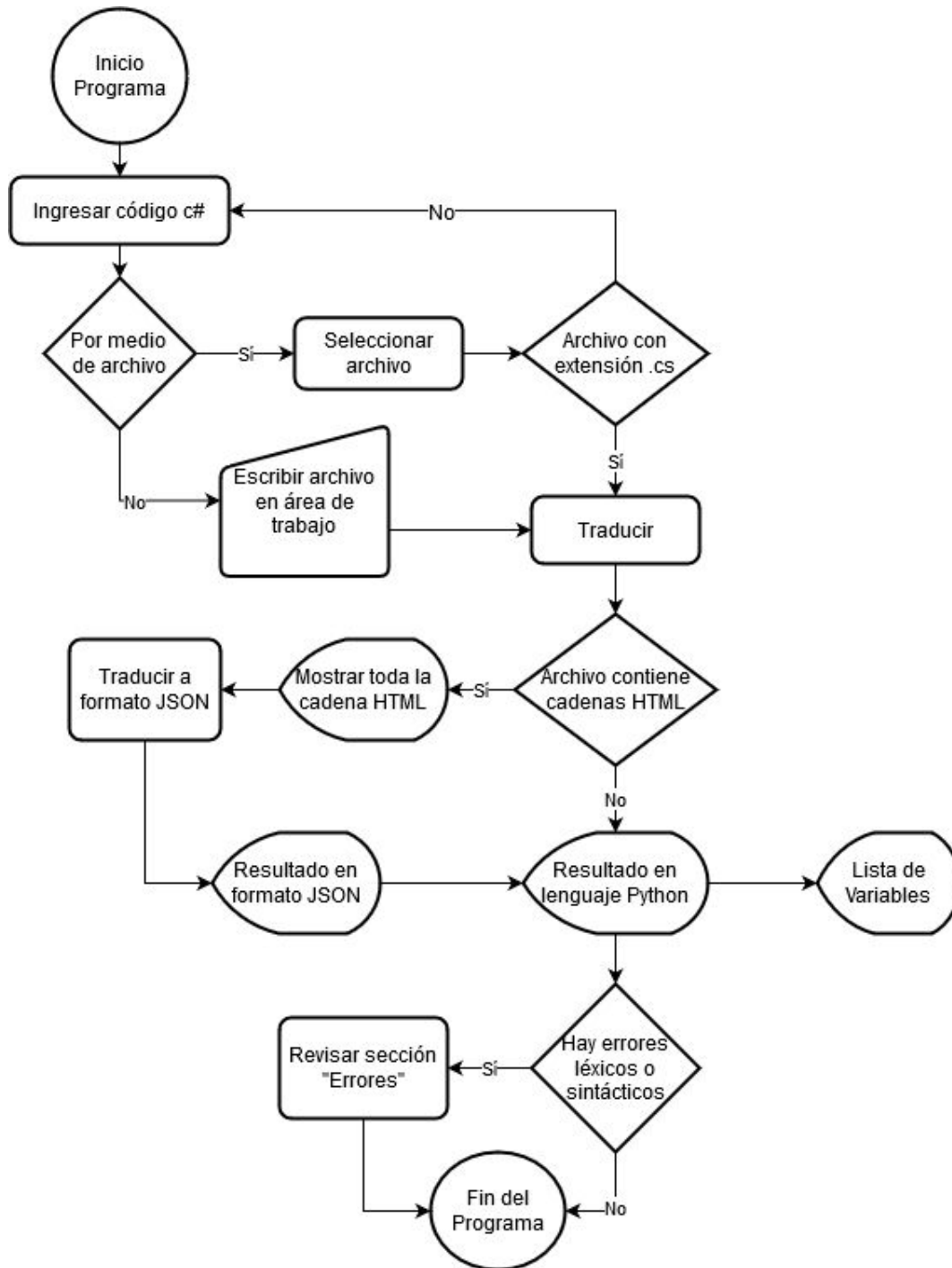
- b. SWITCH: En ella se declara una lista de Casos (case) con su respectiva condición a evaluar. También se puede declarar de manera opcional su caso por defecto (default).
- 9. Sentencias de Repetición
  - a. FOR
  - b. WHILE
  - c. DO-WHILE
- 10. Sentencia de Imprimir: la que sirve para mostrar texto en consola con la estructura de `console.write(EXPRESION);`
- 11. RETURN: la sentencia que indica el retorno de un valor (EXPRESION)
- 12. BREAK: esta sentencia puede estar únicamente en las sentencias de repetición y dentro de una sentencia switch.
- 13. CONTINUE: esta sentencia únicamente se utiliza en las sentencias de repetición.
- 14. Aspectos Generales a considerar:
  - a. Los parámetros se declaran con el formato: TIPO id. En caso sea necesario declarar uno o más parámetros, se procederá a declarar por medio de coma (,): TIPO id1, TIPO id2, ... , TIPO idn.
  - b. Cualquier expresión (EXPRESION) a declarar en el programa puede ser aritmética, lógica, relacional, un TIPO de dato primitivo o bien, una cadena HTML.
    - i. Aritmética
      - 1. Suma (+).
      - 2. Resta (-).
      - 3. Multiplicación (\*).
      - 4. División (/).
    - ii. Lógica
      - 1. And (&&)
      - 2. Or (||)
      - 3. Not (!)
    - iii. Relacionales
      - 1. Mayor (>)
      - 2. Menor (<)
      - 3. Mayor ó igual (>=)
      - 4. Menor ó igual (<=)
      - 5. Igual (==)
      - 6. Distinto de (!=)
    - iv. Cadena HTML: esta consiste en un texto escrito en lenguaje HTML contenida dentro de comillas simples (").
- 15. Lenguaje HTML: Tal como se mencionó antes, está admitido el lenguaje HTML



- a. Las etiquetas admitidas son:
  - i. HTML: indica el comienzo del documento HTML
  - ii. HEAD: delimita el encabezado del documento
  - iii. BODY: delimita el cuerpo del documento
  - iv. TITLE: título del documento (se muestra en la pestaña del navegador).
  - v. DIV: divide bloques de contenido.
  - vi. BR: realiza un salto de línea.
  - vii. P: párrafo de texto
  - viii. H1 a H4: Encabezados de página
  - ix. BUTTON: Muestra un botón.
  - x. LABEL: etiqueta de texto
  - xi. INPUT: caja para ingresar
- b. Atributos admitidos:
  - i. Color de fondo (style="background:COLOR"): este atributo será únicamente para las etiquetas de BODY y DIV. Para este estilo, los colores permitidos son:
    - 1. Amarillo (yellow)
    - 2. Verde (green)
    - 3. Azul (blue)
    - 4. Rojo (red)
    - 5. Blanco (white)
    - 6. Celeste (skyblue)

## Navegación

### 1. Programa en general

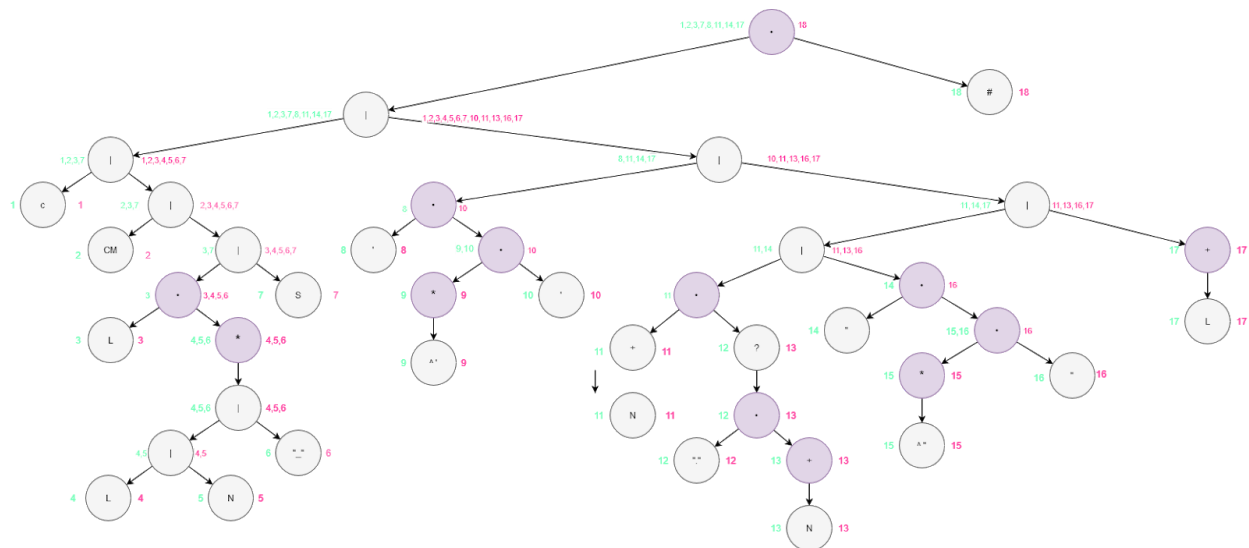


## 2. Análisis Léxico para lenguaje C#

### a. Expresiones regulares

- Letra =  $L = [A-Za-z\tilde{N}\tilde{n}]$
- Número =  $N = [0-9]$
- Comentario simple =  $// [^\n]* \backslash n$
- Comentario múltiple =  $/* [^\n]* */$
- Identificador =  $[L | \_ ] \cdot [L | N | \_ ]^*$
- Símbolo =  $[“,”, “=”, “”, “+”, “*”, “-”, “/”, “(”, “)”, “[”, “]”, “>”, “<”, “:”, “.”, “&”, “|”, “!”]$
- Caracter =  $[^']$
- Cadena HTML =  $[^']*$
- Cadena =  $[^"]^*$

### b. Árbol de Expresión Regular = $[CICM|(L| \_ ) \cdot (L|N| \_ )^* |S| [^'] | [^']* | [^"]^* ] \cdot \#$



### c. Tabla de Sigüientes

# Hoja	Hoj a	Siguiente
1	C	18
2	CM	18
3	L	4,5,6,18
4	L	4,5,6,18

5	N	4,5,6,18
6	“ _ ”	4,5,6,18
7	S	18
8	‘	9,10
9	^ ‘	9,10
10	‘	18

11	N	11,12,18
12	“ . ”	13
13	N	13,18
14	“	15,16
15	^ “	15,16
16	“	18

17	L	17,18
18	#	-----

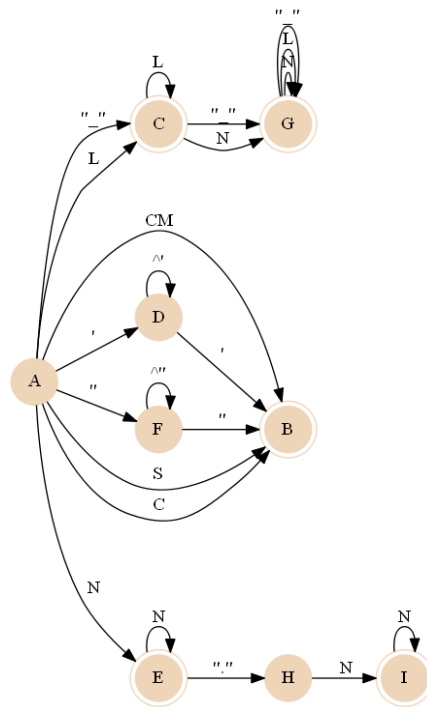
d. Tabla de Transiciones

Estado	Hojas que Conforman	C	CM	L	N	“ ”	S	‘ ’	^	“ ”	“ ”	^
A	1,2,3,7,8,11,14,17	B	B	C	E	-	B	D	-	-	F	-
B	18	-	-	-	-	-	-	-	-	-	-	-
C	4,5,6,17,18	-	-	C	G	G	-	-	-	-	-	-
D	9,10	-	-	-	-	-	-	B	D	-	-	-
E	11,12,18	-	-	-	E	-	-	-	-	H	-	-
F	15,16	-	-	-	-	-	-	-	-	-	B	F
G	4,5,6,18	-	-	G	G	G	-	-	-	-	-	-
H	13	-	-	-	I	-	-	-	-	-	-	-
I	13,18	-	-	-	I	-	-	-	-	-	-	-

\* Los marcados con rojo, son estados de aceptación.

e. Autómata





- f. Autómata en Código: El análisis léxico se encuentra programado en el archivo `../../html/programacion/principal.js` desde la línea 13 a 267

```

switch(estado){
case "A":
    if(texto[indice] == '/' && texto[indice+1] == '/'){
        indice+=2;
        columna+=2;
        estado = "J";
        auxPalabra+="//";
    }
    else if(texto[indice] == '/' && texto[indice+1] == '*'){
        indice+=2;
        columna+=2;
        estado = "K";
        auxPalabra+="/*";
    }
    else if(texto.charCodeAt(indice) == 10){ // \n
        indice++;
        linea++;
        columna = 1;
        estado = "A";
    }
    else if(esBlanco(texto, indice)){
        indice++;
        columna++;
        estado = "A";
    }
    else if(texto.charCodeAt(indice) == 95 || esLetra(texto, indice)){ // "_" || es letra = true
        auxPalabra += texto[indice];
        estado = "C";
        columna++;
        indice++;
    }
    else if(texto.charCodeAt(indice) == 39){ // '
        estado = "D";
        auxPalabra+="'";
        columna++;
        indice++;
    }
}

```

Utiliza métodos auxiliares, para comprobar si es letra, número, símbolo, etc.

```
function esNumero(entrada, posicion){
    if(entrada.charCodeAt(posicion) > 47 && entrada.charCodeAt(posicion) < 58)
        return true;
    else return false;
}

function esLetra(entrada, posicion){
    if(entrada.charCodeAt(posicion) > 64 && entrada.charCodeAt(posicion) < 91)
        return true;
    else if(entrada.charCodeAt(posicion) > 96 && entrada.charCodeAt(posicion) < 123)
        return true;
    else return false;
}

function esSimbolo(entrada, posicion){
    if(entrada.charCodeAt(posicion) == 33 || entrada.charCodeAt(posicion) == 38){
        return true;
    }else if (entrada.charCodeAt(posicion) > 39 && entrada.charCodeAt(posicion) < 48){
        return true;
    }else if (entrada.charCodeAt(posicion) > 57 && entrada.charCodeAt(posicion) < 63){
        return true;
    }else if (entrada.charCodeAt(posicion) > 122 && entrada.charCodeAt(posicion) < 126){
        return true;
    }else{ return false; }
}
```

Estos métodos hacen uso de los valores ASCII para hacer la respectiva comprobación.

También de variables auxiliares que se encuentran declaradas al inicio del archivo

```
var listaToken = new Array();
var tokenHTML = new Array();
var estado = "A";
var indice = 0;
var linea = 1;
var columna = 1;
var reservadas = ["int", "double", "char", "bool", "string", "void", "main", "if", "do", "false", "continue", "else", "switch", "case", "break",
    "console", "write", "default", "for", "while", "true", "return"];
```

### 3. Análisis Sintáctico para el lenguaje C#

#### a. Gramática utilizada

INICIO -> LINSTRUCCION

LINSTRUCCION -> comSimple LINSTRUCCION

| comMultilinea LINSTRUCCION

| int id FOV LINSTRUCCION

| double id FOV LINSTRUCCION

| char id FOV LINSTRUCCION

| bool id FOV LINSTRUCCION

```

| string id FOV LINSTRUCCION

| void MOM LINSTRUCCION

| if para E parc llavea LINSTRUCCION llavec LELSEIF ELSE LINSTRUCCION

| switch para E parc llavea LCASES DEFAULT llavec LINSTRUCCION

| for para DECLFOR pyc E pyc AOD parc llavea LINSTRUCCION llavec
LINSTRUCCION

| while para E parc llavea LINSTRUCCION llavec LINSTRUCCION

| do llavea LINSTRUCCION llavec while para E parc pyc LINSTRUCCION

| console punto write para E parc pyc LINSTRUCCION

| return OPCR LINSTRUCCION

| break pyc LINSTRUCCION

| continue pyc LINSTRUCCION

| id VARIABLE LINSTRUCCION

| ε

AOD -> E mas mas

| E menos menos

| ε

FOV -> para LPARAM parc llavea LINSTRUCCION llavec

| coma id LISTAID igual E pyc

| igual E pyc

LPARAM -> int id LPARAM

| double id LPARAM

| char id LPARAM

| bool id LPARAM

```



| string id LPARAM

| coma PARAM LPARAM

|  $\epsilon$

PARAM -> int id

| double id

| char id

| bool id

| string id

LISTAID -> coma id LISTAID

|  $\epsilon$

MOM -> id para LPARAM parc llavea LINSTRUCCION llavec

| main para parc llavea LINSTRUCCION llavec

LELSEIF -> else if para E parc llavea LINSTRUCCION llavec LELSEIF

|  $\epsilon$

ELSE -> else llavea LINSTRUCCION llavec

|  $\epsilon$

LCASES -> case E dosp LINSTRUCCION LCASES

|  $\epsilon$

DEFAULT -> default dosp LINSTRUCCION

DECLFOR -> int id igual E

| double id igual E

| char id igual E

| bool id igual E

| string id igual E



| id igual E

OPCR -> E pyc

| pyc

E -> id TIPOP E

| numero TIPOP E

| cadena TIPOP E

| caracter TIPOP E

| true TIPOP E

| false TIPOP E

| negacion E

| menos E

| cadenaHTML

|  $\epsilon$

TIPOP -> mas

| menos

| multiplicacion

| division

| and

| or

| mayor

| menor

| mayor igual

| menor igual

| igual igual

l no igual

- b. Codificación de Análisis Sintáctico: Para programarlo se utilizó el método de Parea como base. Se encuentra en .././html/programacion/sintactico.js de línea 2 a 822

Se utilizó el método booleano “tkIguar” el cuál recibe la cadena de entrada y la cadena con la que debería de hacer ‘match’. Si lo anterior es cierto, retorna verdadero y aumenta una posición el índice (i) de la lista de tokens generada en el análisis léxico, de lo contrario, únicamente retorna falso.

```
function tkIguar(elemento, entrada){
    if(elemento == entrada){
        if(i < lista.length){
            i++;
            return true;
        }
        else return false;
    }else return false;
}
```

Seguido de eso, se encuentran como métodos todas las NO TERMINALES de la gramática

```
function LINSTRUCCION(conTab){...}
function LCASES(conTab){...}
function DEFAULT(conTab){...}
function DECLFOR(){...}
function LELSEIF(conTab){...}
function ELSE(conTab){...}
function LPARAM(){...}
function LISTAID(tipo){...}
function OPERACION(){...}
function E(){...}
```

Luego con la ayuda del método “tkIguar” se va comparando cada elemento de la lista

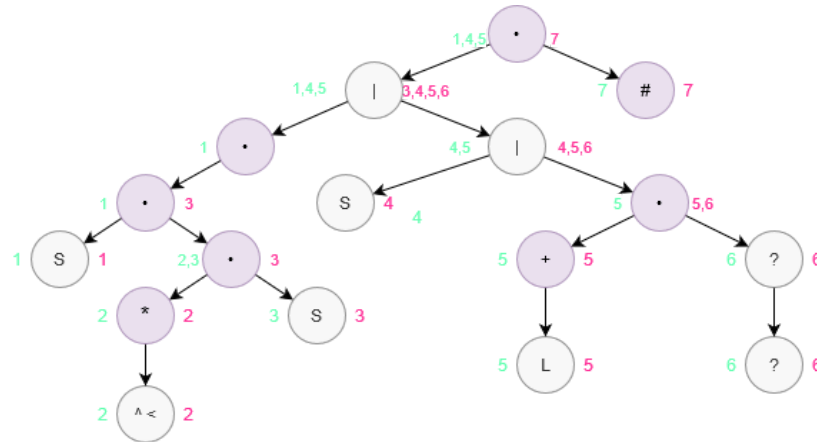
```
/* METODOS */ else if(tkIguar(lista[i].valor.toLowerCase(), "void")){...}
/* IF */ else if(tkIguar(lista[i].valor.toLowerCase(), "if")){...}
/* SWITCH */ else if(tkIguar(lista[i].valor.toLowerCase(), "switch")){...}
/* FOR */ else if(tkIguar(lista[i].valor.toLowerCase(), "for")){...}
/* WHILE */ else if(tkIguar(lista[i].valor.toLowerCase(), "while")){...}
/* DO */ else if(tkIguar(lista[i].valor.toLowerCase(), "do")){...}
/* CONSOLE.WRITE */ else if(tkIguar(lista[i].valor.toLowerCase(), "console")){...}
/* RETURN */ else if(tkIguar(lista[i].valor.toLowerCase(), "return")){...}
/* BREAK */ else if(tkIguar(lista[i].valor.toLowerCase(), "break")){...}
/* CONTINUE */ else if(tkIguar(lista[i].valor.toLowerCase(), "continue")){...}
```

#### 4. Análisis Léxico para cadenas HTML

a. Expresiones Regulares

- i. Símbolo =  $S = [ "<", ">", "/", "=", "'", ":", "." ]$
- ii. Reservadas =  $R = \{html, head, body, title, div, br, p, h1, h2, h3, h4, button, red, yellow, green, blue, white, skyblue, label, input, style, background\} = L+N?$
- iii. Cadena =  $S [^ <]^* S$

b. Árbol:  $[S [^ <]^* S \mid L+N? \mid S] \cdot \#$



c. Tabla de Siguietes

# Hoja	Hoja	Siguiente
1	S	2,3
2	^ <	2,3
3	S	7
4	S	7
5	L	5,6,7
6	N	7
7	#	---

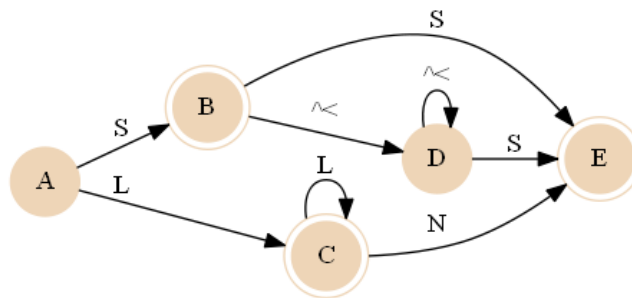
d. Tabla de Estados

Estado	Hojas que conforman	^ <	L	N	S
--------	---------------------	-----	---	---	---

A	1,4,5	-	C	-	B
B	2,3,7	D	-	-	E
C	5,6,7	-	C	E	-
D	2,3	D	-	-	E
E	7	-	-	-	-

\* Los estados marcados en rojo, son estados de aceptación

e. Autómata Finito Determinista



f. Codificación: El análisis léxico se realizó de la misma manera que el analizador del lenguaje C#. Éste se encuentra en ../../html/programacion/principal.js desde línea 269 a 379

```

switch(estado){
  case "A":
    if(texto.charCodeAt(indice) == 62 && texto.charCodeAt(indice+1) != 60){ // hay algo entre > <
      let token1 = {iden: 5, tipo: 'Símbolo', valor: texto[indice], l: linea, c: columna};
      tokenHTML.push(token1);
      auxPalabra = "";
      estado = "D";
      indice++;
    }
    else if(texto.charCodeAt(indice) == 10){ // \n
      indice++;
      linea++;
      columna = 1;
      estado = "A";
    }
    else if(esBlanco(texto, indice)){
      indice++;
      columna++;
      estado = "A";
    }
    else if(esSímboloHTML(texto, indice)){
      auxPalabra+=texto[indice];
      estado = "B";
      columna++;
      indice++;
    }
    else if(esLetra(texto, indice)){
      auxPalabra += texto[indice];
      estado = "C";
      columna++;
      indice++;
    }
  }
}

```



Este analizador utiliza los mismos métodos auxiliares que el analizador del lenguaje C#, además de los siguientes:

```
function esSimboloHTML(entrada, posicion){
    if(entrada.charCodeAt(posicion) > 59 && entrada.charCodeAt(posicion) < 63) return true;
    else if(entrada.charCodeAt(posicion) == 47) return true;
    else if(entrada.charCodeAt(posicion) == 34) return true;
    else if(entrada.charCodeAt(posicion) == 58) return true;
    else return false;
}

function esCadenaVacía(entrada, tamaño){
    var loEs = 0;
    for(var i = 0; i < tamaño; i++){
        if(esBlanco(entrada, i)){ loEs++;}
    }
    if(loEs == tamaño) return true;
    else return false;
}
```

Y algunas variables auxiliares:

```
var tokenHTML = new Array();
var estado = "A";
var indice = 0;
var linea = 1;
var columna = 1;
var reservadaHTML = ["html", "head", "body", "title", "div", "br", "p", "h1", "h2", "h3", "h4", "button", "red", "label", "input", "style",
    "background", "yellow", "green", "blue", "white", "skyblue"];
var codHTML = "";
```

## 5. Análisis Sintáctico para cadena HTML

### a. Gramática

INICIO : < html > DOCUMENTO < / html >

DOCUMENTO : HEAD CUERPO

HEAD : < head > TITLE < / head >

TITLE : < title > CADENA < / title >

CUERPO : < body STYLE > LETIQ < / body >

LETIQ : < ETIQ

| ε

ETIQ : div STYLE > LETIQ < / div > LETIQ

| br > LETIQ

| p > CADENA < / p > LETIQ

| h1 > CADENA < / h1 > LETIQ

| h2 > CADENA < / h2 > LETIQ

| h3 > CADENA < / h3 > LETIQ

| h4 > CADENA < / h4 > LETIQ

| button > CADENA < / button > LETIQ

| label > CADENA < / label > LETIQ

| input > LETIQ

STYLE : style = “ background dosp COLOR “

| ε

COLOR : yellow

| green

| blue

| white

| skyblue

| red

CADENA : cadena CADENA

| < br > CADENA

| ε

- b. Codificación: De la misma manera que la codificación del lenguaje C#, con el método de Parea. Éste se encuentra en ../../html/programacion/sintacticoHTML.js desde línea 1 a 205. Su método “tkHlgual”:

```
function tkHlgual(elemento, entrada){  
    if(elemento == entrada){  
        if(j < listaH.length){  
            j++;  
            return true;  
        }else return false;  
    }else return false;  
}
```

Cada NO TERMINAL se encuentra, de igual manera, como métodos

```
function INICIO(){...}
function DOCUMENTO(){...}
function HEAD(){...}
function TITLE(){...}
function CUERPO(){...}
function LETIQ(numTab){...}
function STYLE(numTab){...}
function COLOR(){...}
```

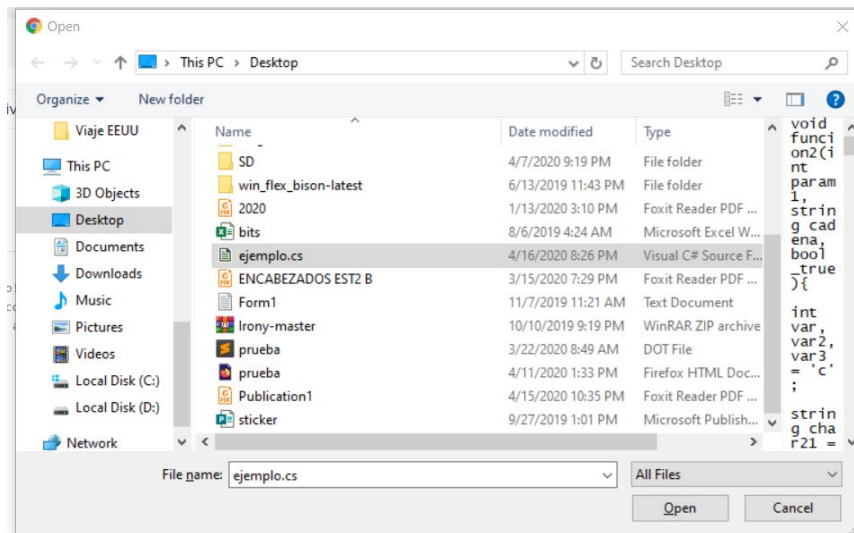
Y por último se encuentran las comparaciones con casa palabra en la lista de Tokens generada por el Scanner

```
/* DIV */if(tkH Igual(listah[j].valor.toLowerCase(), "div")){...}
/* BR | INPUT */else if(tkH Igual(listah[j].valor.toLowerCase(), "br") || tkH Igual(listah[j].valor.toLowerCase(), "input")){...}
/* P .. LABEL */else if(tkH Igual(listah[j].valor.toLowerCase(), "p") || tkH Igual(listah[j].valor.toLowerCase(), "h1") || tkH Igual(listah[j].valor.toLowerCase(), "h3") || tkH Igual(listah[j].valor.toLowerCase(), "h4") || tkH Igual(listah[j].valor.toLowerCase(), "button") || tkH Igual(listah[j].valor.toLowerCase(), "label")){...}
```

## Mapa del Sistema

### Página de Inicio

1. **Carga Archivo:** En esta caja se podrán cargar archivos en extensión '.cs'. En dónde al oprimirlo, lo primero que saldrá será la siguiente ventana donde se podrá elegir el archivo deseado.



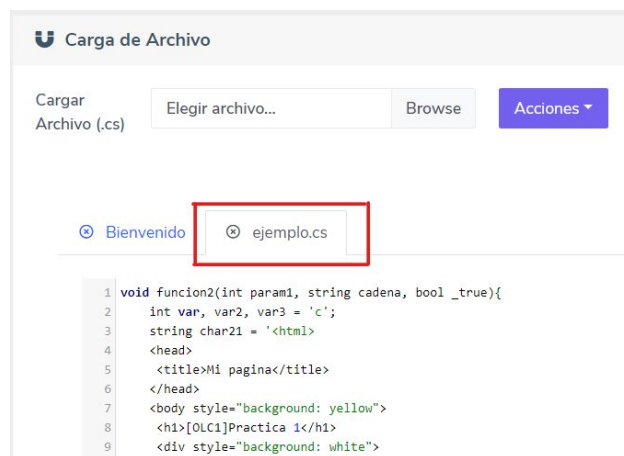
Al momento de dar dos veces clic el programa avisará en la esquina superior derecha



De lo contrario,



Si la carga de archivo fue exitosa, se abrirá en una nueva pestaña

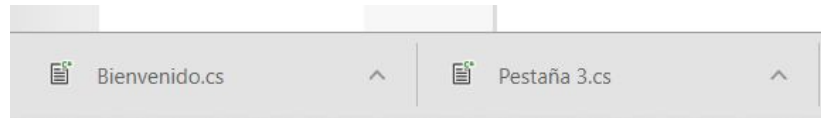


## 2. Botón de 'Acciones': Este botón contiene las siguientes opciones

### a. Agregar Pestaña



- b. Guardar archivos: permite descargar en un archivo .cs el contenido de la pestaña actual.



- c. Traducir: Al momento de traducir, si todo el código en c# está escrito correctamente, la página lo anunciará

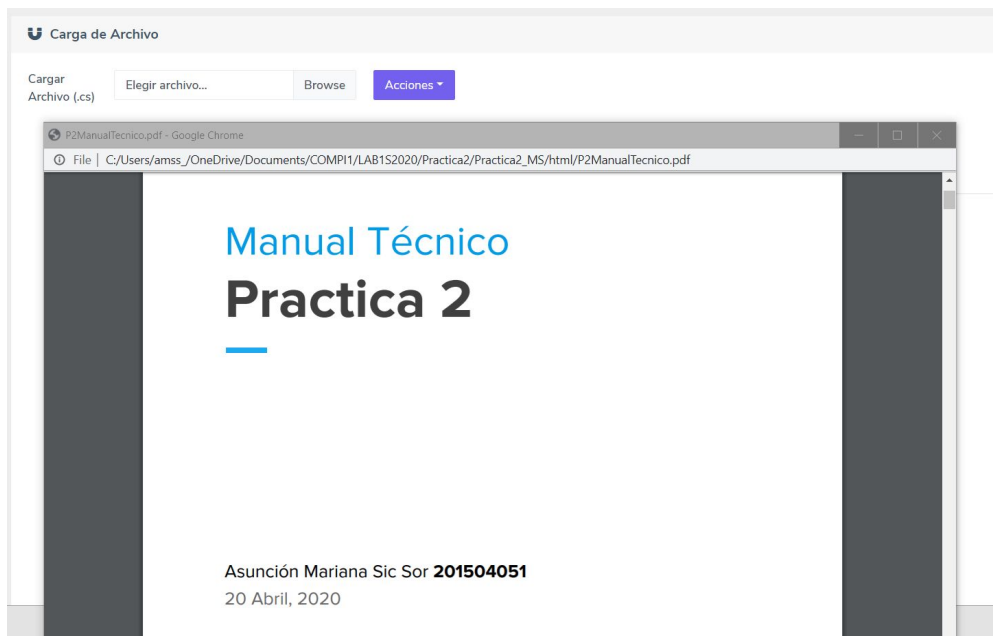


De lo contrario,

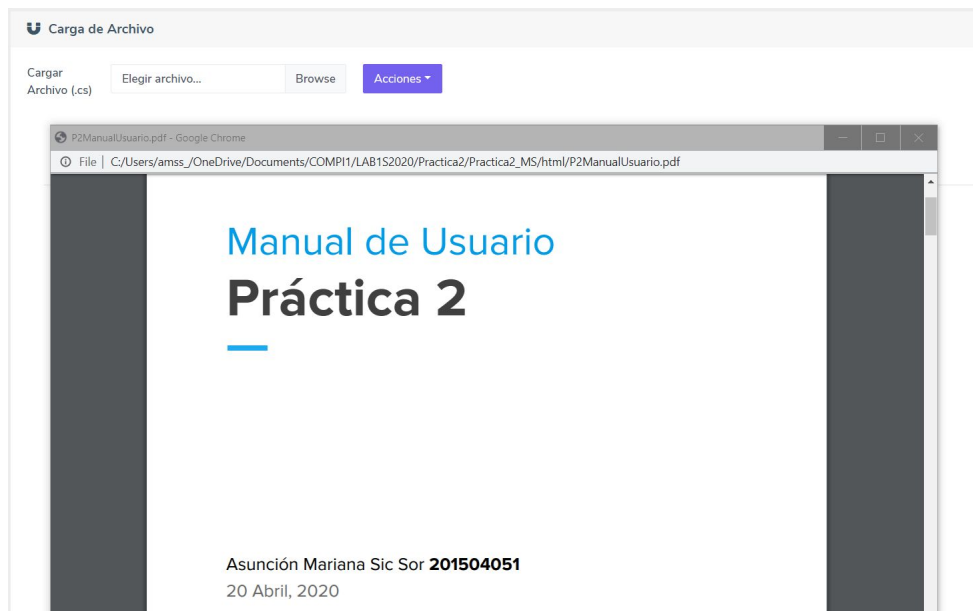


Los códigos resultantes se encuentran en las pestañas de “Python” y “HTML”, de las cuales se dan detalles más adelante.

- d. Ver Manual Técnico: Abre una nueva ventana con el Manual Técnico del programa



- e. Ver Manual de Usuario: Abre una nueva ventana con el Manual de Usuario del programa.



3. **Área de multipestañas:** En esta área se encuentran todas las pestañas que se hayan agregado, se pueden agregar tantas pestañas según sea necesario.

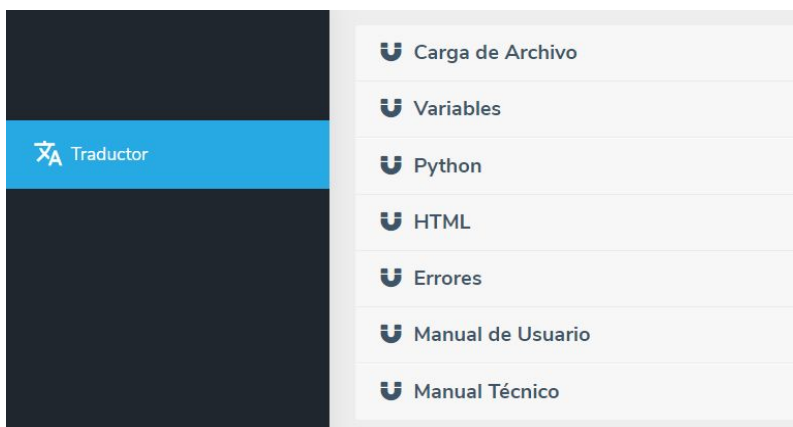
⊗ Bienvenido   ⊗ ejemplo.cs   ⊗ Pestaña 3   ⊗ Pestaña 4

- 4. Área de Trabajo:** En esta área se escribe todo el código en c# que se desee traducir a código Python. Cada pestaña posee su propia área de trabajo.

⊗ Bienvenido   ⊗ ejemplo.cs   ⊗ Pestaña 3   ⊗ Pestaña 4

```
1  /* Todo el
2     código acá */
3
4  int _num1 = 5;
5  double _resultado = 3 + _num1;
6  void main(){
7      Console.WriteLine("El resultado es: " + _resultado);
8  }
9
```

- 5. Página Principal:** La página principal, consiste en siete secciones



- Carga de Archivo: Ya se dio su descripción anteriormente.
- Variables: En esta área se encontrará un listado de las variables encontradas en el archivo de entrada, se detalla su nombre, tipo y línea donde se encuentra.

Variables			
Lista de Variables			
#	Nombre	Tipo	Línea
1	_num1	int	4
2	_resultado	double	5
#	Nombre	Tipo	Línea

- c. Python: Se detalla en la sección de “Código C# a Python”.
- d. HTML: Se detalla en la sección de “Código HTML” y “Código HTML a JSON”
- e. Errores: En esta sección se encuentran los errores léxicos y sintácticos encontrados en el programa. Se detalla su Tipo Error (léxico o sintáctico), línea donde se encuentra, columna donde se encuentra y una breve descripción de él.

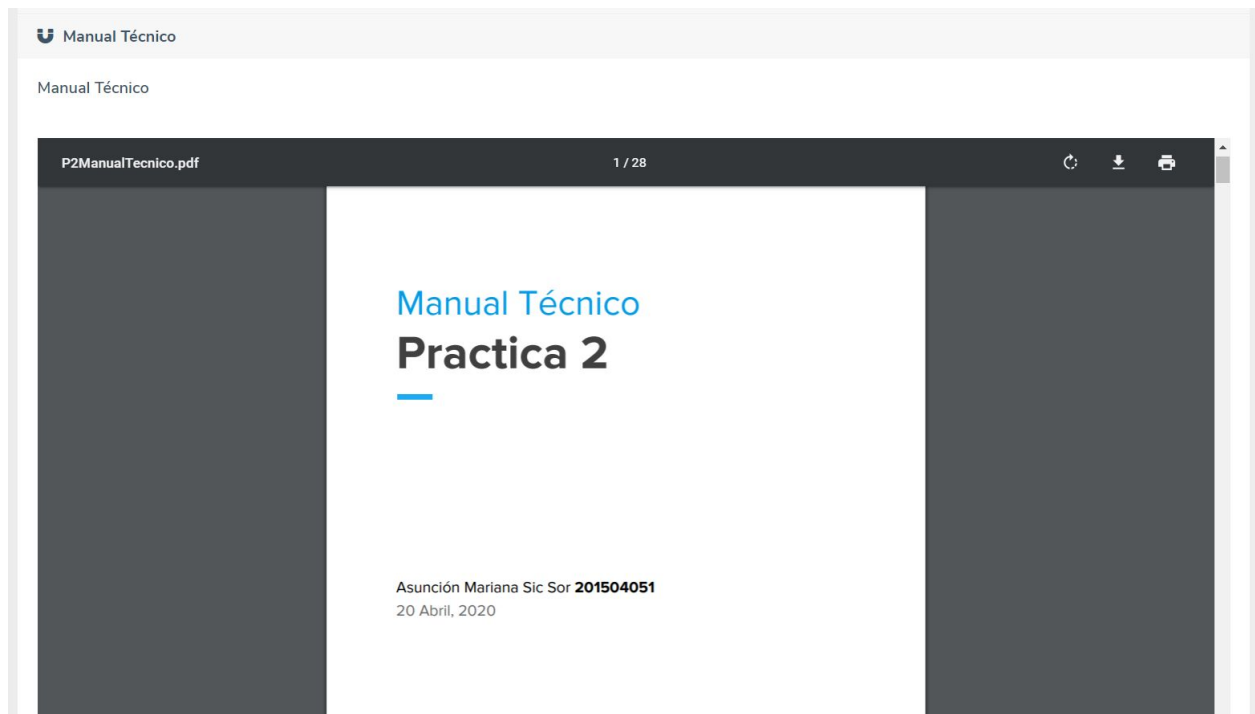
Errores				
Lista de Errores de Análisis				
#	Tipo Error	Línea	Columna	Descripción
1	Léxico	3	1	"@" no pertenece al lenguaje
2	Sintáctico	8	2	Se esperaba ")" ó ";" y vino }

- f. Manual de Usuario: Se encuentra el Manual de Usuario del programa







g. Manual Técnico: Se encuentra el Manual Técnico del programa



Código C# a Python

El código resultante Python se encontrará en la pestaña de “Python”

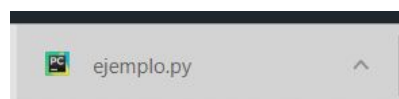
 Python

 Código resultante en Python

```
46
47 def main():
48     if var1 < =5:
49         var var2 = 68.878 * 87 / 457
50     else:
51         var varr34r3w = 896
52         break
53     if (5.8 * 69 + 84) != 568.58:
54         var var3 = 'c'
55         while var3 < 68 :
56             if var1 < =5:
57                 var var2 = 68.878 * 87 / 457
58                 print(var2)
59                 continue
60                 """ comentario
61                 de varias
62                 lineas
63                 """
64                 """
65                 """
66                 print(var1, varr34r3w)
67     elif vary and (223 < 5):
68         #comentario de una linea
69         var suma = 56 + (58 * 7 / 2)
70         # otro comentario de una linea
71         #ahora toca un if mas elaborado
72
73
74 if __name__=="__main__":
75     main()
76
77
```


Descargar Python


El código en esta área no se puede editar, sin embargo, se encuentra el botón “Descargar Python” el cual descarga el código de salida en extensión .py




## Código HTML

Las cadenas escritas en HTML de todo el archivo de entrada se encontrarán en pestaña “HTML”

 HTML

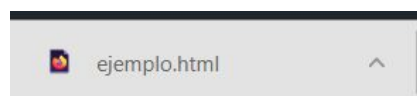
 Código resultante en HTML

 Código resultante JSON

```
1 <html>
2   <head>
3     <title>Mi pagina</title>
4   </head>
5   <body style="background: yellow">
6     <h1>[OLC1]Practica 1</h1>
7     <div style="background: white">
8       <h2>Encabezado 2</h2>
9       <p>
10        Este es un bloque <br>
11        de texto, para una <br>
12        prueba.
13      </p>
14      <br>
15    </div>
16    <div style="background: skyblue">
17      <h2>Llenar los campos</h2>
18      <label>Ingrese su nombre: </label>
19      <br>
20      <input>
21      <br>
22      <button>Mi boton</button>
23      <br>
24    </div>
25    <div style="background: white">
26      <h4>Encabezado 4</h4>
27      <p>
28        Este es un bloque <br>
29        de texto, para una <br>
30        prueba.
31      </p>
```

Descargar HTML

Este código no es editable, sin embargo, se encuentra el botón “Descargar HTML” el cual descarga lo que se ve en la pantalla en un archivo con extensión .html



### Código HTML a JSON

Las cadenas escritas en HTML de todo el archivo de entrada se traducen a formato JSON, ésta se encuentra en la subpestaña “Código resultante JSON” de la misma pestaña “HTML”.



U HTML

Código resultante en HTMLCódigo resultante JSON

```
1  "HTML":{
2    "HEAD":{
3      "TITLE":{
4        "TEXT0":"Mi pagina",
5      },
6    },
7    "BODY":{
8      "STYLE":"background:yellow",
9      "H1":{
10       "TEXT0":"[OLC1]Practica 1",
11     },
12     "DIV":{
13       "STYLE":"background:white",
14       "H2":{
15         "TEXT0":"Encabezado 2",
16       },
17       "P":{
18         "TEXT0":"      Este es un bloque ",
19         "BR":{
20           "true"
21         },
22         "TEXT0":"      de texto, para una ",
23         "BR":{
24           "true"
25         },
26         "TEXT0":"      prueba.      ",
27       },
28       "BR":{
29         "true"
30       },
31     },
32   },
33 }
```

Descargar Json

Este código no es editable, sin embargo, se encuentra el botón “Descargar Json” el cual descarga lo que se ve en la pantalla en un archivo con extensión .json

