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

Práctica 2 - Traductor

Objetivos

General:

1. El estudiante aprenda a generar un analizador léxico y sintáctico.

Específico:

1. El estudiante implemente tanto un analizador léxico como un sintactico entendiendo la funcionalidad de ambos y cómo se relacionan.
2. El estudiante comprenda la funcionalidad de un traductor.

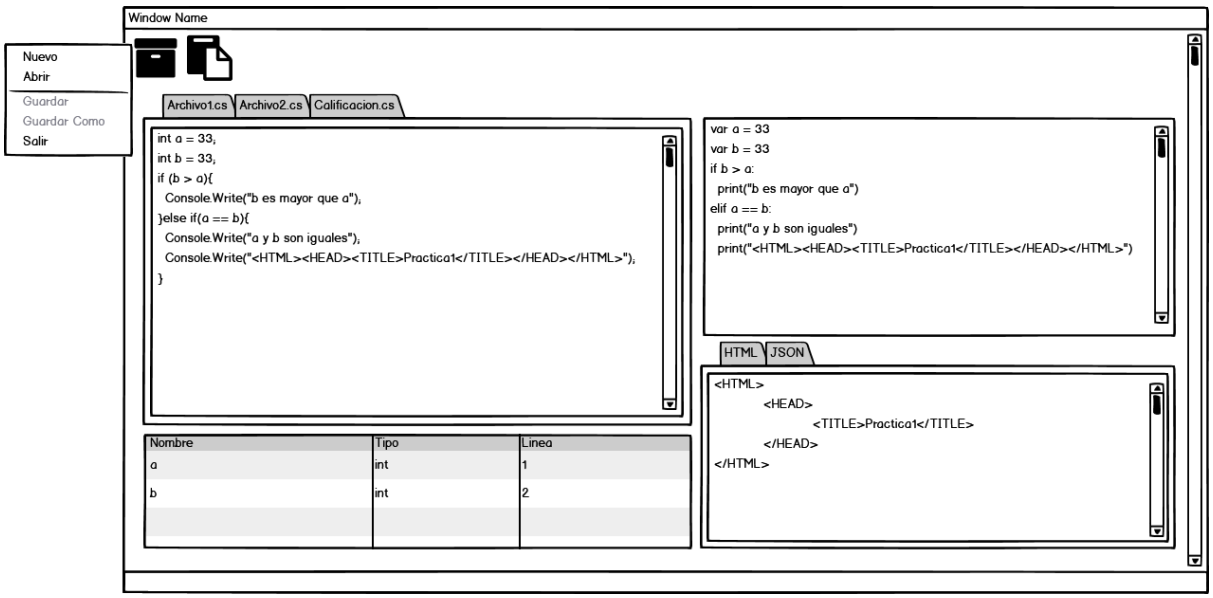
Descripción

Debido a lo tedioso que puede llegar a ser el sobreescribir una aplicación de un lenguaje a otro se le solicita que, como estudiante de sistemas, realice un traductor de lenguaje de programación C# a Python, el cual contiene algunas de las sentencias básicas del lenguaje como lo son declaración y asignación de variables, sentencias de retorno, sentencias de condición y repetición, métodos, funciones entre otros; Del cual se espera que genere la salida del lenguaje python así como el resultado final de texto tipo HTML que contendrá el archivo a traducir y posteriormente obtener la representación en formato JSON de ese código HTML.

Área de Edición y Resultados

El área de edición y resultados será aquella en la cual se realizarán acciones como:

1. Archivo: En esta área se tendrá un menú el cual se encargará de crear nuevos documentos en blanco, abrir documentos existentes, guardar documentos o guardar como.
2. Reportes: Este sera el menu en el cual se podrán generar los reportes de salida en formato .py para la salida en python y en formato .html para la salida html de la página obtenida.
3. Lista de Ventanas: Para facilitar la traducción de varios archivos se podrán generar múltiples pestañas las cuales pueden generarse con nuevos documentos o al momento de abrir uno existente.
4. Consolas: Se tendrán 2 consolas de salida las cuales deben de contener la salida en código python y la salida del código html obtenido, respectivamente. **Ambas consolas deben de tener la restricción de no ser editables.**
5. Tabla de Variables: En este se mostrarán todas las variables identificadas, de las cuales se desea saber: tipo,nombre,línea.



Especificación del programa fuente

Traductor de código C# a Python

Se desea una herramienta para migrar aplicaciones desarrolladas en lenguaje C# a lenguaje Python, sin necesidad de tener que reescribir el código.

Por lo cual se le ha asignado la tarea de crear una aplicación la cual reciba como entrada un archivo en lenguaje C# y devuelve como salida un archivo en lenguaje Python.

A continuación se describe la sintaxis estándar y su conversión a código Python:

Comentarios
El manejo de los comentarios dentro de los lenguajes pueden ser de 2 formas:

- **Comentario de una sola línea:**

C#	// Este es un comentario
Python	# Este es un comentario

- **Comentarios multilínea:**

C#	/* Este es un comentario multilínea */
Python	''' Este es un comentario multilínea '''

Los comentarios pueden venir en cualquier parte del archivo.

Tipos de Dato
Dado que el lenguaje python es de tipado dinámico este no contiene una palabra reservada específica para cada tipo de dato por lo cual cualquier tipo de dato será sustituido por ‘var’.

C#	Python
int	var
double	var
char	var
bool	var
string	var

Declaración y asignación de Variables

C#	TipoDato **Lista_ID** ; TipoDato **Lista_ID** = **EXPRESION** ;
Python	id = **EXPRESION**

Lista_ID: Se podrán declarar n variables en una línea separados por coma si así se desea.

Lista_ID	id1, id2, id3,....., idn

Un id podría estar formado por letras, números y guión bajo.

Expresión

Dentro de una expresión pueden venir operadores aritméticos, llamadas a funciones, llamadas a variables, cadenas, etc.

Ejemplo:

C#	var1 = 5+8*9;
Python	var1 = 5+8*9

Declaración de métodos

- **C#**

```
void id ( PARAMETROS){
    <Sentencias>
}
```

- **Python**

```
def id( PARAMETROS ):
    <Sentencias>
```

Pueden o no llevar parámetros.

Declaración de funciones

- **C#**

```
<TIPO> id ( PARAMETROS){
    <Sentencias>
}
```

- **Python**

```
def id( PARAMETROS ):
    <Sentencias>
```

Pueden o no llevar parámetros.

Declaración del main

- **C#**

```
void main ( ){
    <Sentencias>
}
```

- **Python**

```
def main( ):
    <Sentencias>

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

No lleva parámetros.

Parámetros

Listas de declaraciones para métodos y funciones.

C#	void mifuncion(int a, int b){
	  <Sentencias>
	}

Python	def mifuncion(a, b): <Sentencias>

Sentencias de Control

- **if**

C#	if (b > a){
	  Console.Write("b mayor que a");
	}else if(a == b){
	  Console.Write("a y b son iguales");
	}

Python	if b > a:
	  print("b mayor que a")
	elif a == b:
	  print("a y b son iguales")

- **switch**

C#	switch(valor){
	  case 1:
	  precio = 55;
	  break;
	  case 2:
	  precio = 25;
	  case 3:
	  precio = 40;
	  default:
	  Console.Write("No válido. Escoja 1, 2, o 3.");
	}

Python	def switch(case, precio):
	  switcher = {
	  1: precio = 55,
	  2: precio = 25,
	  3: precio = 40,
	  4: print("No válido. Escoja 1, 2, o 3."),
	  }

Sentencias de Repetición

- **for**

C#	for (int a=0; a<10; a++){
	  Console.Write("el valor de a es: " + a);
	}

Python	for a in range(1,10): print(“el valor de a es: ”, a)

- **while**

C#	while(a < 10){
	  Console.Write("el valor de a es: " + a);
	}

Python	while a < 10 : print("el valor de a es: ", a)

- **do-while**

C#	int a = 1;
	do {
	  Console.Write("el valor de a es: " + a);
	} while(a < 5);

Python	i = 1
	while True:
	  print("el valor de a es: ", a)
	  a = a + 1
	  if (a < 5):
	  break

Sentencia Imprimir

C#	Console.Write("el valor de a es: " + a);
Python	print("el valor de a es: ", a)

Sentencia Return

- **Para métodos**

C#	return;
Python	return

- **Para funciones**

C#	return 5*5;
Python	return 12+8

Sentencia Break

La sentencia break solo se podrá utilizar dentro de las sentencias de repetición y dentro de una sentencia switch.

C#	break;
Python	break

Sentencia Continue

La sentencia continue solo se podrá utilizar solo dentro de las sentencias de repetición.

C#	continue;
Python	continue

Operadores para las Expresiones

- **Aritméticos**

Tipo Operacion	C#/Python
Suma	+
Resta	-
Multiplicación	*
División	/

- **Lógicos**

Tipo Operacion	C#	Python
And	&&	and
Or	||	or
Not	!	not

- **Relacionales**

Tipo Operacion	C#/Python
Mayor	>
Menor	<
Mayor igual	>=
Menor igual	<=
Igual	==
Distinto	!=

Análisis de cadenas y generación de código HTML

Es necesario poder analizar las cadenas de texto ingresadas en la sentencia de impresión, esto para poder generar a su salida un código HTML. Estas cadenas serán distintas al texto común ya que estarán contenidas dentro de comillas simples (‘’).

Etiquetas HTML permitidas

Etiqueta	Descripción
<html>	Indica el comienzo del documento HTML.
<head>	Delimita el encabezado del documento.
<body>	Delimita el cuerpo del documento.
<title>	Título del documento (se muestra en la pestaña del navegador).
	Divide bloques de contenido.
 	Salto de línea.
<p>	Párrafo de texto
<h1>	Encabezados de página puede ir de h1 hasta h4.
<button>	Muestra un botón.
<label>	Etiqueta de texto.
<input>	Caja para ingresar texto.

Atributos para la etiqueta Body y Div:
Las etiquetas body y div contarán con un atributo para poder colocar un color de fondo a los mismos.

style="background:**Color**"

Color
Los colores que serán tomados en cuenta para los fondos y bloques de contenido en la página html son:

- yellow
- green
- blue
- red
- white
- skyblue

Ejemplo:

```
<html>
<head>
    <title>Mi Pagina</title>
</head>
<body style="background:yellow">
    <h1>[OLC1]Practica 1</h1>
    <div style="background:white">
        <h2>Encabezado h2</h2>
        <p>
            Este es un bloque<br>
            de texto, para una<br>
            prueba.
        </p>
        <br>
    </div>

    <div style="background:skyblue">
        <h2>Llenar los campos</h2>
        <label>Ingrese su nombre:</label>
        <br>
        <input>
        <br>
        <button>Mi boton</button>
        <br>

    </div>
</body>
</html>
```

Generación de formato JSON

Con la salida HTML obtenida es necesario traducir esta a su representación en formato JSON, para tener una guia de la traducción a implementar se muestra el siguiente ejemplo:

HTML	JSON

<HTML>
<HEAD>
<TITLE>Practica 1</TITLE>
</HEAD>
<BODY style="background:white">
<H1>Etiqueta h1</H1>
</BODY>
</HTML>
	“HTML”:{
	“HEAD”:{
	“TITLE”:{
	“TEXTO”:”Practica 1”
	}
	},
	“BODY”:{
	“STYLE”:”background:white”,
	“H1”:{
	“TEXTO”:”Etiqueta h1”
	}
	}
	}

Manejo y Recuperación de Errores

Para generar la traducción de código, la entrada debe estar libre de errores léxicos y sintácticos. La aplicación debe estar en la capacidad de **recuperarse de errores léxicos y sintácticos**, para poder corregir de manera eficiente los archivos.

Para los errores léxicos podrá utilizar cualquier método de recuperación, como en la práctica y proyecto realizado. **Para la recuperación de errores sintácticos se deberá utilizar el método de recuperación modo pánico.**

Reporte de errores léxicos y sintácticos:

No.	Tipo error	Línea	Columna	Descripción
1	Léxico	12	6	El carácter ‘¬’ no pertenece al lenguaje.
2	Sintáctico	45	10	Se esperaba… (se encontró id...)...
3	...	...	...	...

- El reporte debe ser en formato HTML y de existir errores deberá ser visualizado al momento de terminar el análisis.

Entregables

1. Código fuente de la solución.
2. Manual Técnico y de Usuario.
3. Link del repositorio de versiones.

Consideraciones

1. Trabajar de manera individual, utilizando como lenguaje de programación **JavaScript, No se permite el uso de herramientas**.
2. Está permitido el uso de **TypeScript como sustituto para JavaScript**.
3. El sistema operativo y el IDE son libres.
4. Cualquier copia total o parcial será reportada y no permitirá la continuidad en el laboratorio.
5. Los archivos de entrada serán proporcionados el día de la calificación.
6. No habrá prórroga y **entregas fuera de la fecha establecida tienen una nota de CERO.**
7. Fecha de entrega **lunes 20 de abril a las 23:59.**