



Universidad de San Carlos de Guatemala  
Facultad de Ingeniería  
Escuela de Ciencias y Sistemas

# Ps-Traductor

Organización de Lenguajes y Compiladores 1

Proyecto 1 - Segundo Semestre 2022

## Manual de Usuario

Presentado por

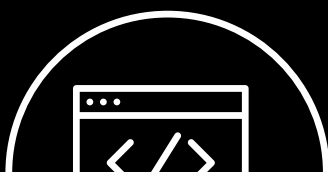
Rodrigo Hernández

Carnet

201900042



# ENCUENTRO



• I. INTRODUCCIÓN	1
1. Objetivo	
2. Requerimientos	
• II. OPCIONES DEL SISTEMA	2
1. Ingreso a la aplicación	2
2. Lenguaje OLC	3
3. Traducción	17
4. Archivo	23
i. Cargar Archivo	
ii. Guardar Como	
5. Reporte	24
i. Diagrama de Flujo	
ii. Errores	
iii. AST	
6. Ver	25
• RECOMENDACIONES	26

# I. Introducción

## 1. Objetivo

Otorgar al usuario un apoyo documentado del uso de la interfaz gráfica al igual del manejo del pseudo código y poder obtener una traducción exitosa de su entrada de texto, sin errores y con éxito uso de la herramienta.

## 2. Requerimientos

- Computadora portátil o de escritorio.
- Mínimo 1GB de Memoria RAM
- Cualquier sistema operativo que contenga un lector de PDF.
- Cualquier sistema operativo que contenga un navegador web.
- Graphviz
- JRE 8.1 en adelante
- JDK 8.1 en adelante
- Resolución gráfica máximo 1900 x 1070 píxeles.

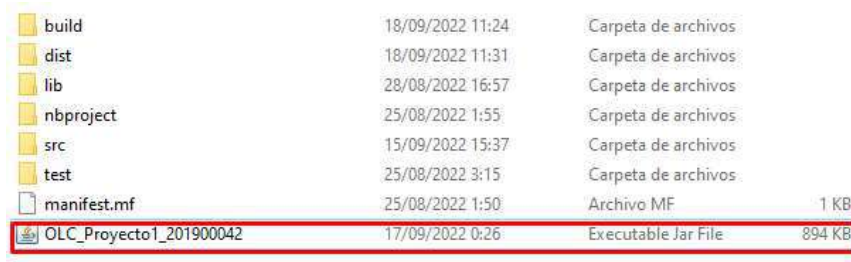
## II. Opciones del Sistema

En las siguientes páginas se estará explicando el uso de cada una de las funcionalidades que tiene la aplicación:

1. Ingreso a la aplicación.
2. Lenguaje OLC.
3. Traducción
4. Archivo:
  - i. Cargar Archivo.
  - ii. Guardar Como.
5. Reporte:
  - i. Diagrama de Flujo.
  - ii. Errores.
  - iii. AST.
6. Ver

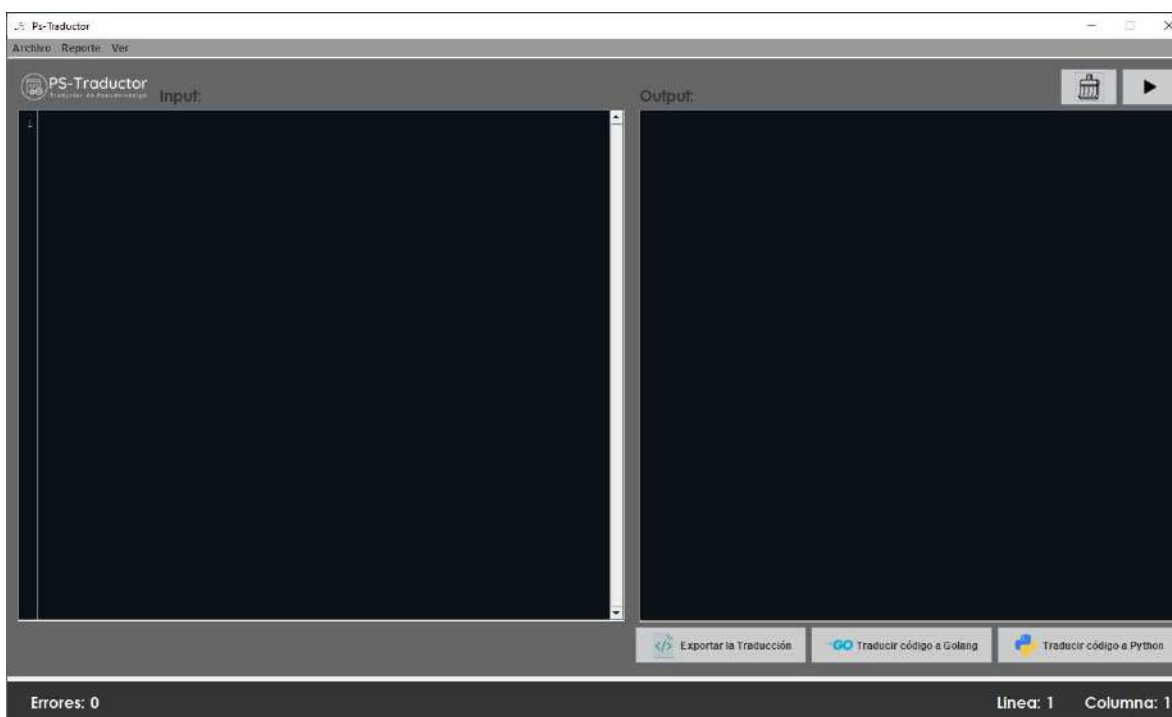
### 1. Ingreso a la aplicación

Para poder ingresar a la aplicación es necesario ubicarse en la carpeta del proyecto y encontrar la siguiente aplicación ejecutable:



build	18/09/2022 11:24	Carpeta de archivos	
dist	18/09/2022 11:31	Carpeta de archivos	
lib	28/08/2022 16:57	Carpeta de archivos	
nbproject	25/08/2022 1:55	Carpeta de archivos	
src	15/09/2022 15:37	Carpeta de archivos	
test	25/08/2022 3:15	Carpeta de archivos	
manifest.mf	25/08/2022 1:50	Archivo MF	1 KB
OLC_Proyecto1_201900042	17/09/2022 0:26	Executable Jar File	894 KB

Posteriormente darle doble click y abrirá la siguiente ventana:

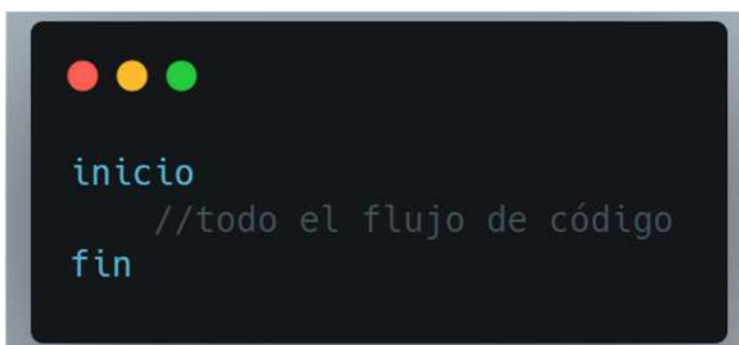


## 2. Lenguaje OLC

En esta parte se encuentra la descripción del manejo del lenguaje OLC que es de tipo pseudo-código.

### a. Global

Esta instrucción permite englobar todo el código dentro de palabras reservadas inicio y fin.



## b. Comentarios

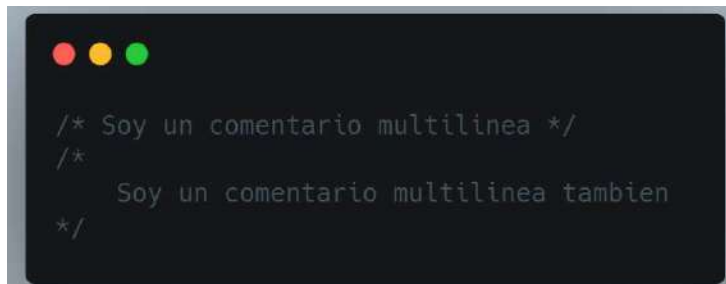
Esta instrucción puede estar en cualquier parte del código. Existen dos tipos de comentarios:

- Comentario de una línea: iniciaran con los caracteres "//", el cual indicará que en adelante todos los caracteres serán considerados como comentarios y el fin del comentario estará delimitado por un salto de línea.



```
//Soy un comentario de una linea
```

- Comentario multilinea: estarán encerrados dentro de los caracteres "/\*" y "\*/".



```
/* Soy un comentario multilinea */  
/*  
    Soy un comentario multilinea tambien  
*/
```

## c. Variables (Identificadores) y nombre de Funciones y Métodos

Para declarar o llamar un nombre de una variable, nombrar u ejecutar una función o método es obligatorio iniciar con un carácter guion bajo y al finalizar también el mismo carácter. Ejemplo:



```
//Es valido:  
_a_  
_b_  
_identificador_  
  
//No es valido:  
a  
b  
identificador
```

### d. Tipo de Datos:

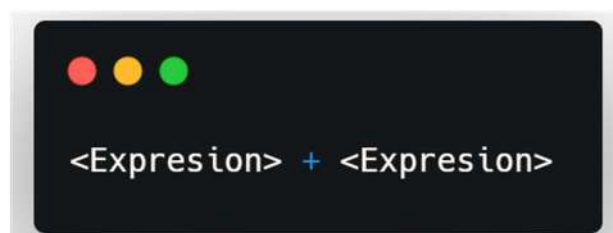
Palabra Reservada	Descripción	Ejemplos
Numero	Puede contener: 1) Conjunto de dígitos que tienen una única vez el carácter punto dentro del conjunto de caracteres. 2) Conjunto de solo dígitos.	10.02 11.1 56.0 105.152 4568 5 94
Cadena	Conjunto de caracteres dentro de comillas dobles.	"Esta es una cadena" "hola mundo" "organización de lenguajes y compiladores" "1"
Boolean	Tipo de dato verdadero o falso.	Verdadero Falso
Caracter	Un carácter dentro de una comilla simple. Es posible ingresar el código ascii (únicamente se usarán los ascii de caracteres del alfabeto).	'o' 'L' "\${70}" "\${75}"


### e. Operadores

En todas las operaciones pueden venir 2 o más operandos.

#### i. Operadores Aritméticos




**Suma:**



**Resta**  
`<Expresion> - <Expresion>`**Multiplicación**  
`<Expresion> * <Expresion>`**División**  
`<Expresion> / <Expresion>`**Potencia**  
`<Expresion> potencia [<Expresion>]`**Módulo**  
`<Expresion> modulo <Expresion>`**Paréntesis**  
`(<Conjunto de Operaciones>)`




## i. Operadores Relacionales

**Mayor**  
<Expresion> mayor <Expresion>**Menor**  
<Expresion> menor <Expresion>**Mayor o igual que**  
<Expresion> mayor\_o\_igual <Expresion>**Menor o igual que**  
<Expresion> menor\_o\_igual <Expresion>**Igual**  
<Expresion> es\_igual <Expresion>**Diferente**  
<Expresion> es\_diferente <Expresion>

## ii. Operadores Lógicos

Or



```
<Expresion> or <Expresion>
```

And



```
<Expresion> and <Expresion>
```

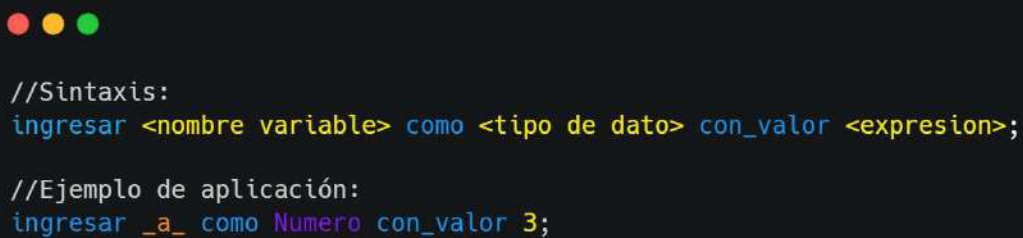
Not



```
not <Expresion>
```

## f. Declaración

Esta instrucción permite ingresar variables dentro del flujo del código. Para ingresar el nombre se necesita reconocer la palabra reservada "ingresar" y el tipo de dato. A continuación, un ejemplo de una declaración:



```
//Sintaxis:  
ingresar <nombre variable> como <tipo de dato> con_valor <expresion>;  
  
//Ejemplo de aplicación:  
ingresar _a_ como Numero con_valor 3;
```

Es posible realizar declaraciones múltiples, para ello es necesario ingresar una lista de nombres, separadas por comas. A continuación, un ejemplo de una declaración múltiple:

```
//Sintaxis:
ingresar <lista de variables> como <tipo de dato> con_valor <expresion>;
<lista de variables> = <nombre1>, <nombre2>, <nombre3>, ...

//Ejemplo de aplicación:
ingresar _a_, _b_, _c_ como Cadena con_valor "hola mundo";
```

## g. Asignación

Esta instrucción cambia el valor de una variable determinada y la actualiza con el valor de una expresión, su estructura consta del nombre de la variable y una expresión. A continuación, un ejemplo de una asignación:

```
//Sintaxis:
<nombre de variable> -> <expresion>;

//Ejemplo de aplicación:
_a_ -> 10;
```

Es posible realizar múltiples asignaciones de la siguiente forma: antes del carácter “->” se ingresa el nombre de las variables separadas por el carácter coma.

```
//Sintaxis:
<lista de variables> -> <expresion>;
<lista de variables> = <nombre1>, <nombre2>, <nombre3>, ...

//Ejemplo de aplicación:
_a_, _b_, _c_ -> "hola mundo";
```

## h. Condicional Si

Esta es una instrucción que permite ejecutar un bloque de instrucciones cuando una condición es válida. La instrucción necesita una condición para ejecutar un bloque asignado, cuando la condición es falsa se ejecuta otro bloque de instrucciones. Es posible que el bloque de instrucciones que ejecuta una condición falsa, sea opcional. Es obligatorio que ingrese una condición en esta instrucción.

```
//Sintaxis:

//Instrucción con bloque de instrucciones con condición verdadera
si <condicion> entonces
    <instrucciones>
fin_si

//Instrucción con bloque de instrucciones con condición verdadera y falsa
si <condicion> entonces
    <instrucciones>
de_lo_contrario
    <instrucciones>
fin_si

//Ejemplo de aplicación:
//Instrucción con bloque de instrucciones con condición verdadera
si _a_ mayor _b_ entonces
    imprimir _a_;
fin_si

//Instrucción con bloque de instrucciones con condición verdadera y falsa
si _a_ mayor _b_ entonces
    imprimir _a_;
de_lo_contrario
    imprimir _b_;
fin_si
```

Caso especial: se usará la palabra reservada "o\_si" cuando una condición es falsa y se trata de volver hacer otra validación con otra condición. Este tipo de ejecución puede ser opcional o repetirse una o muchas veces. A continuación, se muestra un ejemplo de este caso especial.



```
//Sintaxis:
```

```
//Instrucción con bloque de instrucciones con unicamente o_si
```

```
si <condicion> entonces  
    <instrucciones>  
o_si <condicion_nueva> entonces  
    <instrucciones>  
fin_si
```

```
//Instrucción con bloque de instrucciones con o_si y de_lo_contrario
```

```
si <condicion> entonces  
    <instrucciones>  
o_si <condicion_nueva> entonces  
    <instrucciones>  
de_lo_contrario  
    <instrucciones>  
fin_si
```

```
//Ejemplo de aplicación:
```

```
//Instrucción con bloque de instrucciones con unicamente o_si
```

```
si _a_ mayor _b_ entonces  
    imprimir _a_;  
o_si _a_ mayor _c_ entonces  
    imprimir _c_;  
fin_si
```

```
//Instrucción con bloque de instrucciones con o_si y de_lo_contrario
```

```
si _a_ mayor _b_ entonces  
    imprimir _a_;  
o_si _c_ mayor _a_ entonces  
    imprimir _c_;  
de_lo_contrario  
    imprimir _b_;  
fin_si
```

## i. Selección Múltiple

Este tipo de condición permite ejecutar una lista de instrucciones en base a un valor ingresado, existen varias opciones posibles y cuando el valor coincida con una de las opciones se ejecuta un conjunto de instrucciones. Existe una palabra reservada "de\_lo\_contrario" para ejecutar automáticamente cuando la lista de opciones no se cumple, pero es de forma opcional.

```
//Sintaxis:
segun <valor> hacer
  ¿ valor1 ? entonces
    <instrucciones para valor1>
  ¿ valor2 ? entonces
    <instrucciones para valor2>
  de_lo_contrario entonces
    <instrucciones por si no cumple ninguno de los valores de arriba>
fin_segun

//Ejemplo de aplicación:
segun _x_ hacer
  ¿ 1 ? entonces
    imprimir "hola humano";
  ¿ 2 ? entonces
    imprimir "hola mundo";
  de_lo_contrario entonces
    imprimir "adios";
fin_segun
```

## j. Ciclo Para

Este ciclo ejecuta un conjunto de instrucciones, con un límite de repeticiones. Es necesario ingresar un valor inicial y también un valor final, el valor final es el que le indica al ciclo, en momento para terminar de realizar las repeticiones. Otro de los elementos necesarios en este ciclo es el número de pasos que realizará entre cada repetición. Cuando el ciclo no tiene definido el número de pasos, se tomará como defecto el incremento en 1. Es posible que la lista de instrucciones esté vacía.

```
//Sintaxis:

//Con salto no definido
para <variable> -> <valor inicial> hasta <valor final> hacer
    <instrucciones> || null
fin_para

//Con salto definido
para <variable> -> <valor inicial> hasta <valor final> con incremental <valor del salto> hacer
    <instrucciones> || null
fin_para

//Ejemplo de aplicación:

//Con salto no definido
para _i_ -> 0 hasta 10 hacer
    imprimir _i_;
fin_para

//Con salto definido
para _j_ -> 0 hasta 20 con incremental 5 hacer
    imprimir _j_;
fin_para
```

## k. Ciclo mientras

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. Para poder realizar una repetición es necesario que exista una condición. Es posible que la lista de instrucciones esté vacía.

```
//Sintaxis:
mientras <condicion> hacer
    <instrucciones> || null
fin_mientras

//Ejemplo de aplicación:
mientras _x_ es_diferente 10 hacer
    imprimir _x_;
    _x_ -> _x_ + 1;
fin_mientras
```



## I. Ciclo Repetir hasta

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. A diferencia del ciclo anterior, este ciclo realiza una única repetición sin restricción. Para poder realizar las demás repeticiones es necesario que exista una condición. Es posible que la lista de instrucciones esté vacía.

```
//Sintaxis:
repetir
    <instrucciones> || null
hasta_que <condición>

//Ejemplo de aplicación:
repetir
    imprimir "helou";
    _i_ -> _i_+1;
hasta_que _i_ es_diferente 2
```

## m. Retorno

Esta instrucción está encargada de devolver un valor específicamente. Esta instrucción puede reconocer una <condición>, un número o una <expresión aritmética>.

```
//Sintaxis:
retornar <expresión aritmética> ;
retornar <condición>;
retornar <número>;

//Ejemplo de aplicación:
retornar _x_+_y_;
```

## n. Método

Esta instrucción permite agrupar un conjunto de instrucciones y asignarles un nombre para identificarlo dentro del contenido del archivo. No necesita una instrucción de "Retorno" y si en caso es reconocido este tipo de instrucción, debe reportar dicho error.



```
//Sintaxis:
metodo <nombre>
    <instrucciones>
fin_metodo

//Ejemplo de aplicación:
metodo _hola_mundo_
    imprimir "hola mundo";
fin_metodo
```

Es posible agregar parámetros al método, estos parámetros tendrán definido el tipo de dato y su respectivo nombre. Cada uno de los parámetros estará separado por un carácter coma.

```
//Sintaxis:
metodo <nombre> con_parametros (<lista de parametros>)
    <instrucciones>
fin_metodo
<lista de parametros> = <nombre> <tipo de dato> , <nombre> <tipo de dato>, ...

//Ejemplo de aplicación:
metodo _hola_mundo_con_parametros (_x_ Numero, _y_ Numero)
    imprimir _x+_y_;
fin_metodo
```

## o. Función

Esta instrucción permite agrupar un conjunto de instrucciones y asignarles un nombre para identificarlo dentro del contenido del archivo. Esta instrucción si es posible reconocer una instrucción de "Retorno" en su estructura.

```
//Sintaxis:
funcion <nombre> <tipo dato>
    <instrucciones> || null
    <instrucciones de retorno>
fin_funcion

//Ejemplo de aplicación:
funcion _suma_ numero
    ingresar _a_ como numero con_valor 12;
    retornar _a_;
fin_funcion
```

Es posible agregar parámetros a la función, estos parámetros tendrán definido el tipo de dato y su respectivo nombre. Cada uno de los parámetros estará separado por un carácter coma.

```
//Sintaxis:
funcion <nombre> <tipo dato> con_parametros (<lista de parámetros>)
    <instrucciones> || null
    <instrucciones de retorno>
fin_funcion
<lista de parametros> = <nombre> <tipo de dato> , <nombre> <tipo de dato>, ...

//Ejemplo de aplicación:
funcion _suma_ numero con_parametros (_x_ numero, _y_ numero)
    ingresar _total_ como numero con_valor _x_ + _y_;
    retornar _total_;
fin_funcion
```

## p. Llamada de Funciones y Métodos

Este tipo de instrucción realiza la ejecución de un método o función, para poder realizarlo es necesario ingresar el identificador de la función o método y la lista de parámetros necesarios.

```
//Sintaxis:

//ejecutar sin parámetros
ejecutar <identificador>();
//ejecutar con parámetros
ejecutar <identificador>(<Lista de parámetros>);
<Lista de parámetros> = <Expresion1>, <Expresion2>, ...

//Ejemplo de aplicación:

//ejecutar sin parámetros
ejecutar _hola_mundo_();
//ejecutar con parámetros
ejecutar _suma_(2,2);
```

## q. Impresión

Esta instrucción muestra el contenido de una expresión o valor de una variable. Para poder utilizarla es necesario una expresión o valor de una variable. Al terminar de realizar la impresión se genera un salto de línea por defecto.

```
//Sintaxis:

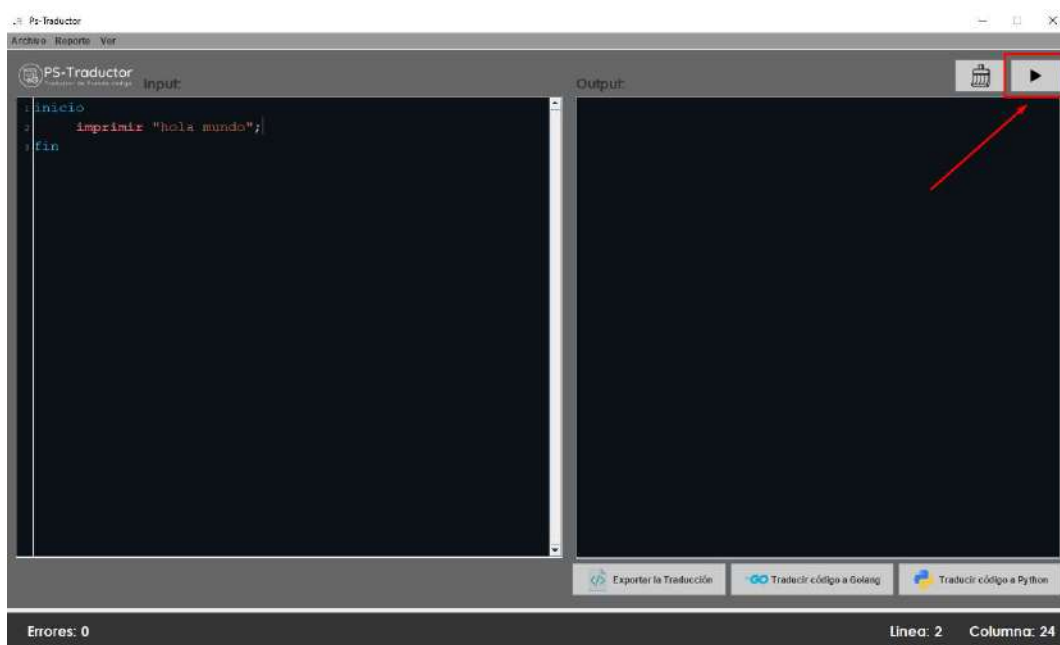
//impresión sin salto de línea
imprimir <expresion>;
//impresión con salto de línea
imprimir_nl <expresion>;

//Ejemplo de aplicación:

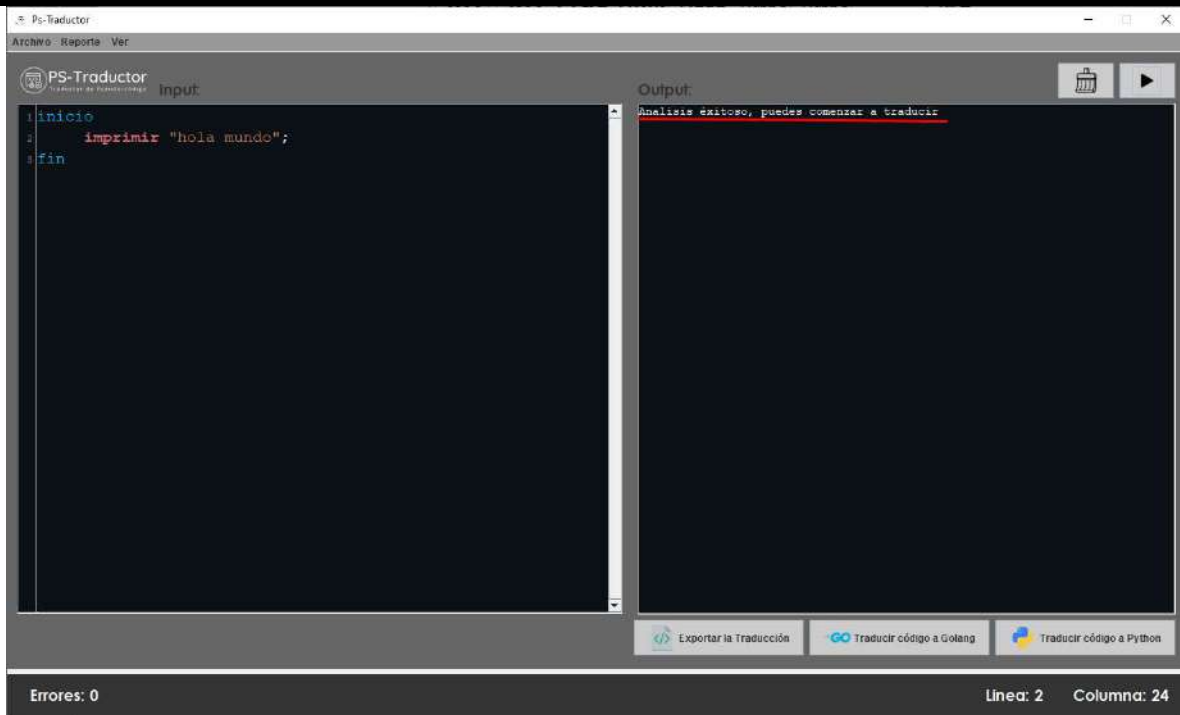
//impresión sin salto de línea
imprimir "hola mundo";
//impresión con salto de línea
imprimir_nl 1;
```

## 3. Traducción

Al finalizar de generar su pseudocódigo, primero tiene que ejecutarlo en el siguiente botón:



Posteriormente si aparece el siguiente mensaje:



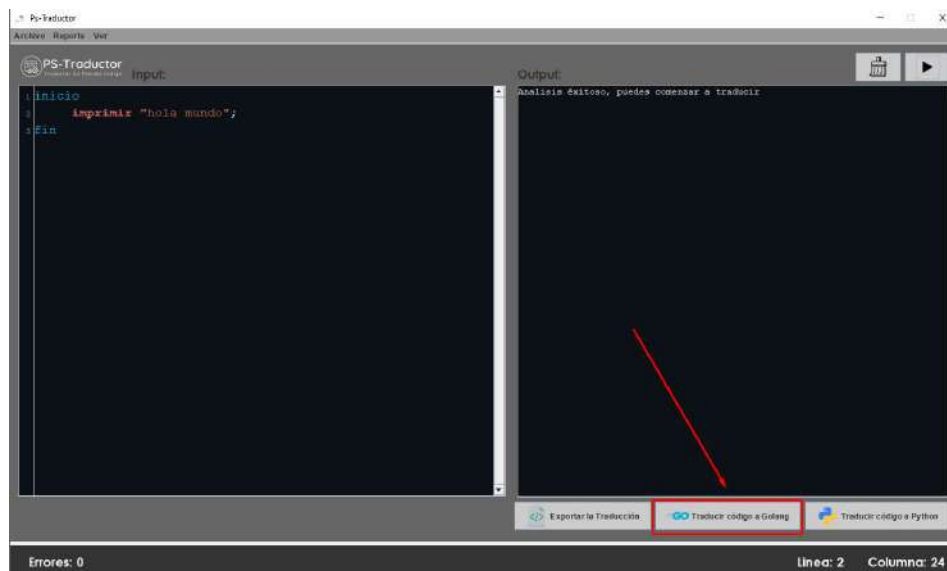
Entonces puede proceder a traducir, de lo contrario tiene que revisar sus errores los cuales aparecerán en el cuadro de texto derecho.

Luego pueden generar la traducción de los siguientes lenguajes de programación:

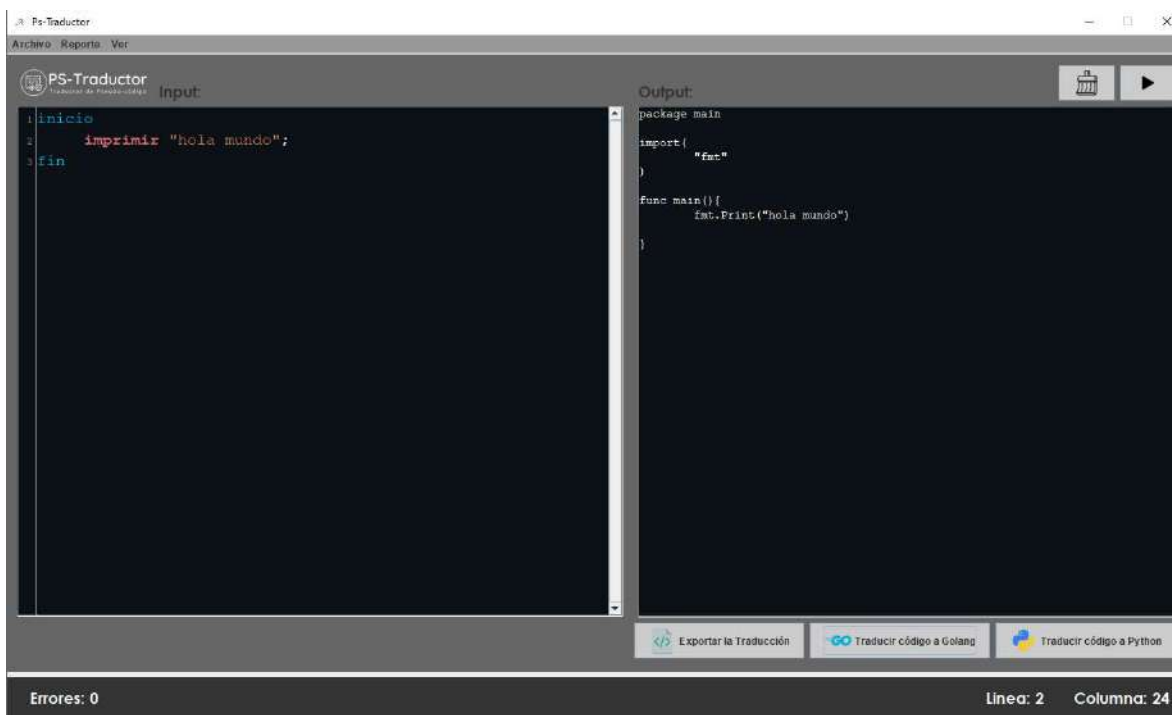
## a. Golang

### i. Mostrar la Traducción

Para poder visualizar la traducción a lenguaje Golang es necesario presionar el siguiente botón:

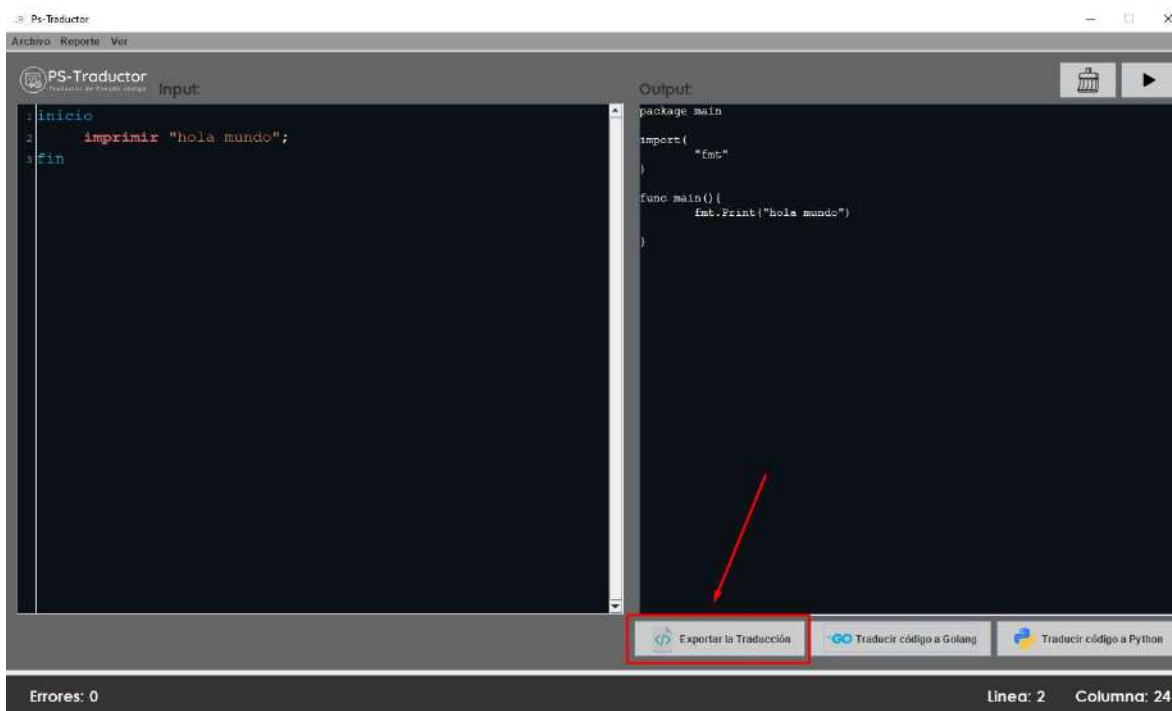


Le aparecerá la traducción en el cuadro de texto derecho:

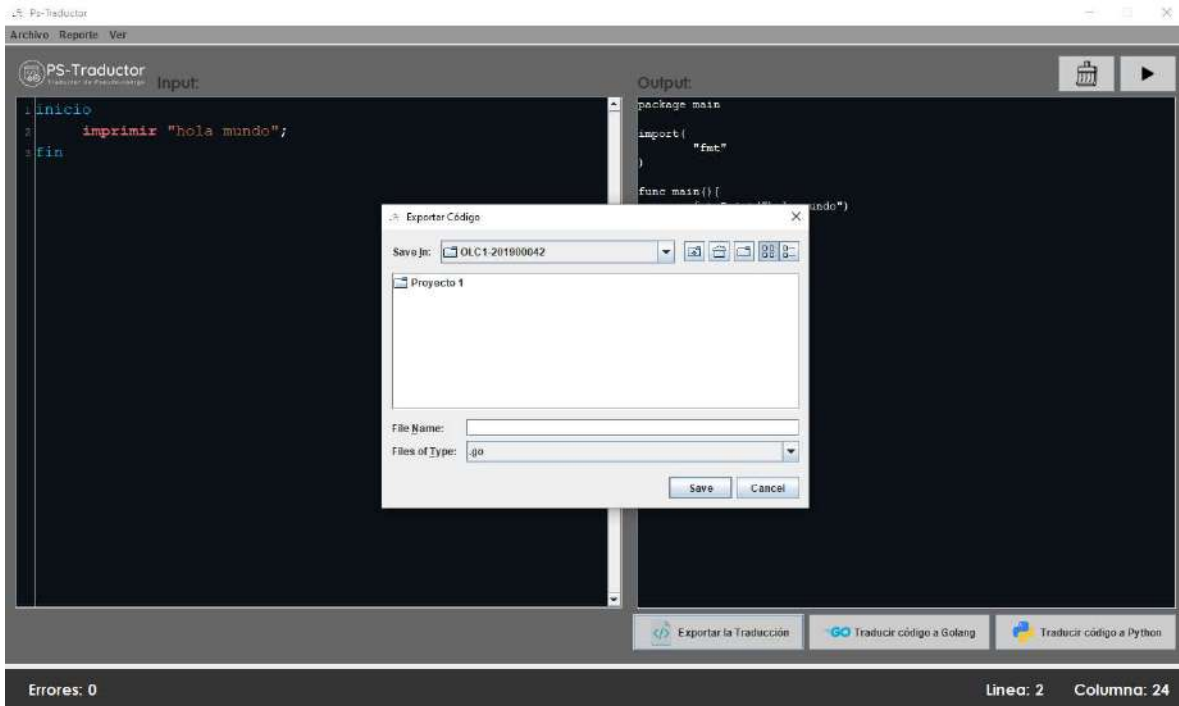


## ii. Exportar la Traducción

Al finalizar la traducción puede exportar el código presionando el siguiente botón:



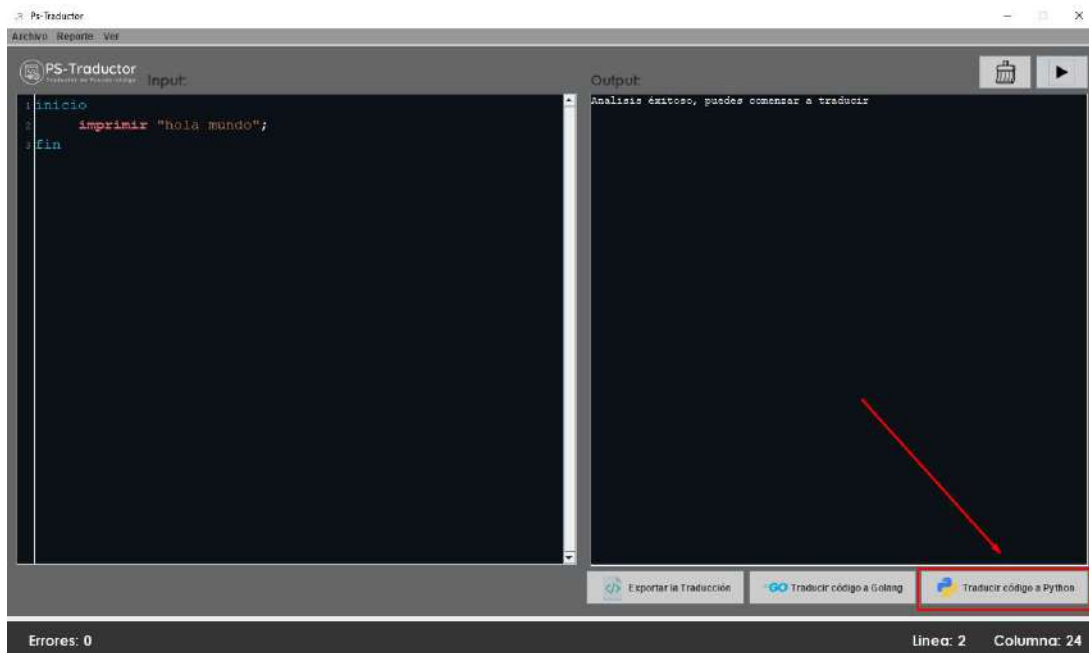
Posteriormente aparecerá una ventana emergente para poder exportar el código de salida y poder guardarlo en su dispositivo con la extensión .go.



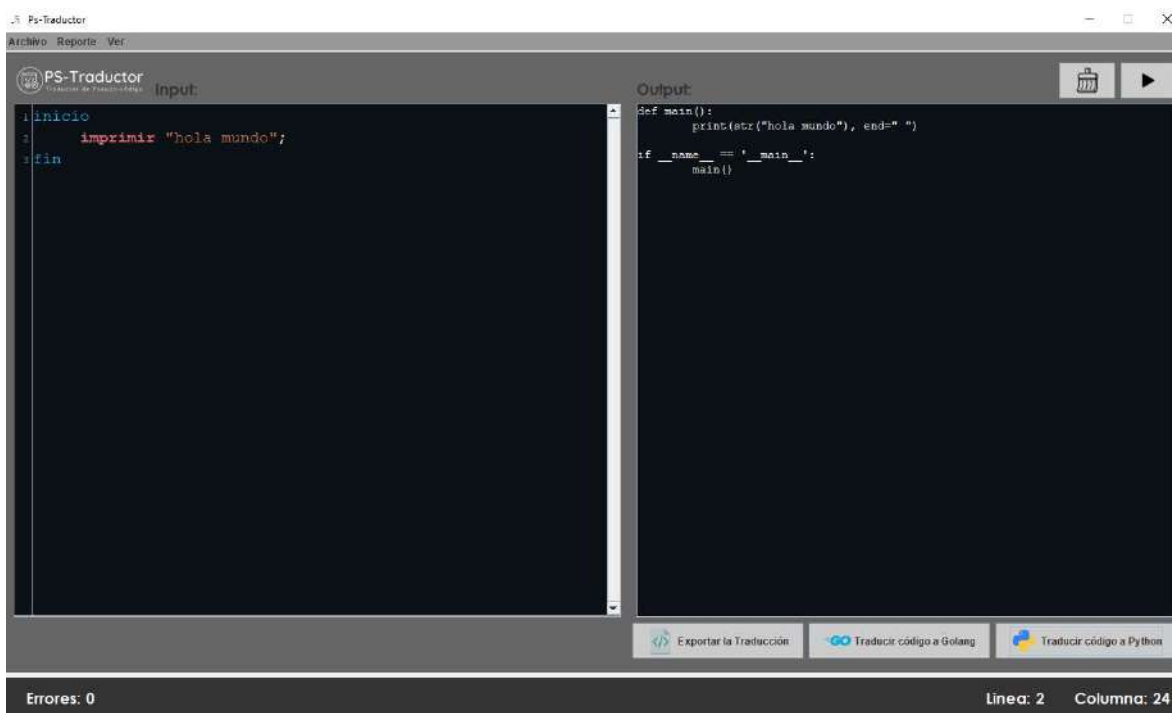
## b. Python

### i. Mostrar la Traducción

Para poder visualizar la traducción a lenguaje Python es necesario presionar el siguiente botón:

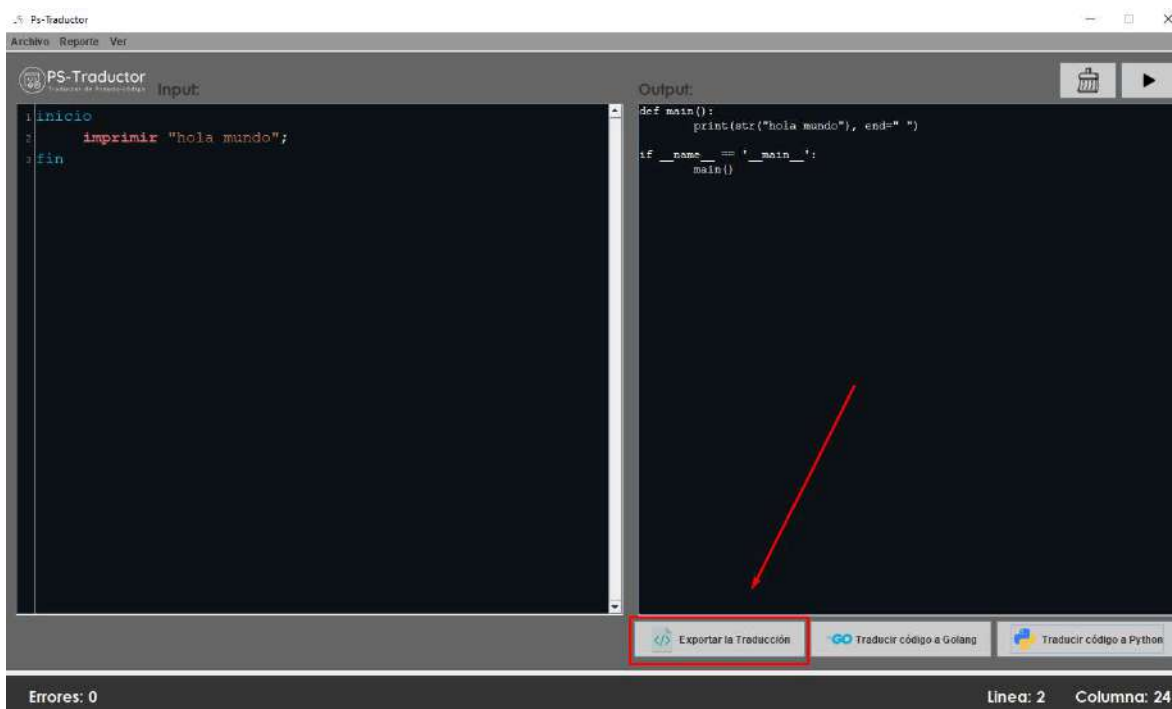


Le aparecerá la traducción en el cuadro de texto derecho:

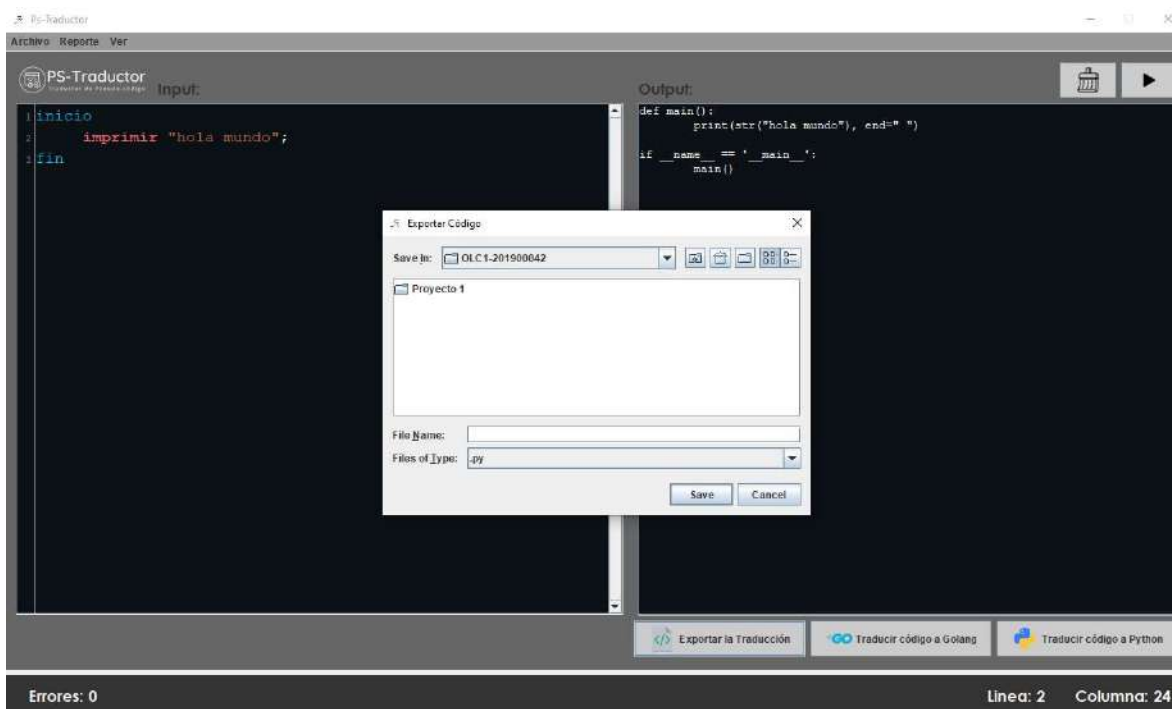


## ii. Exportar la Traducción

Al finalizar la traducción puede exportar el código presionando el siguiente botón:



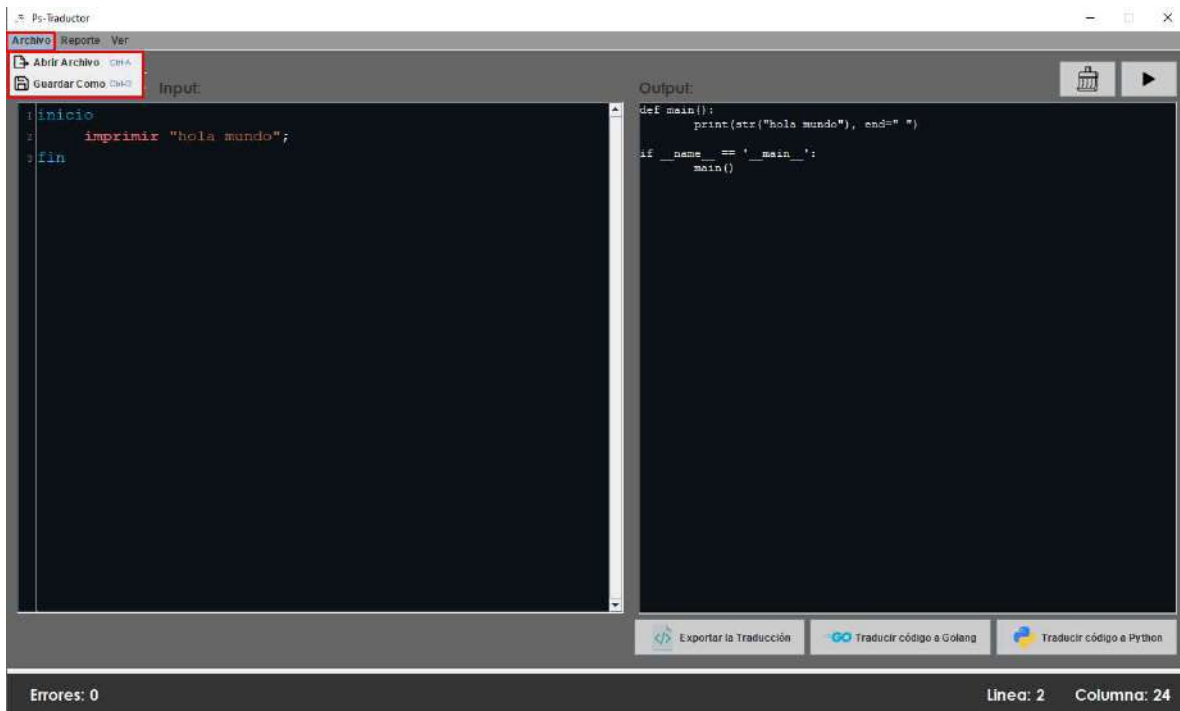
Posteriormente aparecerá una ventana emergente para poder exportar el código de salida y poder guardarlo en su dispositivo con la extensión .py.





## 4. Archivo

Este apartado se encuentra en la parte superior de la interfaz:



Y cuenta con las siguientes funcionalidades:

### a. Abrir Archivo

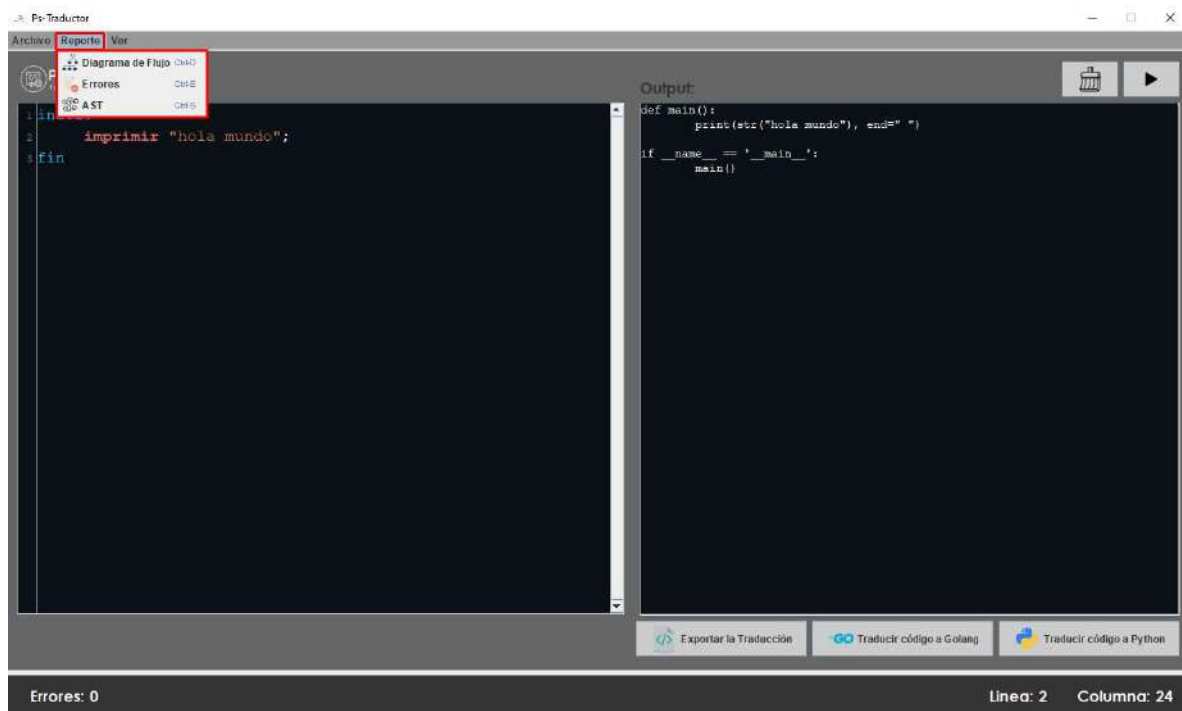
Esta opción le permite únicamente cargar archivos que contengan el lenguaje y la extensión OLC (.olc).

### b. Guardar Como

Esta opción le permite guardar el pseudocódigo que tiene escrito en la consola y guardarla en su dispositivo con la extensión olc.

## 5. Reporte

Este apartado se encuentra en la parte superior de la interfaz.



### i. Diagrama de Flujo

Este apartado se dedica a generar un diagrama de flujo de todo el pseudocódigo escrito en la interfaz y lo guarda como PDF con el nombre de Diagrama.pdf.

### ii. Errores

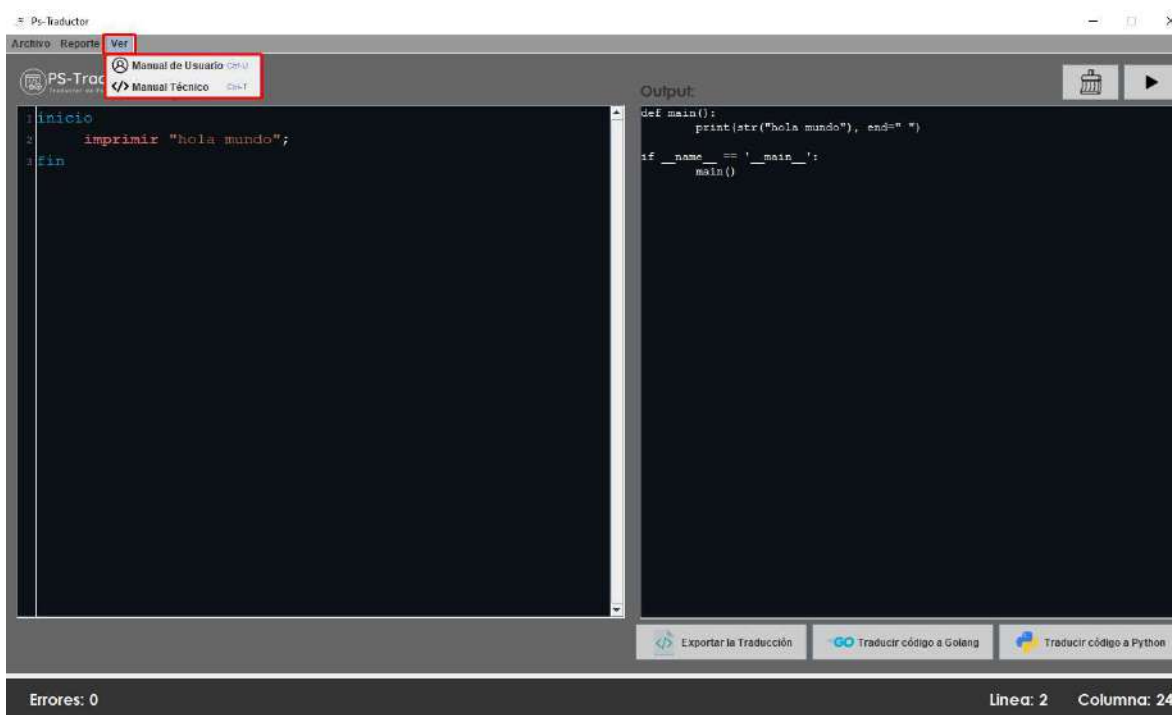
Este apartado genera un reporte de Errores Léxicos y sintácticos en html para visualizar la localización de los mismos y lo puede encontrar en su dispositivo como Reporte\_Errores.html.

### iii. AST

Este apartado genera un reporte del Abstract Syntax Tree en pdf del pseudocódigo escrito en la interfaz y lo puede encontrar como AST.pdf.

## 6. Ver

Este apartado se encuentra en la parte superior de la interfaz:



Dentro de este apartado puede seleccionar las opciones para poder visualizar el manual de Usuario o manual Técnico del sistema.

### III. Recomendaciones

Este programa hace validaciones léxicas y sintácticas así que para que tenga una traducción exitosa sin errores semánticos es necesario que siga las reglas que manejan los tipos de datos para evitar semejantes errores.