
# Proyecto 2
# MANUAL DE USUARIO

**RUDY AARÓN GOPAL MARROQUÍN GARCIA**

**201903872**

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Guatemala

2021

# Descripción General.

TYPESTY es un interprete sencillo capaz de traducir código y realizar una ejecución basada en typescript posee una interfaz gráfica en la cual nos permite crear y mantener diferentes pestañas, además de la posibilidad de cargar nuestros archivos .ty con código adentro.

El lenguaje mostrará en consola todas las sentencias impresas, también será capaz de indicar los errores sintácticos, léxicos y semánticos encontrados durante la ejecución indicando la posición en la cual se encuentra el problema

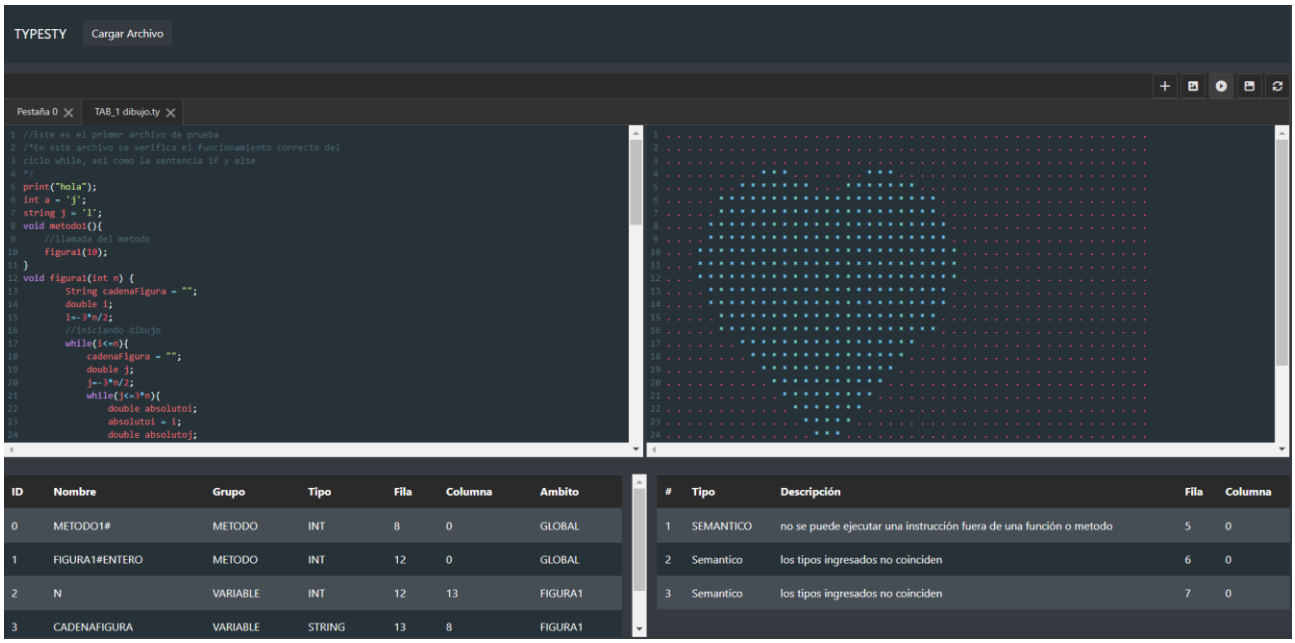
La aplicación nos permite guardar los archivos que escribamos, también nos permitirá visualizar el árbol AST generado por la entrada ingresada en el área de programación.

# Requerimientos mínimos del sistema

Para el uso correcto de la aplicación es necesario los siguientes componentes y programas:

- Memoria ram de 2gb o superior
- Espacio disponible en disco de 500 mb
- Angular version 11.2.9
- Typescript 4.2.4 o superior
- Nodejs 6.14.7
- graphviz version 2.38.0 (20140413.2041)
- Sistema operative Windows 10 x64

# EXPLICACIÓN DE APLICACIÓN




1. BOTON +: este botón nos permite agregar una nueva pestaña.
2. BOTON IMG: este botón generara el árbol AST de la entrada ingresada
3. BOTON PLAY: Este botón iniciara la ejecución del código indicado en la terminal
4. BOTON GUARDAR: Este botón guardara la pestaña actual en un archivo .py
5. BOTON RESET: Este botón eliminara todas las pestañas abiertas actualmente
6. Cargar Archivo: este botón nos permite abrir archivos .py

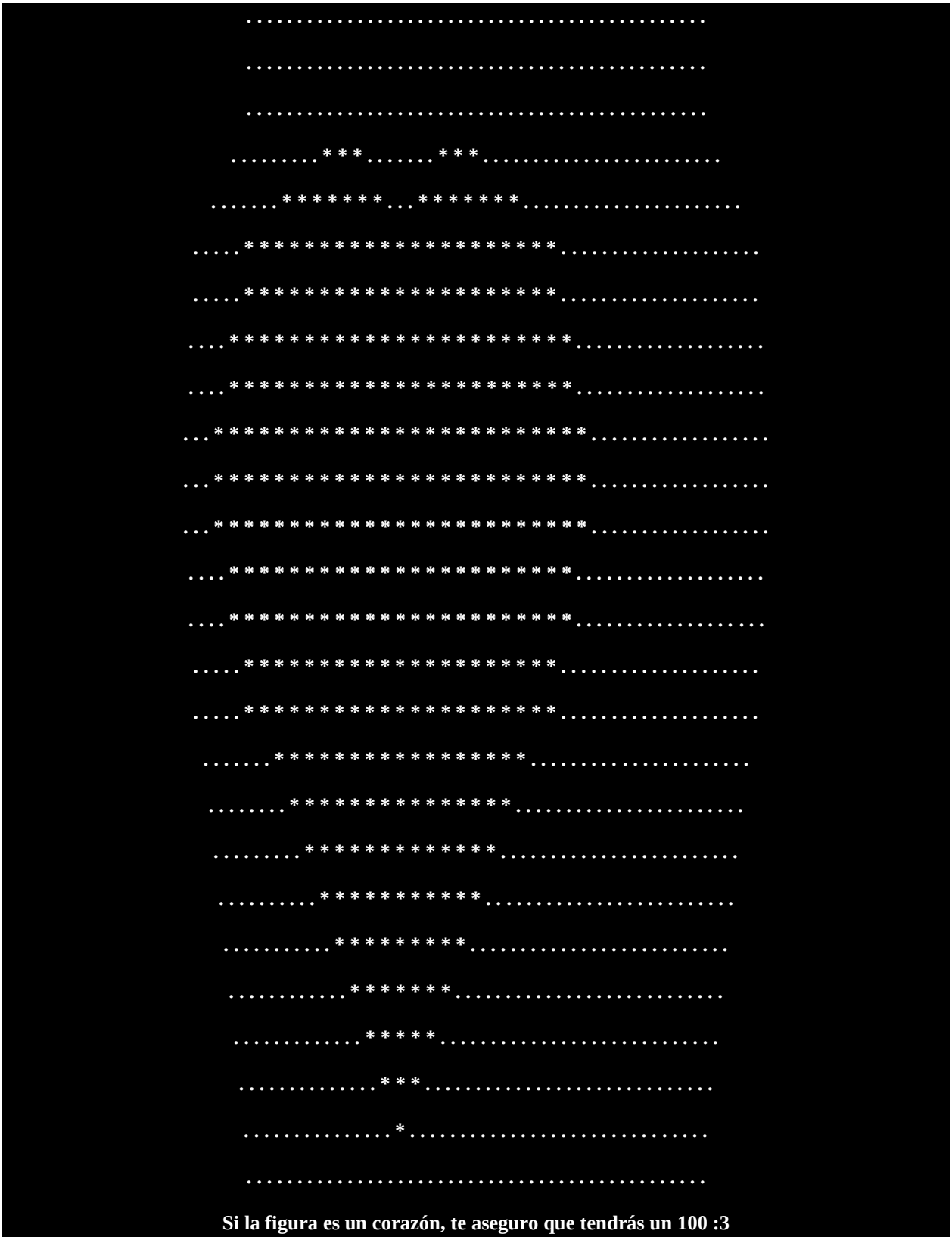
## EXPLICACIÓN DEL LENGUAJE

1. **Tipos:** la aplicación es capaz de manejar 5 tipos primitivos de datos entre los cuales tenemos char, int, double, boolean, string.
2. **Declaración de variables:** para declarar una variable “TIPO ID = EXPRESION” o “TIPO ID” con la primera declaración podemos asignarle a la variable un valor inicial, mientras que en la segunda forma obtendrá el valor por defecto de cada tipado.
3. **Asignación de variables:** para poder asignar una variable se utiliza la siguiente estructura “ID = EXPRESION” con dicha instrucción podemos asignar un nuevo valor a la variable;
4. **Declaración de vector y listas:** Para poder declarar un vector se utiliza la siguiente estructura “TIPO [] ID = NEW INT[EXP]” y la de una lista será “LIST <TIPO> ID = NEW LIST<TIPO>” existen diferentes variaciones de declaración, pudiendo realizarse la siguiente para el vector “TIPO [] ID = {EXP, EXP, … EXP}” y para una lista “LIST <TIPO> ID = TOCHARARRAY(EXP)”
5. **Asignación de vector y lista:** la asignación de un vector es la siguiente “ID[EXP] = EXP” y la asignación de una lista es “ID[[EXP]] = EXP”
6. **Agregar dato a lista;** para agregar un nuevo elemento se utiliza la función add “ID.ADD(EXP)”
7. **Ciclos**
    a. **FOR:** la estructura de un for es la siguiente “FOR(DECLARACION/ASIGNACION; CONDICION; ACTUALIZACION){ INSTRUCIONES}”
    b. **WHILE:** la estructura de un while es la siguiente “WHILE(CONDICION){INSTRUCCIONES}”
    c. **DO WHILE:** la estructura de un while es la siguiente “DO { INSTRUCIONES} WHILE(CONDICION)”
8. **Nativas:** existen diferentes funciones nativas con la estructura “FUNCION(EXP)” en las cuales podemos mencionar, toString, length, round, truncate, toUpper, toLower
9. **Funciones y métodos**
    a. **funciones:** los métodos tienen la siguiente estructura “TIPO ID ( PARAMETROS ){ INSTRUCIONES }”
    b. **métodos:** los métodos tienen la siguiente estructura “VOID ID ( PARAMETROS ) { INSTRUCCIONES } “
10. **Exec:** exec es una forma de llamar una función y a la vez indicara cual es nuestra función principal a ejecutar, solo puede existir uno por terminal “EXEC ID ( PARAMETROS ) “
11. **LLAMADAS:** para llamar una función o un método se utiliza la siguiente estructura “ ID ( PARAMETROS )”

## EJEMPLOS DE ESTRUCTURA

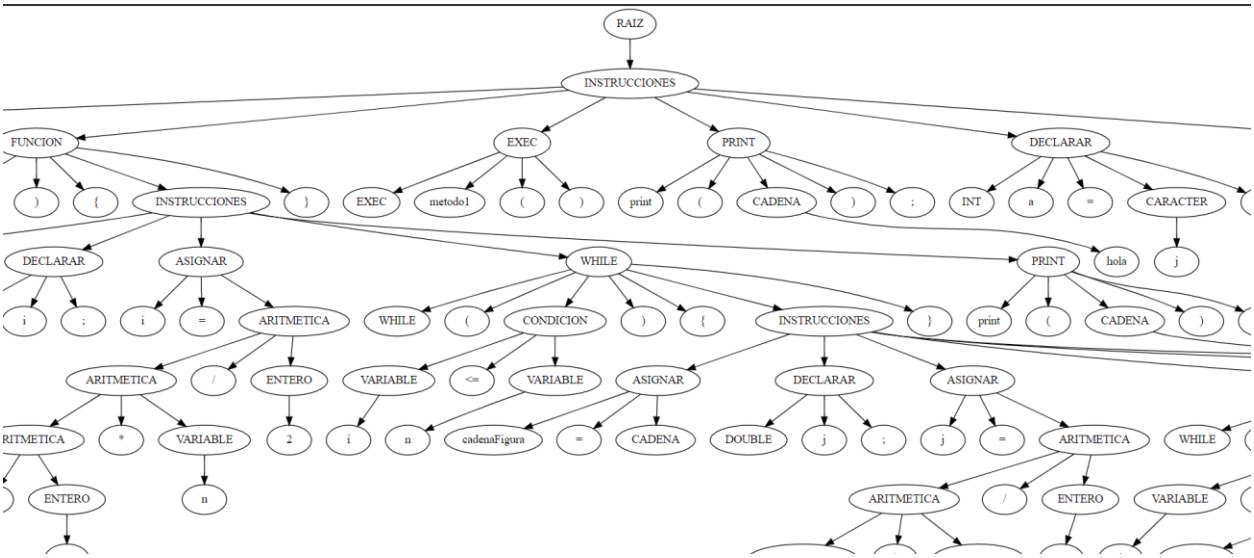
```
void metodo1(){
   //llamada del metodo
   figura1(10);
}
void figura1(int n) {
      String cadenaFigura = "";
      double i;
      i=-3*n/2;
      //iniciando dibujo
      while(i<=n){
         cadenaFigura = "";
         double j;
         j=-3*n/2;
         while(j<=3*n){
            double absolutoi;
            absolutoi = i;
            double absolutoj;
            absolutoj = j;
            if(i < 0)
            {
               absolutoi = i * -1;
            }
            if(j < 0)
            {
               absolutoj = j * -1;
            }
            if((absolutoi + absolutoj < n)
                  || ((-n / 2 - i) * (-n / 2 - i) + (n / 2 - j) * (n / 2 - j) <= n * n / 2)
                  || ((-n / 2 - i) * (-n / 2 - i) + (-n / 2 - j) * (-n / 2 - j) <= n * n / 2)) {
               cadenaFigura = cadenaFigura + "* ";
            }
            else
            {
               cadenaFigura = cadenaFigura + ". ";
            }
            j=j+1;
         }
         print(cadenaFigura);
         i=i+1;
      }
      print("Si la figura es un corazón, te aseguro que tendrás un 100 :3");
   }

exec metodo1();
```

**SALIDA**




**Si la figura es un corazón, te aseguro que tendrás un 100 :3**

# ARBOL AST


RAIZ
INSTRUCCIONES
FUNCION
EXEC
PRINT
DECLARAR
)
{
INSTRUCCIONES
}
EXEC
metodo1
(
)
print
(
CADENA
)
;
INT
a
=
CARACTER
DECLARAR
ASIGNAR
WHILE
PRINT
hola
j
i
;
i
=
ARITMETICA
WHILE
(
CONDICION
)
{
INSTRUCCIONES
}
print
(
CADENA
)
ARITMETICA
/
ENTERO
VARIABLE
<=
VARIABLE
ASIGNAR
DECLARAR
ASIGNAR
RITMETICA
*
VARIABLE
2
i
n
cadenaFigura
=
CADENA
DOUBLE
j
;
j
=
ARITMETICA
WHILE
ENTERO
n
ARITMETICA
/
ENTERO
VARIABLE