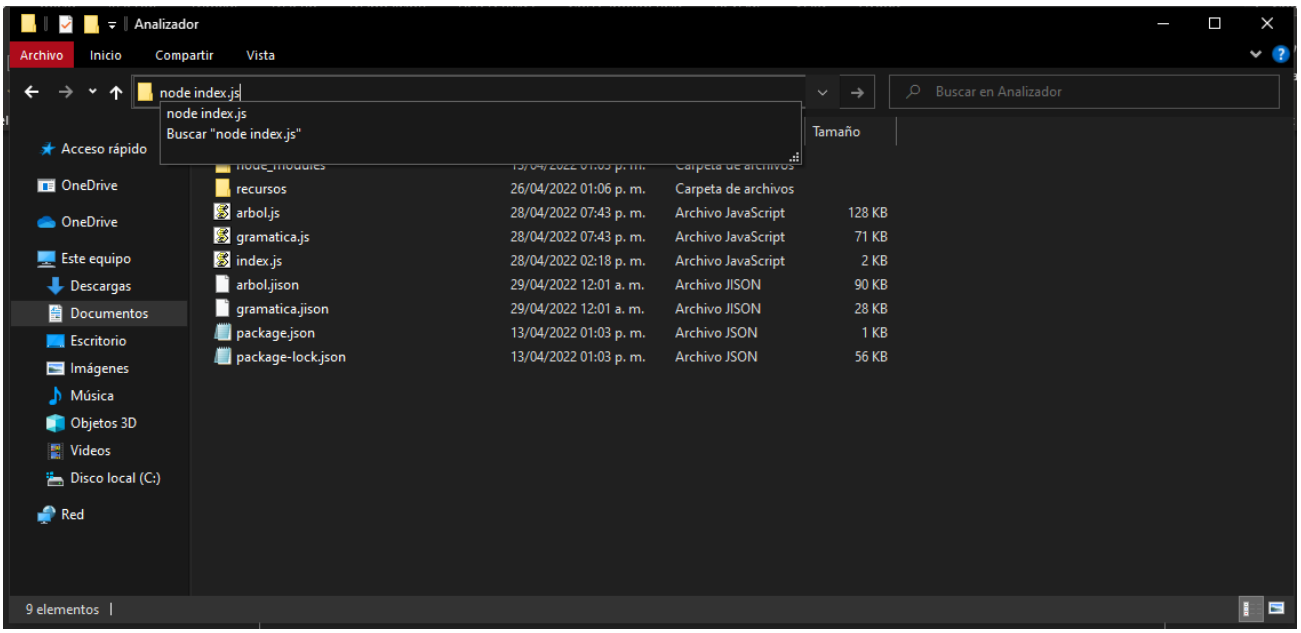
CompScript

Manual de Usuario

CompScript

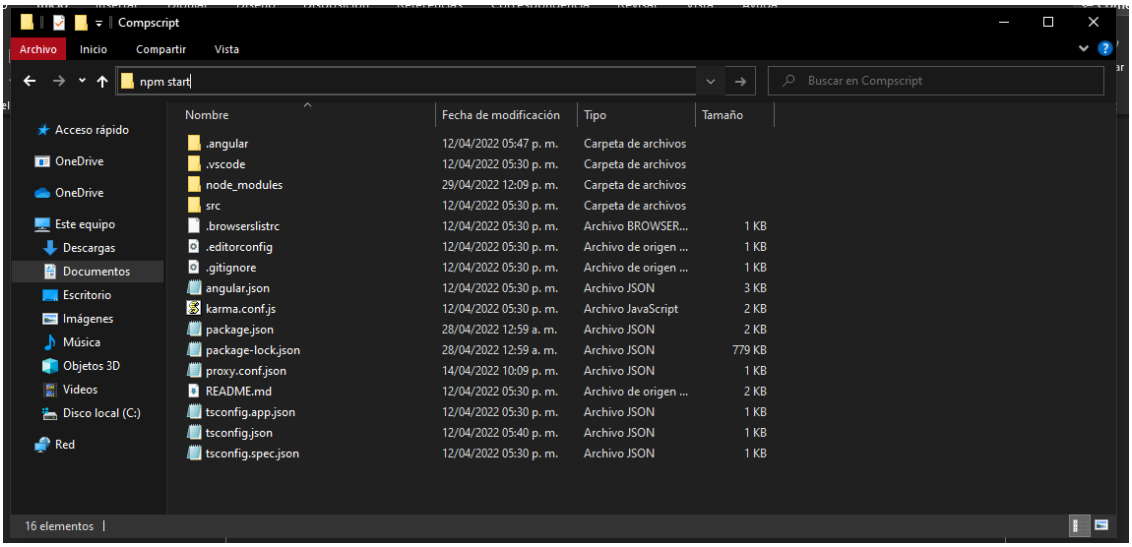
CompScript es un lenguaje de programación orientado a principiantes en el area de programacion. El editor de texto se encuentra alojado en linea. Si asi lo desea, puede descargar los archivos fuente del proyecto y correrlo desde un servidor local. Para ello, es indispensable que cuente con Node.js y Angular.CLI instalado en su equipo de trabajo.

Para ejecutar el servidor, ingrese a la carpeta donde se encuentre ubicado y en la barra de navegacion, escriba el comando node index.js.



Si todo fue hecho correctamente, podra visualizar la consola del servidor listo para aceptar peticiones.

Para ejecutar la aplicación web, de forma similar, escriba el comando npm start en la barra de navegación para correr la aplicación web.



En su navegador de preferencia, ingrese localhost:4200 y podrá visualizar la aplicación.

El funcionamiento de la aplicación es bastante simple: Ingrese el texto dentro del area de texto y pulse el boton analizar para ejecutar las instrucciones: La consola en la parte inferior mostrará los errores o cualquier cosa que haya mandado con la instrucción print.

En la barra superior puede dirigirse al apartado reportes. Dentro puede revisar un resumen de la ejecución del codigo.

Si por alguna razón, el codigo no se ejecutó, en el apartado de errores puede ver que fue lo que salió mal durante la ejecución. En la tabla de simbolos podra ver cada uno de los metodos y variables declaradas durante la ejecución y el valor que adopto durante la misma. En el apartado AST, puede ver el arbol de sintaxis de la ejecución.

ACERCA DEL LENGUAJE:

- *Case Insensitive: El lenguaje no distinguirá entre mayúsculas o minúsculas.*

- *Comentarios: Los comentarios son una forma elegante de indicar que función tiene cierta sección del código que se ha escrito simplemente para dejar algún mensaje en específico*. Existen dos tipos:
  - ✓ *Comentarios de una línea: Estos comentarios deberán comenzar con // y terminar con un salto de línea*
  - ✓ *Comentarios de una línea: Estos comentarios deberán comenzar con /* y terminar con */.*

- *Tipos de Datos. Ver Tabla*

TIPO	DEFINICION	DESCRIPCION	EJEMPLO
Entero	Int	Este tipo de datos aceptará solamente números enteros.	1, 50, 100, 25552, etc.
Doble	Double	Admite valores numéricos con decimales.	1.2, 50.23, 00.34, etc.
Booleano	Boolean	Admite valores que indican verdadero o falso.	True, false
Caracter	Char	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples. ' '	'a', 'b', 'c', 'E', 'Z', '1', '2', '^', '%', ')', '=', '!', '&', '/', '\\', '\n', etc.
Cadena	String	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. " "	"cadena1", "-- ** cadena 1"

- Suma: Es la operación aritmética que consiste en realizar la suma entre dos o más valores. El símbolo para utilizar es el signo más (+). Las combinaciones permitidas se pueden ver en la tabla siguiente.

+	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble	Entero	Entero	Cadena
Doble	Doble	Doble	Doble	Double	Cadena
Boolean	Entero	Doble			Cadena
Caracter	Entero	Doble		Cadena	Cadena
Cadena	Cadena	Cadena	Cadena	Cadena	Cadena

- Resta: Es la operación aritmética que consiste en realizar la resta entre dos o más valores. El símbolo por utilizar es el signo menos (-). Las combinaciones permitidas se pueden ver en la tabla siguiente.

-	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble	Entero	Entero	
Doble	Doble	Doble	Doble	Doble	
Boolean	Entero	Doble			
Caracter	Entero	Doble			
Cadena					

- Multiplicación: Operación aritmética que consiste en sumar un número (multiplicando) tantas veces como
  indica otro número (multiplicador). El sino para representar la operación es el asterisco (*).

*	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble		Entero	
Doble	Doble	Doble		Doble	
Boolean					
Caracter	Entero	Doble			
Cadena					

- División: Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal (/)

/	Entero	Doble	Boolean	Caracter	Cadena
Entero	Doble	Doble		Doble	
Doble	Doble	Doble		Doble	
Boolean					
Caracter	Doble	Doble			
Cadena					

- Potencia: Es una operación aritmética de la forma a^b donde a es el valor de la base y b es el valor del exponente que nos indicará cuantas veces queremos multiplicar el mismo número. Por ejemplo 5^3, a=5 y b=3 tendríamos que multiplicar 3 veces 5 para obtener el resultado final; 5x5x5 que da como resultado 125. Para realizar la operación se utilizará el signo (^).

^	Entero	Doble	Boolean	Caracter	Cadena
Entero	Entero	Doble			
Doble	Doble	Doble			
Boolean					
Caracter					
Cadena					

- Módulo: Es una operación aritmética que obtiene el resto de la división de un numero entre otro. El signo para utilizar es el porcentaje %

%	Entero	Doble	Boolean	Caracter	Cadena
Entero	Doble	Doble			
Doble	Doble	Doble			
Boolean					
Caracter					
Cadena					

- Negación Unaria: Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original. Se utiliza el símbolo menos (-).

-num	Resultado
Entero	Entero
Doble	Doble
Boolean	
Caracter	
Cadena	

- Operadores Relacionales
  Son los símbolos que tienen como finalidad comparar expresiones, dando como resultado valores booleanos.

OPERADOR	DESCRIPCIÓN	EJEMPLO
==	Igualación: Compara ambos valores y verifica si son iguales: - Iguales= True - No iguales= False	1 == 1 "hola" == "hola" 25.5933 == 90.8883 25.5 == 20
!=	Diferenciación: Compara ambos lados y verifica si son distintos. - Iguales= False - No iguales= True	1 != 2, var1 != var2 25.5 != 20 50 != 'F' "hola" != "hola"
<	Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo. - Derecho mayor= True - Izquierdo mayor= False	(5/(5+5))<(8*8) 25.5 < 20 25.5 < 20 50 < 'F'
<=	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo. - Derecho mayor o igual= True - Izquierdo mayor= False	55+66<=44 25.5 <= 20 25.5 <= 20 50 <= 'F'
>	Mayor que: Compara ambos lados y verifica si el izquierdo es mayor que el derecho. - Derecho mayor= False - Izquierdo mayor= True	(5+5.5)>8.98 25.5 > 20 25.5 > 20 50 > 'F'
>=	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho. - Derecho menor o igual= True - Izquierdo menor= False	5-6>=4+6 25.5 >= 20 25.5 >= 20 50 >= 'F'

- Operador Ternario: El operador ternario es un operador que hace uso de 3 operandos para simplificar la instrucción 'if' por lo que a menudo este operador se le considera como un atajo para la instrucción 'if'. El primer operando del operador ternario corresponde a la condición que debe de cumplir una expresión para que el operador retorne como valor el resultado de la expresión segundo operando del operador y en caso de no cumplir con la expresión el operador debe de retornar el valor de la expresión del tercer operador.

```
//Ejemplo del uso del operador ternario

int edad = 18;
boolean banderaedad = false;
banderaedad = edad > 17 ? true : false;
```

- Operadores Lógicos: Son los símbolos que tienen como finalidad comparar expresiones a nivel lógico (verdadero o falso). A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

OPERADOR	DESCRIPCION	EJEMPLO
||	OR: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve verdadero en otro caso retorna falso	(55.5)|| bandera==true Devuelve true
&&	**AND**: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve verdadero en otro caso retorna falso	(flag1) && ("hola" == "hola") Devuelve true
!	NOT: Devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá falso, de lo contrario retorna verdadero.	!var1 Devuelve falso

- Signos de Agrupación: Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por ( ).
- Finalización de instrucciones: para finalizar una instrucción se utilizara el signo punto y coma (;).

- Declaración y asignación de variables: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador. Las variables podrán ser declaradas global y localmente. Durante la declaración de variables también se tendrá la opción de poder crear múltiples variables al mismo tiempo, al crear múltiples variables al mismo tiempo se tendrá la opción de crear todas las variables con un mismo valor, para ello se realizará una asignación al final del listado de las variables, en caso de no indicar esta asignación se dejará el valor por defecto para cada variable

```
//Ejemplos
int numero;
int var1, var2, var3;
string cadena = “hola”;
char var_1 = ‘a’;
boolean verdadero;
boolean flag1, flag2, flag3 = true;
char ch1, ch2, ch3 = ‘R’;
```

- Casteos; Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo. Para hacer esto, se colocará la palabra reservada del tipo de dato destino ente paréntesis seguido de una expresión.

```
//Ejemplos
int edad = (int) 18.6; //toma el valor entero de 18
char letra = (char) 70; //tomar el valor 'F' ya que el 70 en ascii es F
double numero = (double) 16; //toma el valor 16.0
```

- Incremento y Decremento: Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de un valor, de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria.

```
//Ejemplos
int edad = 18;
edad++; //tiene el valor de 19
edad--; //tiene el valor 18

int anio=2020;
anio = 1 + anio++; //obtiene el valor de 2022
anio = anio--;  //obtiene el valor de 2021
```

- Vectores: Los vectores son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string. El lenguaje permitirá únicamente el uso de arreglos de una o dos dimensiones.
    - ✓ Declaración tipo 1: En esta declaración, se indica por medio de una expresión numérica del tamaño que se desea el vector, además toma los valores por default para cada tipo.
    - ✓ Declaración tipo 2: En esta declaración, se indica por medio de una lista de valores separados por coma, los valores que tendrá el vector, en este caso el tamaño del vector será el de la misma cantidad de valores de la lista.

```
//Ejemplo de declaración tipo 1
Int vector1[ ] = new int[4]; //se crea un vector de 4 posiciones, con  0 en cada posición
Char vectorDosd[ ][ ] = new char [4] [4] ; // se crea un vector de dos dimensiones de 4x4

//Ejemplo de declaración tipo 2
string vector2[ ] = ["hola", "Mundo"]; //vector de 2 posiciones, con "Hola" y "Mundo"

char vectordosd2 [ ][ ] = [[ 0 ,0],[0 , 0]]; // vector de dos dimensiones con valores de 0 en
cada posición
```

```
//Ejemplo de acceso
string vector2[ ] = ["hola", "Mundo"]; //creamos un vector de 2 posiciones de tipo string
string valorPosicion = vector2[0]; //posición 0, valorPosicion = “hola”

Char vectorDosd[ ][ ] = new char [4] [4] ; // creamos vector de 4x4
Char valor = vectorDosd[0][0]; // posición 0,0
```

- if: La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia.

- Switch Case: Switch case es una estructura utilizada para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos if.

```
if (x > 50)
 {
    Print("Mayor que 50");
    //Más sentencias
}
else if (x <= 50 && x > 0)
{
    Print ("Menor que 50");if (x
    > 25)
    {
        Print("Número mayor que 25");
        //Más sentencias
    }
    else
    {
        Print("Número menor que 25");
        //Más sentencias
    }
    //Más sentencias
}
else
{
    Print("Número negativo");
    //Más sentencias
}
```

- While: El ciclo o bucle While, es una sentencia que ejecuta una secuencia de instrucciones mientras la condición de ejecución se mantenga verdadera.

```
//Ejemplo de cómo se implementar un ciclo while

while (x<100){
        if (x > 50)
        {
                Print("Mayor que 50");
                //Más sentencias
        }
        else
        {
                Print("Menor que 100");
                //Más sentencias
        }
        X++;
        //Más sentencias

 }
```

- For: El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella.

```
//Ejemplo 1: declaración dentro del for con incremento
for ( int i=0; i<3;i++ ){
        Println("i="+i)
        //más sentencias
}
/*RESULTADO
  i=0
  i=1
  i=2
*/

//Ejemplo 2: asignación de variable previamente declarada y decremento por asignación
for ( i=5; i>2;i=i-1 ){
        Print("i="+i+"\n")
        //más sentencias
}
/*RESULTADO
  i=5
  i=4
  i=3
*/
```

- Do-While: El ciclo o bucle Do-While, es una sentencia que ejecuta al menos una vez el conjunto de instrucciones que se encuentran dentro de ella y que se sigue ejecutando mientras la condición sea verdadera.

- Break: La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo.

```
//Ejemplo en un ciclo for
for(int i = 0; i < 9; i++){
    if(i==5){
        Println("Me salgo del ciclo en el numero " + i);
        break;
    }
    Println(i);
}
```

- Continue: La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error.

```
//Ejemplo en un ciclo for
for(int i = 0; i < 9; i++){
    if(i==5){
        Println("Me salte el numero " + i);
        continue;
    }
    Println(i);
}
```

- Return: La sentencia return finaliza la ejecución de un método o función y puede especificar un valor para ser devuelto a quien llama a la función.

```
//Ejemplos
//--> Dentro de un metodo
mi_metodo(): void{
    int i;
    for(i = 0; i < 9; i++){
        if(i==5){
            return; //se detiene
        }
        Print(i);
    }
}
//--> Dentro de una función
sumar(int n1, int n2): int {
    int n3;
    n3 = n1+n2;
    return n3;      //retorno el valor
}
```

- Función Print: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter. Esta función no concatena un salto de línea al final del contenido que recibe como parámetro.

```
//Ejemplo
Print("Hola mundo!!");
Print("Sale compi \n" + valor + "!!");
Print(suma(2,2));
/*
Salida Esperada:
Hola Mundo!!Sale compi
25!!4
*/
// 25 es el valor almacenado en la variable "valor"
```

- Función Println: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero,doble, booleano, cadena y carácter. Esta función concatena un salto de línea al finalizar el contenido que recibe como parámetro.

```
//Ejemplo
Println("Hola mundo!!");
Println("Sale compi \n" + valor + "!!");
Println(suma(2,2));
/*
Salida Esperada:
Hola Mundo!!
Sale compi
25!!
4
*/
// 25 es el valor almacenado en la variable "valor"
```

- Función toLower: Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas

```
//Ejemplo
string cad_1 = toLower("hOla MunDo");  // cad_1 = "hola mundo"
string cad_2 = toLower("RESULTADO = " + 100);  // cad_2 = "resultado = 100"
```

- Run: Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia RUN para indicar que método o función es la que iniciará con la lógica del programa.
-

```
//Ejemplo  1
funcion1():void{
    Print("hola");
}

run funcion1();
/*RESULTADO
    hola
*/

//Ejemplo 2
funcion2(string mensaje):void{
    Print(mensaje);
}

Run funcion2("hola soy un mensaje");
/*RESULTADO
    Hola soy un mensaje
*/
```