

Proyecto de Compiladores

**TUBY**

Ruben Alejandro Hernandez Gonzalez A01175209

Andres Marcelo Garza Cantu A00814236

3 de Mayo del 2017

**Ing. Elda Quiroga**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Ruben Alejandro Andres Marcelo**

**Hernandez Gonzalez Garza Cantu**

[**a.1) Visión, Objetivos y Alcance del Proyecto**](#_lodlcwllwc29) **4**

[VISIÓN/PROPÓSITO DEL PROYECTO](#_rapar3xu1ef2) 4

[OBJETIVO DEL LENGUAJE CON ÁREA DE APLICACIÓN](#_hb29woho7d6u) 4

[**a.2) Análisis de Requerimientos y Casos de Uso Generales**](#_5e3a6qlt73bh) **5**

[Requerimientos](#_5b8anw82rltv) 5

[Casos de uso:](#_6nrm7dnc0z1x) 5

[**a.3) Descripción de los principales Test Cases**](#_1fk3dndaq8d6) **5**

[Test Cases](#_twek24o07d53) 5

[**a.4) Descripción del proceso**](#_6xgf2memmgah) **5**

[Descripción del proceso general seguido para el desarrollo del proyecto](#_41jia5i2ger5) 5

[Bitácora](#_14azpg5aiq59) 6

[Reflexión Ruben Alejandro Hernandez Gonzalez](#_1xmldihzxjea) 7

[Reflexión Andrés Marcelo Garza Cantú](#_8bgl7kb2w21o) 7

[**b.1) Nombre del Lenguaje**](#_y9dl1te1oryb) **7**

[**b.2) Principales características del lenguaje**](#_mprk43qjzjof) **7**

[**b.3) Posibles errores de compilación/ejecución**](#_9rr2guel5qnl) **8**

[Errores en compilación](#_vngix1vlp4o2) 8

[Errores en ejecución](#_h4vwusd08zi6) 8

[**c.1) Equipo de computo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto**](#_nq9djzlkufic) **8**

[**c.2) Descripción de Análisis de Léxico.**](#_m8q33g6whaeo) **8**

[Tokens asociados al lenguaje de programación](#_bcx75crk3dzi) 8

[**c.3) Descripción de Análisis de Sintaxis**](#_ropdhxkeg6ji) **9**

[**c.4) Descripción de Generación de Código Intermedio**](#_as5aoump88wf) **10**

[Asignación](#_kxogak5laa9i) 10

[Bloque](#_60bbcgbixilx) 11

[Cambio](#_yofesynohs3o) 12

[Ciclo](#_3d1kt883ur9i) 12

[Condición](#_qnsmnlkpgiri) 12

[Declaración](#_jppevwd5llwp) 13

[Entrada](#_g87dr5rpqm6c) 13

[Expresión](#_3ys3leozdm8x) 14

[Expra](#_ksfo7uvxa79w) 14

[Exp](#_net7igz3iec8) 14

[Término](#_4ksdt9ncwywk) 14

[Factor](#_xd4c4yvmriio) 15

[Funcion](#_qv1t6pkxrsbt) 15

[Llamada-Función](#_r7t00kez7yww) 16

[Llamada-Principal](#_9h5tfv5k932l) 16

[Llamada-Ids](#_4mt6g3xc4k5h) 16

[Programa](#_c80rqbhbvvuo) 16

[Salida](#_t0imqiqfjm3x) 17

[Terminal](#_x8o5krmmbr13) 17

[Tipo](#_8hui7bbtjnhh) 17

[Valor Salida](#_8n47pbgvwvg) 18

[**c.5) Descripción del proceso de Administración de memoria**](#_50txfzsgrw8z) **19**

[**d.1) lenguaje y utilerías especiales utilizadas.**](#_aq3s3zkzk4ed) **19**

[**d.2) Descripción detallada del proceso de Administración de Memoria en ejecución.**](#_v86qsuuhx0x2) **21**

[**e.1) Pruebas que comprueben el funcionamiento del lenguaje**](#_5z8zc45mzuqp) **24**

[Prueba de manejo de funciones](#_qla5aupzmieg) 24

[Prueba de matrices y ciclos](#_ymd4f5t6xl2a) 25

[Prueba de Condicion](#_byrqk0ylq5z3) 27

[Prueba de Ciclo Mientras](#_ikifl1j3myim) 28

[Prueba Find con valor de Entrada](#_agkphgezg7z2) 29

[Prueba Find con valores Estáticos](#_61h3zgbkgpaj) 30

[Prueba de Sort de Arreglos](#_2n5iv8gazlwo) 32

[**Pruebas de error**](#_ktle8qhnf8ul) **35**

[Error de funciones con el mismo nombre.](#_jug80lik01zs) 35

[Error de Doble parametro.](#_okwr49uz0beu) 35

[Error de falla doble.](#_ys3zgphl8th6) 36

[Codigo de interes:](#_g1aw9a1cmnbc) 36

[Python](#_mtrviab4k4q6) 36

[PLY](#_aokj4rbum9qo) 37

[PyCharm](#_8d22kc4hfiyo) 37

[Blockly](#_dqutb2y6zdg) 37

[**Identificadores**](#_rw756uqlox97) **37**

[Variables numéricas](#_rwol4xmop3pi) 38

[Variables de texto](#_xcg9oppgp886) 38

[Variables Booleanas](#_dikc8kb4p041) 38

[**Palabras reservadas**](#_v8xuzggfya9s) **38**

[**Diseño de programas**](#_qixvy3jqi2mj) **39**

[**Expresiones**](#_5i82kx7buqlq) **39**

[**Condiciones**](#_bdk0qymiylgf) **39**

[**Ciclos**](#_7ammh2dbx2pw) **39**

[**Funciones**](#_fa0jt44nf9fc) **39**

Documentación

a) Descripción del Proyecto

## a.1) Visión, Objetivos y Alcance del Proyecto

#### VISIÓN/PROPÓSITO DEL PROYECTO

Nuestra visión de este proyecto es el enseñarles a usuarios que hablan en español a familiarizarse con lógica computacional. Debido a que en la población mexicana, [[1]](#footnote-0) solo un 5%de la población entienden el lenguaje en inglés; esto se vuelve un impedimento al querer introducir estudiantes al área de la tecnología computacional.La barrera del lenguaje no deberia de existir dentro del área,computacional , con este proyecto buscamos cerrar esa brecha educacional. Sin embargo buscamos usar un lenguaje de alto nivel ,Con entrada gráfica , que permita al usuario mediante la lengua española construir codigo fácilmente y comprender los estatutos y sintaxis básicas de múltiples lenguajes de programación.

#### OBJETIVO DEL LENGUAJE CON ÁREA DE APLICACIÓN

Se desea implementar la lengua española a un lenguaje sencillo de programación. Con el propósito de poder facilitar el aprendizaje de la lógica de programación para los estudiantes que no cuenten con el conocimiento del lenguaje inglés. Se busca que el lenguaje sea de alto nivel,con entrada gráfica ,con el fin de facilitar la comprensión de programacion basica para alguien que no tenga experiencia con el área de la programación y busque enseñar o aprender esta habilidad. El lenguaje devolverá una retroalimentación sencilla y fácil de comprender que facilite la codificación del programa. El lenguaje tendrá capacidades de manejo de funciones, manejo de distintos tipos de valores, vectores (arreglo de una dimensión ), matrices (arreglo de dos dimensiones), ciclos, estatutos condicionales y funciones especiales para entrada en tiempo de ejecución y salida de información en tiempo de ejecución.

## 

## 

## a.2) Análisis de Requerimientos y Casos de Uso Generales

#### Requerimientos

* El lenguaje debe estar en español - x
* Ser un lenguaje de texto - x
* Debe correr en terminal - x

#### 

#### Casos de uso:

* Soporte de llamada de funciones.
* Generar código a través de input gráfico. - x
* Declaración de variables globales - x
* Declaración de variables locales - x
* Declaración de vectores - x
* Declaración de matrices - x
* Soporte de operaciones aritméticas básicas - x
* Soporte de ciclos simples y condiciones - x

## a.3) Descripción de los principales Test Cases

#### Test Cases

* Factorial cíclico y recursivo
* Fibonacci cíclico y recursivo
* Find en un vector
* Sort en un vector
* Multiplicación de dos matrices

## a.4) Descripción del proceso

#### Descripción del proceso general seguido para el desarrollo del proyecto

El proyecto fue desarrollado utilizando python como el lenguaje de programación dedicado a implementar la lógica utilizada para desarrollar el programa. Dentro de Python utilizamos PLY que es una implementación de las herramientas para desarrollar compiladores Lex y yacc dentro de nuestro compilador. Y utilizamos Github para tener un control de versiones constante sobre los cambios realizados por cada integrante.

#### Bitácora

Enero/5/2017

Primer lluvia de ideas, preparación de propuesta.

Enero/9/2017

Finalización de la primera propuesta.

Febrero/12/2017

Última revisión de propuesta. Hacia falta correcciones en descripción y gramática.

Febrero/22/2017

Se empezó a implementar el primer avance del proyecto

Marzo/3/2017

Se arreglaron unos errores en la gramática, se arregló en las tablas, se corrigió en el documento y código. Se termino la primer entrega.

Marzo/12/2017

Tabla de simbolos implementada correctamente.

Marzo/19/2017

Cubo Semántico Implementado

Marzo/26/2017

Cuadruplos Aritméticos Completados.

Marzo/29/2017

Implementacion de creación de variables temporales.

Abril/11/2017

Cuádruplos condicionales y de Ciclos agregados

Abril/14/2017

Desarrollo de clase de Procedimientos y manejo de Memoria

Abril/17/2017

Inicio de implementación de blockly para manejo entrada gráfica de compilación.

Abril/20/2017

Implementación de cuadruplos para funciones.

Abril/22/2017

Generación de casos de uso , Corrección de funcionalidad en cuadruplos y corrección en la aplicación de Blockly.

Abril/30/2017

Generacion de vectores

Mayo/1/2017

Generación de Matrices y funcionamiento de procedimiento, documentación

Mayo/2/2017

Ultimos avances.

#### Reflexión Ruben Alejandro Hernandez Gonzalez

El desarrollar un compilador desde cero es una experiencia que todo desarrollador de software debe de haber obtenido de alguna manera u otra. La exige a la pesona alto desempeño en varias áreas de su conocimiento de la carrera. Le da una experiencia de versatilidad a las personas que cumplen con los requisitos necesarios para poder pasar la materia. Este curso me ayudó a practicar más a fondo habilidades necesarias con las que tengo que contar en mi área de trabajo y por eso es recomendable tener experiencia en el área de compiladores. Ayuda mucho a desarrollar el área creativa de un desarrollador de software para resolver problemas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Firma)

#### Reflexión Andrés Marcelo Garza Cantú

El desarrollo de un compilador,permite a cualquiera que pase por este curso tener un mayor conocimiento sobre cómo funcionan los lenguajes actuales. Este conocimiento te permite mejorar tus desarrollo de aplicaciones o código , ya que investigar y aprender de diversos lenguajes será más fácil y la evaluación que hagamos del lenguaje será más precisa. Esta materia es esencial para cualquier ITC y de las más importantes por la versatilidad que uno adquiere y la necesidad de administrar un proyecto en calendario, ayuda al estudiante a organizarse más para poder cumplir con las fechas pedidas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Firma)

b) Descripción del Lenguaje

## b.1) Nombre del Lenguaje

Tuby

## b.2) Principales características del lenguaje

Tuby es un lenguaje orientado a objetos basado en el idioma español con propósito de ser utilizado como lenguaje Orientado a Objetos para poder dar una introducción al área de programación de alto nivel a personas con poco entendimiento del idioma inglés. El idioma deberá poder utilizar ciclos, condiciones, funciones y clases.

## b.3) Posibles errores de compilación/ejecución

#### Errores en compilación

* No se cuenta con Python
* No se cuenta con la herramienta PLY
* No se cuentan archivos existentes con el nombre que se proporcionó.
* Variable previamente declarada.
* Función previamente declarada.
* Llamada de función con parámetros erróneos.
* Variable no declarada.
* Operaciones con tipos de valor incompatibles

#### Errores en ejecución

* Errores de lógica
* valor fuera del rango de direcciones de un arreglo o matriz
* Stack Overflow
* Timeout

c) Descripción del compilador

## c.1) Equipo de computo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto

El compilador se desarrolló en 2 equipos de cómputo Windows 10, con lenguaje Python y utilizando PLY como utilería y corriendo con el IDE Pycharm.

Es necesario Python 3.4.2 para su desempeño óptimo, Se utilizó la librería de Blockly para la generación de código.

## c.2) Descripción de Análisis de Léxico.

#### Tokens asociados al lenguaje de programación

'entero': 'KEYWORD\_TYPE\_ENTERO',

'real': 'KEYWORD\_TYPE\_REAL',

'booleano': 'KEYWORD\_TYPE\_BOOLEANO',

'si': 'KEYWORD\_SI',

'sino': 'KEYWORD\_SINO',

'mientras': 'KEYWORD\_MIENTRAS',

'principal': 'KEYWORD\_PRINCIPAL',

'caracter': 'KEYWORD\_TYPE\_CARACTERES',

'entrada': 'KEYWORD\_ENTRADA',

'salida': 'KEYWORD\_SALIDA',

'funcion': 'KEYWORD\_FUNCION',

'nulo': 'KEYWORD\_NULO',

'retorno': 'KEYWORD\_RETORNO',

'verdadero': 'KEYWORD\_VERDADERO',

'falso': 'KEYWORD\_FALSO'

t\_SEMICOLON = r'\;'

t\_PUNTO = r'[\.]'

t\_COMMA = r'[\,]'

t\_COLON = r'\:'

t\_BRACKET\_IZQ = r'\{'

t\_BRACKET\_DER = r'\}'

t\_PARENTESIS\_IZQ = r'\('

t\_PARENTESIS\_DER = r'\)'

t\_CORCHETE\_IZQ = r'\['

t\_CORCHETE\_DER = r'\]'

t\_INTER\_IZQ = r'\¿'

t\_INTER\_DER = r'\?'

t\_OPERADOR\_IGUAL = r'\='

t\_OPERADOR\_COMPARATIVO = r'[>]|[<]'

t\_OPERADOR\_AND\_OR = r'&&|\|\|'

t\_EXP\_OPERADOR = r'\+|\-'

t\_TERM\_OPERADOR = r'\\*|\/'

t\_ignore = ' \t\n\r'

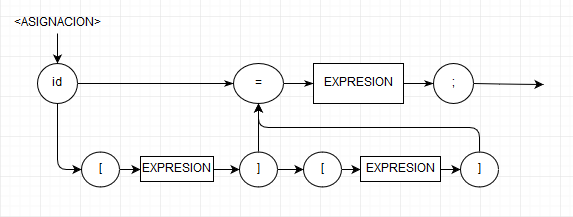
## c.3) Descripción de Análisis de Sintaxis

La memoria la manejamos utilizando un monolito que guarda el valor de memoria en un hash que

La clase de memoria real asigna un valor corre cuádruplo por cuádruplo donde se detectan los

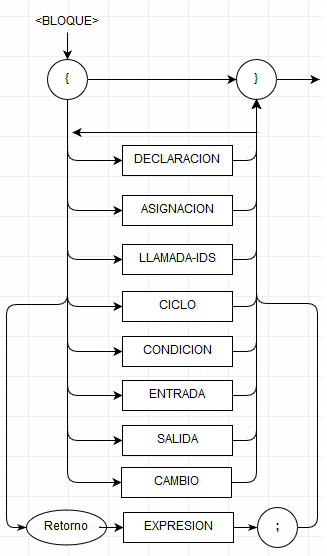
## c.4) Descripción de Generación de Código Intermedio

#### Asignación



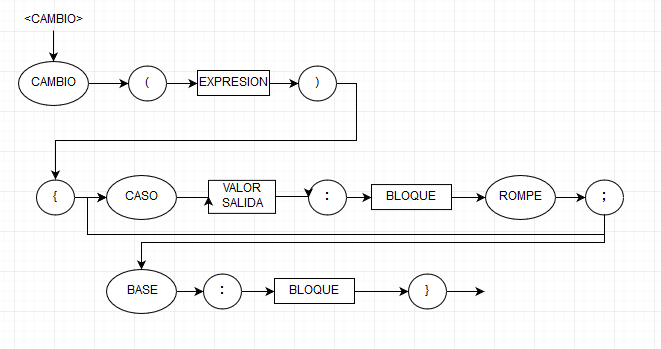
Nuestro diagrama de sintaxis de Asignación está desarrollado con la opción de proveer múltiples ids para poder hacer referencia a las funciones dentro de clases dentro del lenguaje. También cuenta con la opción de introducir el valor de matrices o arreglos.

#### Bloque



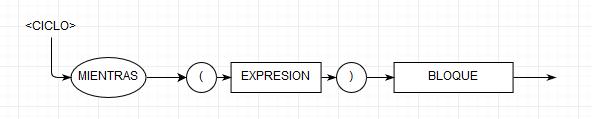
Bloque nos permite introducir información a correr dentro de cada función nueva que se encuentra encapsulado entre corchetes.

#### Cambio



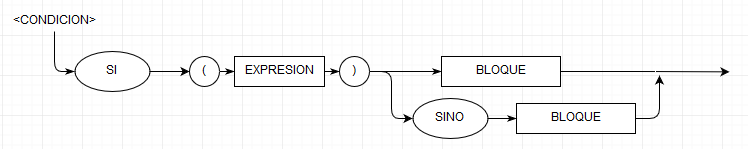
Esta función funcionara igual que una de switch en algunos otros lenguajes. Se verifica una condición en la que se correran diferentes bloques que cumplan diferentes casos de los resultados de la expresión.

#### Ciclo



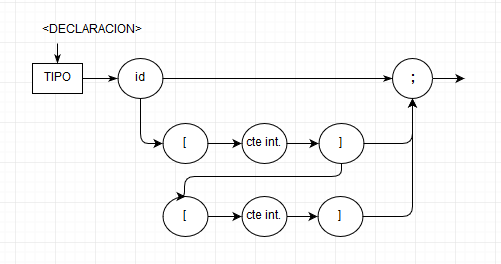
Ciclo es nuestra función while, se le va a proveer una expresión dentro de paréntesis para poder decidir cuando se sale del ciclo.

#### Condición



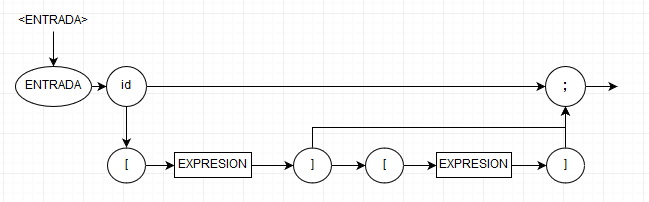
Condición es nuestra sintaxis de if y else de nuestro lenguaje. Si se cumple con la condición de la primer expresión, se corre el primer bloque, si no, corre el bloque de sino o simplemente se salta ambos bloques si es que no se encuentra una condición Sino.

#### Declaración



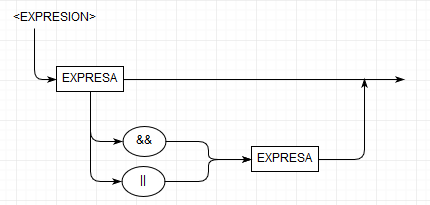
Declaración se utiliza para definir una nueva variable, si es arreglo o matriz se introducen la cantidad de brackets necesarios.

#### Entrada



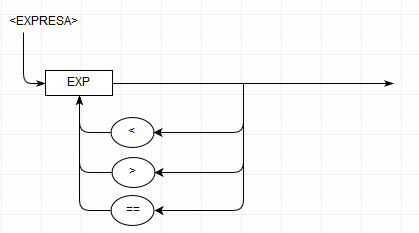
Valor de entrada fuera del programa. Se le pide al usuario teclear lo que se desea introducir dentro del programa.

#### Expresión



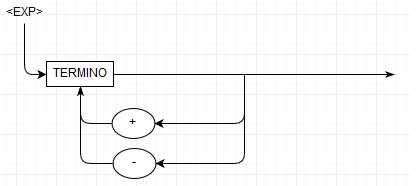
Expresión se dedica proveer valores numéricos o booleanos a través de un orden sintáctico de las operaciones a ejecutar. Empieza con los operadores de AND y OR.

#### Expra



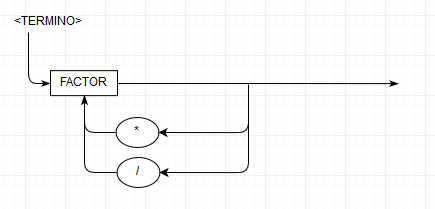
Parte del orden de sintaxis de Expresión encargada de proveer los operadores comparativos de menor que y mayor que.

#### Exp



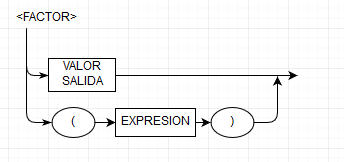
Parte de del orden de Expresión en el que se ingresan los operadores de suma o resta.

#### Término



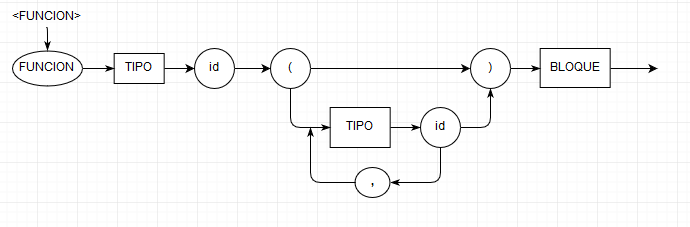
Término es el otra parte de Expresión en la que se registran los operadores de multiplicación y división en el orden de cuádruplos.

#### Factor



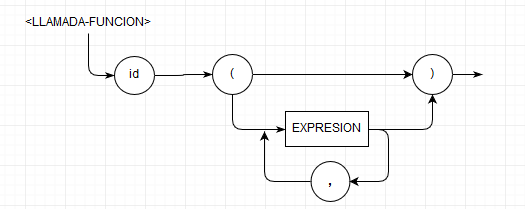
Factor es la última parte de Expresión, en el cual se proveen los valores que serán distribuidos a la expresión por realizar o introduce jerarquía de operaciones dentro de la expresión.

#### Funcion



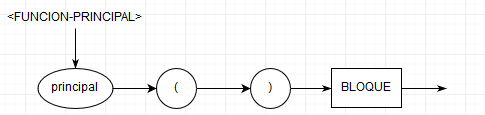
Nuestro manejo de sintaxis de Funciones, se pueden proveer múltiples parámetros, y al terminar de proveerlos se inicia el bloque en el que se realizarán las acciones.

#### Llamada-Función



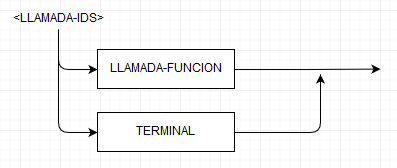
Llamada-Función es lo que utilizamos para hacer una llamada a los datos que una función nos va a proporcionar dentro de nuestro programa.

#### Llamada-Principal



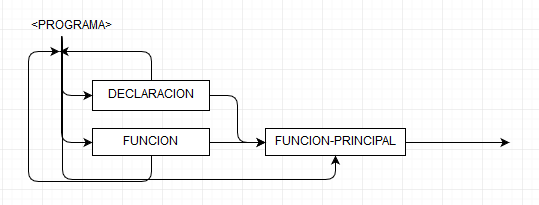
Función que utilizamos como main para nuestro programa, es donde se inicia la ejecución de los cuádruplos y de allí se empiezan a llamar los otros elementos dentro del programa.

#### Llamada-Ids



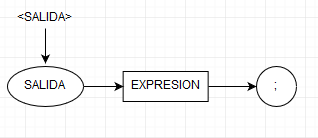
Esta tabla nos ayuda a ordenar la llamada entre Llamada-Función y valores Terminales que podríamos llamar desde Valor Salida

#### Programa



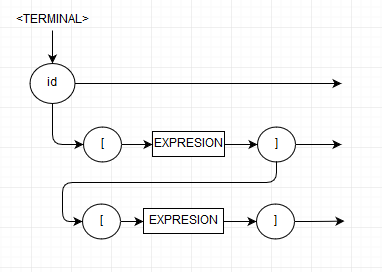
Es la llamada a nuestro programa a correr en nuestro compilador. Dentro de el se pueden realizar declaraciones globales, clases o funciones antes de iniciar la función principal.

#### Salida



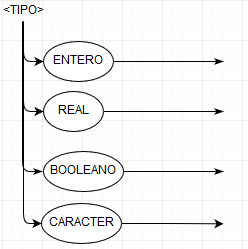
Es el valor a desplegar en pantalla para que el usuario pueda ver resultados de su programa.

#### Terminal



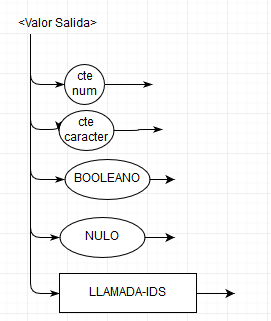
Valores terminales del programa, es aquí donde se hace referencia a matrices o llamadas a función de clases como valores de salida.

#### Tipo



Definición del tipo de nuestras variables, se le regresa el valor cada vez que se hace referencia dentro del programa.

#### Valor Salida



Valores de salida que se introducirán dentro de las expresiones del programa.

## c.5) Descripción del proceso de Administración de memoria

Para el manejo de memoria se generaron las dos clases siguientes: Monolito y MemoriaReal. La clase memoria real funciona de la siguiente manera: Cuando se declara una variable, constante o temporal , dependiendo de su tipo genera una dirección de memoria, del 0 al 2500 son variables booleanas, del 2501 al 5000 son variables enteras, del 5001 al 7500 son reales y del 7501 al 10,000 son de tipo carácter, una vez definido su dirección de memoria, el se almacenan en el monolito como un diccionario ({memID:valor}), donde el valor inicial dependiendo del tipo es definido y almacenado en el monolito.

Valores default de almacenamiento:

* “falso” para booleanos
* 0 para enteros
* 0.0 para reales
* “n/a” para caracteres

de excederse de las dimensiones posibles a declarar el programa avisará con anterioridad y dejara de correr de exceder las casillas posibles.

d) Descripción de la Máquina Virtual

## d.1) lenguaje y utilerías especiales utilizadas.

Generador de código Tuby (Desarrollado en Blockly)

LA funcion de Ejecutar se encarga de leer cada operacion del cuadruplo y ejecutarlas debidamente. La clase se anexa a continuacion

**class** MaquinaVirtual:

**def** \_\_init\_\_(self):

self.stack = list()

**def** Ejecutar(self,cuadruplo,mon, proc):

indiceActual = 0

stackIndices = []

stackIndicesFinales = []

stackERA = []

stackParam = 0

print(cuadruplo.cuadruplos[0][0])

**while**(**True**):

indiceTemporal = 0

operador = cuadruplo.cuadruplos[indiceActual][0]

cuad = cuadruplo.cuadruplos[indiceActual]

**if**(operador == **"+"**):

Suma(cuad,mon)

**elif**(operador == **"-"**):

Resta(cuad,mon)

**elif**(operador == **"\*"**):

Multiplicacion(cuad,mon)

**elif**(operador == **"/"**):

Division(cuad,mon)

**elif**(operador == **"="**):

Asignar(cuad,mon)

**elif**(operador == **"output"**):

Salida(cuad,mon)

**elif**(operador == **"input"**):

Entrada(cuad,mon)

**elif**(operador == **"GOTOF"**):

indiceTemporal = GotoF(cuad,mon,indiceActual)

**elif**(operador == **"GOTOT"**):

indiceTemporal = GotoT(cuad,mon,indiceActual)

**elif**(operador == **"GOTO"**):

indiceTemporal = Goto(cuad,mon)

**elif**(operador == **"gosub"**):

print(**"run Gosub"**)

stackIndices.append(indiceActual+1)

print (stackIndices)

indiceTemporal = Gosub(cuad,proc)

**elif**(operador == **"RETU"**):

print(**"run RETU"**)

nuevoIndice = stackIndices.pop()

indiceTemporal = nuevoIndice

**elif**(operador == **"ERA"**):

print(**"run ERA"**)

final = ERA(cuad,proc)

print (final)

stackIndicesFinales.append(final)

print (stackIndicesFinales)

stackERA.append(cuad[1])

print (stackERA)

**elif**(operador == **"param"**):

stackParam = stackParam + 1

func = stackERA[-1]

Param(cuad, mon, proc, func, stackParam)

print (func)

**elif**(operador == **"&&"**):

AndOp(cuad,mon)

**elif**(operador == **"||"**):

OrOp(cuad,mon)

**elif**(operador == **">"**):

MayorQue(cuad,mon)

**elif**(operador == **"<"**):

MenorQue(cuad,mon)

*##Funciones de arreglos*

**elif**(operador == **"ver"**):

Ver(cuad,mon)

**elif**(operador == **"+v"**):

DirArreglo(cuad,mon)

*##FIN DE FUNCIONES DE ARREGLO*

**elif**(operador == **"FIN"**):

**break**;

**if**(indiceTemporal == 0):

indiceActual = indiceActual + 1

**else**:

indiceActual = indiceTemporal + 1

## d.2) Descripción detallada del proceso de Administración de Memoria en ejecución.

La memoria se crea de la siguiente manera se ingresa una variable de tipo entero denominada rango. En Rango se almacena el rango de valores de memoria de inicio (sease 0, 100000,250000,50000, etc.) de esa direccion +2500 son las direcciones booleanos,de 2501 a 5000 son de tipo entero , de 5001 a 7500 son de tipo real y de 7501 hasta 7500 + 2500 son caracteres.

Al momento de declarar una variable, definir una constante o del cuádruplo requerir de una variable temporal, se harán los siguientes 3 pasos:

primero definir la dirección de memoria de la variable, temporal o constante

después darle un valor Default

y finalmente lo guardará en forma de diccionario dentro del monolito de la siguiente manera

**{‘memoryID’:valor}**

El cuádruplo trabaja directamente con el monolito para accesar los valores de memoria y darles sus respectivos valores.

class MemoriaReal:

def \_\_init\_\_(self, rango = 0):

self.booleanos = rango

self.enteros = rango + 2500

self.reales = rango + 5000

self.caracteres = rango + 7500

self.cont\_bool = self.booleanos

self.cont\_ent = self.enteros

self.cont\_real = self.reales

self.cont\_car = self.caracteres

def insertaBooleano(self):

if(self.cont\_bool < self.enteros):

self.cont\_bool = self.cont\_bool + 1

return self.cont\_bool

else:

print("memoria fuera de limites")

raise SystemExit

def insertaEntero(self):

if(self.cont\_ent < self.reales):

self.cont\_ent = self.cont\_ent + 1

return self.cont\_ent

else:

print("memoria fuera de limites")

raise SystemExit

def insertaReales(self):

if(self.cont\_real < self.caracteres):

self.cont\_real = self.cont\_real + 1

return self.cont\_real

else:

print("memoria fuera de limites")

raise SystemExit

def insertaCaracteres(self):

if(self.cont\_car < self.caracteres + 2500):

self.cont\_car = self.cont\_car + 1

return self.cont\_car

else:

print("memoria fuera de limites")

raise SystemExit

def insertaDimBooleano(self,total):

if(self.cont\_bool + total < self.enteros):

inicio = self.cont\_bool + 1

self.cont\_bool = self.cont\_bool + total

fin = self.cont\_bool

retorno = {'inicio': inicio, 'fin':fin}

return retorno

else:

print("memoria fuera de limites")

raise SystemExit

def insertaDimEntero(self,total):

if(self.cont\_ent + total < self.reales):

inicio = self.cont\_ent + 1

self.cont\_ent = self.cont\_ent + total

fin = self.cont\_ent

retorno = {'inicio': inicio, 'fin':fin}

return retorno

else:

print("memoria fuera de limites")

raise SystemExit

def insertaDimReales(self,total):

if(self.cont\_real + total < self.caracteres):

inicio = self.cont\_real + 1

self.cont\_real = self.cont\_real + total

fin = self.cont\_real

retorno = {'inicio': inicio, 'fin':fin}

return retorno

else:

print("memoria fuera de limites")

def insertaDimCaracteres(self,total):

if(self.cont\_car + total < self.caracteres + 2500):

inicio = self.cont\_car + 1

self.cont\_car = self.cont\_car + total

fin = self.cont\_car

retorno = {'inicio': inicio, 'fin':fin}

return retorno

else:

print("memoria fuera de limites")

def eliminaTemporales(self,topeBool, topeInt,topeReal, topeCar):

print("funcion de borrado")

class Monolito:

def \_\_init\_\_(self):

self.Monolito = dict()

self.Constantes = dict()

self.stack = list()

def insertaActual(self, memID, value):

self.Monolito[memID] = value

def insertaConstante(self,memID,value):

self.Constantes[memID] = value

def buscar(self, id):

if(self.Monolito.get(id) is None):

return self.Constantes.get(id)

else:

return self.Monolito.get(id)

def insertaStack(self):

toStack = dict(self.Monolito)

self.stack.append(toStack)

def popStack(self):

return self.stack.pop()

def imprimir(self):

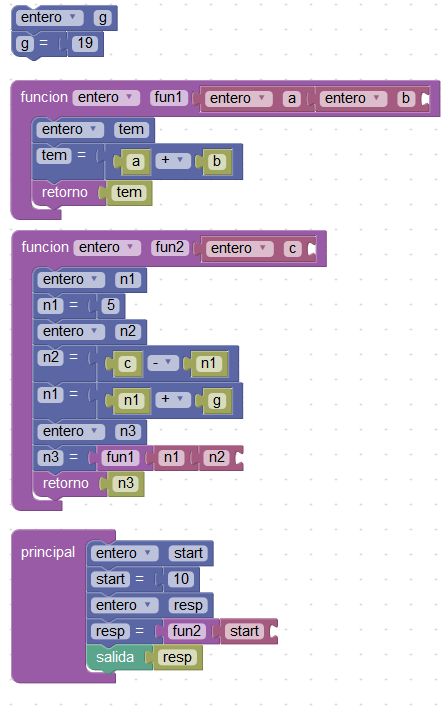
print("Monolito: ",self.Monolito)

print("Constantes: ",self.Constantes)

e) Pruebas del funcionamiento del lenguaje

## e.1) Pruebas que comprueben el funcionamiento del lenguaje

### Prueba de manejo de funciones



entero g;

g = 19;

funcion entero fun1 (entero a, entero b){

entero tem;

tem = a + b;

retorno tem;

}

funcion entero fun2 (entero c){

entero n1;

n1 = 5;

entero n2;

n2 = c - n1;

n1 = n1 + g;

entero n3;

n3 = fun1( n1, n2);

retorno n3;

}

principal(){

entero start;

start = 10;

entero resp;

resp = fun2( start);

salida resp;

}

DEBE devolver 29

### Prueba de matrices y ciclos

principal(){

entero mat1 [2][2];

entero mat2 [2][2];

entero mat3 [2][2];

entero a;

entero b;

entero x;

entero y;

a = 0;

b = 0;

x = 1;

y = 6;

mientras (a < 2){

mientras (b < 2){

mat1 [a][b] = x + (a + b);

mat2 [a][b] = y - (a + b);

b = b + 1;

}

a = a + 1;

}

a = 0;

b = 0;

mientras (a < 2){

mientras (b < 2){

salida mat1 [a][b];

b = b + 1;

}

b = 0;

a = a + 1;

}

a = 0;

b = 0;

mientras (a < 2){

mientras (b < 2){

salida mat2 [a][b];

b = b + 1;

}

a = a + 1;

}

a = 0;

b = 0;

mientras (a < 2){

mientras (b < 2){

mat3 [a][b] = mat1 [a][b] \* mat2 [a][b];

salida mat3 [a][b];

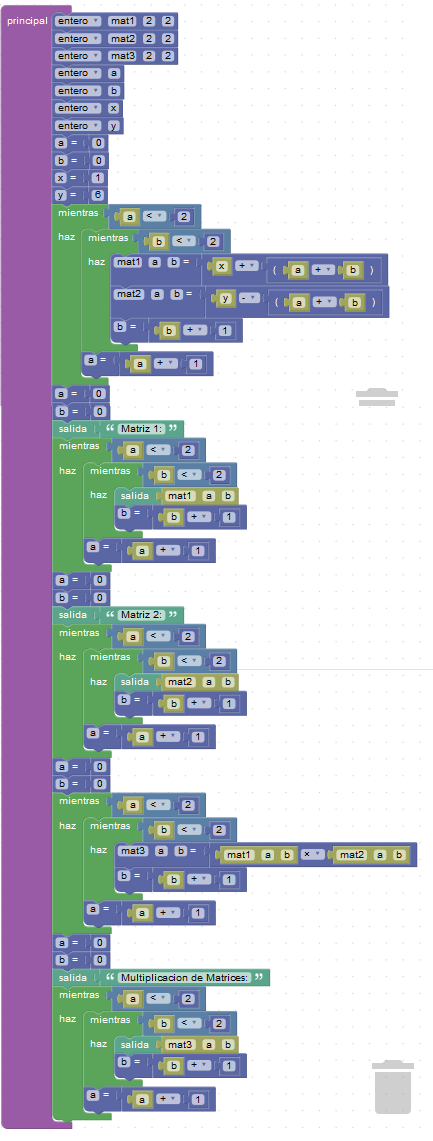
b = b + 1;

}

a = a + 1;

}

}



### Prueba de Condicion

principal(){

entero mayor;

entero menor;

mayor = 1;

menor = 0;

si (mayor < menor) {

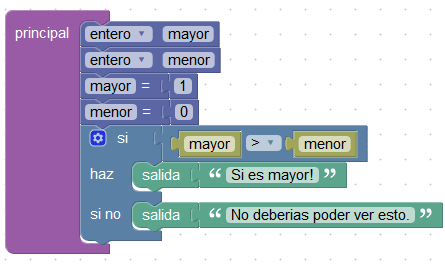
salida "Si es mayor!";

} sino {

salida "No deberias poder ver esto";

}

}



### Prueba de Ciclo Mientras

principal(){

entero a;

entero b[10];

entero c[2][5];

a = 10;

mientras (a > 0){

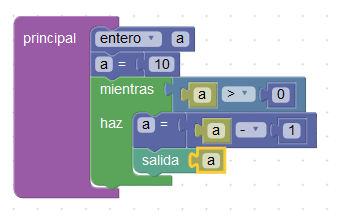
b[2] = a - 1;

a = a -1;

salida a;

}

}



### Prueba Find con valor de Entrada

principal(){

entero lista [5];

lista [0] = 4;

lista [1] = 2;

lista [2] = 9;

lista [3] = 7;

lista [4] = 3;

entero num;

booleano busca;

busca = falso;

entero i;

i = 0;

salida "Teclea un numero del 1 al 10 que deseas encontrar.";

entrada num;

mientras (i < 5){

si (lista [i] ~ num) {

busca = verdadero;

}

i = i + 1;

}

si (busca ~ verdadero) {

salida "Si se encontro el numero!";

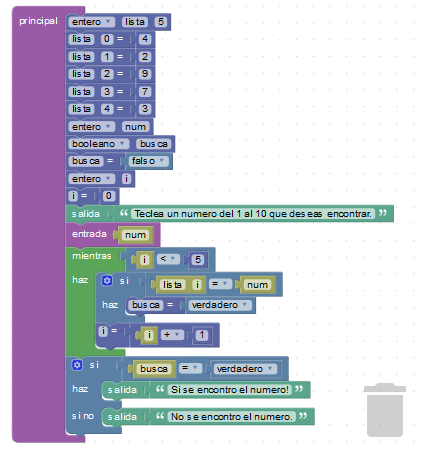
}

sino {

salida "No se encontro el numero.";

}

}



### Prueba Find con valores Estáticos

principal(){

entero lista1 [3];

lista1 [0] = 4;

lista1 [1] = 2;

lista1 [2] = 9;

entero lista2 [3];

lista2 [0] = 6;

lista2 [1] = 2;

lista2 [2] = 4;

booleano bools [3];

bools [0] = falso;

bools [1] = falso;

bools [2] = falso;

entero i;

entero j;

i = 0;

j = 0;

mientras (i < 3){

mientras (j < 3){

si (lista1 [j] ~ lista2 [i]) {

bools [i] = verdadero;

}

j = j + 1;

}

i = i + 1;

}

i = 0;

salida "Los siguientes numeros:";

mientras (i < 3){

salida lista2 [i];

si (bools [i] ~ verdadero) {

salida "Si se encontro!";

}

sino {

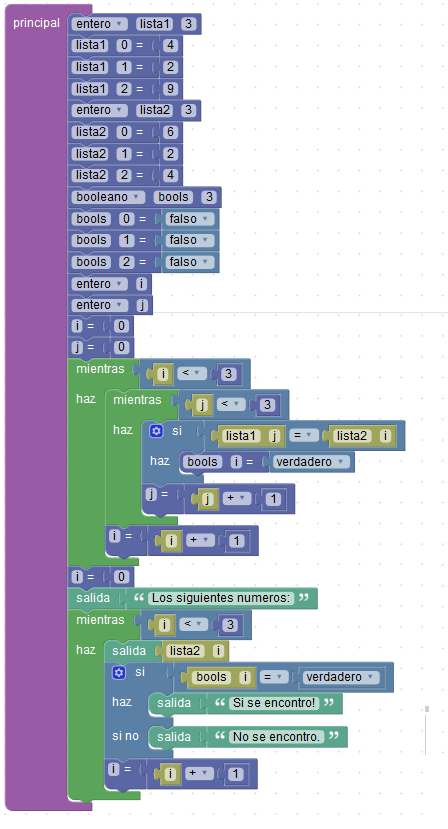
salida "No se encontro.";

}

i = i + 1;

}

}



### Prueba de Sort de Arreglos

principal(){

entero lista [5];

lista [0] = 3;

lista [1] = 1;

lista [2] = 2;

lista [3] = 5;

lista [4] = 4;

entero aux;

entero i;

i = 0;

mientras (i < 5){

salida lista [i];

i = i + 1;

}

entero j;

entero k;

j = 0;

k = 0;

mientras (j < 4){

mientras (k < (4 - j)){

si (lista [k] > lista [k+1]) {

aux = lista [k];

lista [k] = lista [k+1];

lista [k+1] = aux;

}

k = k + 1;

}

j = j + 1;

}

salida "Lista ordenada";

entero a;

a = 0;

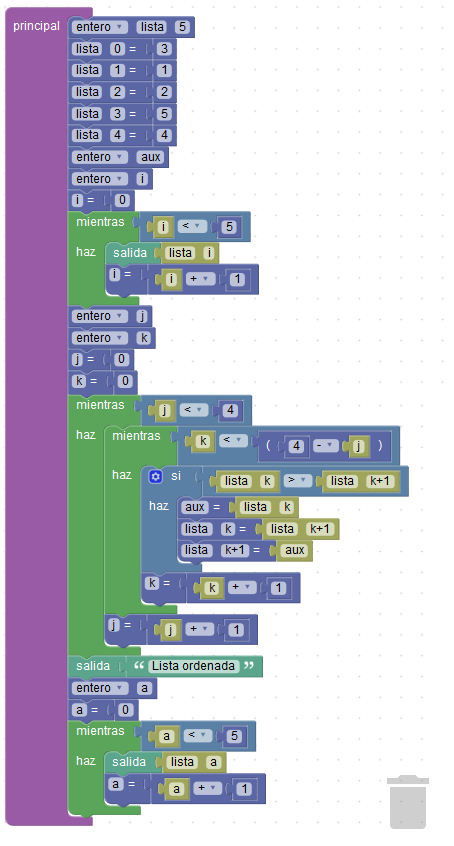
mientras (a < 5){

salida lista [a];

a = a + 1;

}

}



## Pruebas de error

### Error de funciones con el mismo nombre.

### 

funcion real gemini (){

}

funcion entero gemini (){

}

principal(){

salida "Falla.";

}

### Error de Doble parametro.

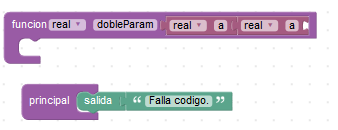
funcion real dobleParam (real a, real a){

}

principal(){

salida "Falla codigo.";

}



### Error de falla doble.

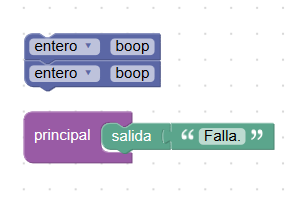
entero boop;

entero boop;

principal(){

salida "Falla.";

}



### Codigo de interés:

Manual de Usuario

Instalación de las herramientas requeridas

Tuby utiliza principalmente 3 herramientas para poder correr de manera adecuada. Las 3 son Python, PLY, Blockly y Pycharm.

#### Python

Tuby está basado en el lenguaje de programación Python. Para poder instalar el lenguaje, puede utilizar el siguiente link en el que se encuentra la version mas nueva 3.5.2, la cual es en la que se desarrolló nuestro lenguaje.<https://www.python.org/downloads/>

#### PLY

PLY es una herramienta de python que utilizamos para poder desarrollar Tuby. Pueden utilizar el siguiente link de descarga para descargar la versión más reciente.<http://www.dabeaz.com/ply/>

Para hacerlo funcionar con el lenguaje, coloque la carpeta ply que extrajo en el directorio que se encuentran los archivo del lenguaje.

#### PyCharm

PyCharm es el IDE que utilizamos para correr y desarrollar el lenguaje. Por medio de Pycharm pudimos verificar que el lenguaje está corriendo adecuadamente. Utilize el siguiente link de descarga para poder descargar, utilizar PyCharm y poder utilizar Tuby ahora<https://www.jetbrains.com/pycharm/>.

#### 

#### Blockly

El compilador es un compilador cuenta con una interfaz de input gráfico para poder hacer que el usuario pueda realizar el código de su programa de una manera más amigable y sencilla. La interfaz corre en Blockly con bloques personalizados para que interprete código utilizando el léxico, sintaxis y semántica de Tuby. La carpeta de Blockly ya viene incluida en la carpeta del proyecto. Desde la carpeta del proyecto, se va a Blockly-Tuby/blockly y se corre el HTML Blockly Tuby.html.

Compilación

Para poder compilar su programa, abra scanner.py en PyCharm. De click en la barra de Menú llamada “Run”. Despues Run > scanner.py para que se compilen las instrucciones del compilador. Le pedirá el nombre del archivo de texto que desea compilar, teclee su nombre para poder compilar el programa.

Convenciones de Léxico

## Identificadores

Existen dos tipos de identificadores, identificadores de variable e identificadores de función

para declarar un identificador de variable deber hacerlo de la siguiente manera:

Tipo Identificador, primero se declara el tipo de valor a almacenar (booleano,entero,real,caracter) después el identificador.

Para declarar un identificador de función se declara de la siguiente manera:

Función Tipo Identificador, primero debes de poner la palabra reservada función , después el tipo de función que será y al final el nombre de la función.

#### Variables numéricas

Se cuentan con 2 opciones de variable numéricas, las cuales son enteros o reales. Los enteros son números que no cuentan con punto decimal y los reales son números que pueden ser utilizados para operaciones decimales. Para definirlos hágalo de la siguiente manera:

**entero numero;**

**real numerodecimal;**

Para poder asignarles un valor, teclee el nombre que le dio, seguido de = y un valor numérico que sea aceptable en sus valores.

**numero = 2;**

**numerodecimal = 1.1;**

#### Variables de texto

Tuby cuenta con variables de texto llamadas caracteres, se definen igual que las numericas, pero utilizan la palabra clave caracter al inicio.

**caracter nombre;**

Para poder asignarles un valor se hace de la siguiente manera.

**nombre = “Ruben”;**

#### Variables Booleanas

Las variables Booleanas pueden ser utilizadas para proporcionar valores de verdadero y falso a condiciones que queramos manejar. Se pueden utilizar para entrar o salir de condiciones o ciclos.

**booleano compruebaFalso;**

**booleano compruebaVerdadero;**

**booleano compruebaFalso = falso;**

**booleano compruebaVerdadero = verdadero;**

#### 

## Palabras reservadas

entero, real, booleano, si, sino, mientras, principal, caracter, entrada, salida, funcion, nulo, retorno, verdadero, falso

## Diseño de programas

Los programas están divididos en una sección para declarar tus clases, funciones o variables globales que quieran utilizar en el programa y la función principal que hará referencia a todas las funciones y variables que se le pide que referencia en sus instrucciones.

## Expresiones

Al realizar una expresión se debe tener compatibilidad entre las variables o constantes introducidas en la expresión.

**expresion1 = 2+3; O**

**expresion2 = expresion1 / 2; O**

**expresion3 = (4 > 5); O**

**expresion4 = “baile” + falso; X**

**expresion5 = verdadero - falso; X**

## Condiciones

Las condiciones nos ayudan a verificar cuando va a seguir o terminar un ciclo, dar pase para realizar ciertas acciones. Dentro de los paréntesis de las condiciones se tiene que introducir una expresión que decía las acciones a tomar dentro de ellas.

**booleano no = falso;**

**si (booleano){**

**contador = contador + 2;**

**}sino {**

**contador = 2;**

**}**

## Ciclos

Los ciclos que utilizamos tiene que ser iniciados con la palabra mientras, seguida de una expresión dentro de un paréntesis. Al momento de que la condición no se cumpla, la ejecucion del lenguaje saldrá del bloque de mientras.

**mientras (hola < adios && adios > buenas noches){**

**baila[hola] = adios;**

**hola = hola + 1;**

**adios = adios + 1;**

**}**

## Funciones

Las funciones de declaran utilizando la palabra clave función, seguía por el tipo de valor que regresaran y después de su nombre, después tendrán que proveer los parámetros que maneja junto con sus tipos dentro de paréntesis () y separados por comas y el bloque con las acciones a realizar.

**funcion entero perro (entero rojo){**

**entero azul;**

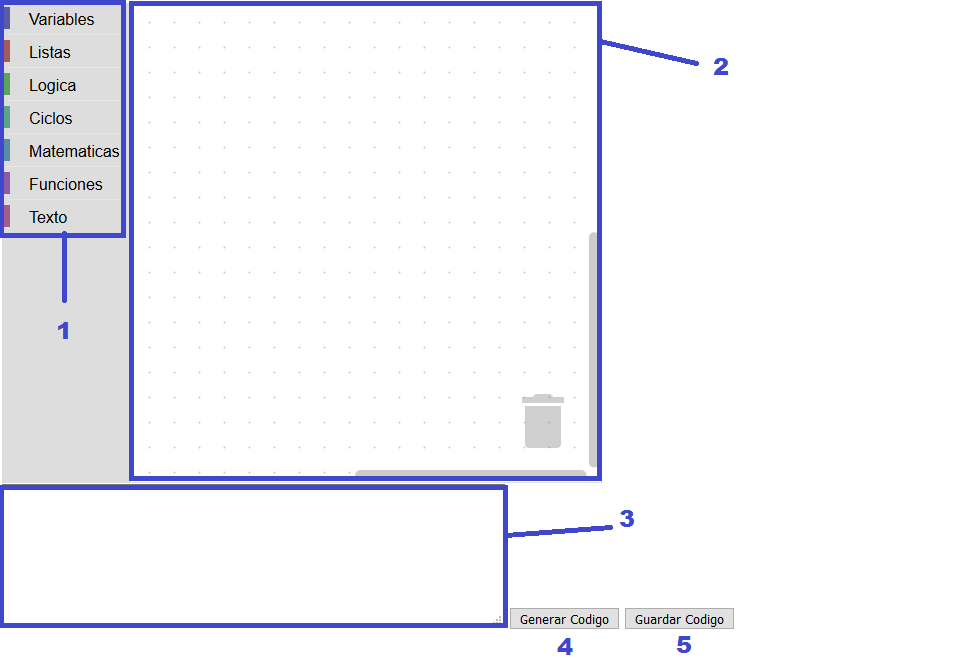
**azul = rojo;**

**retorno azul + 4;**

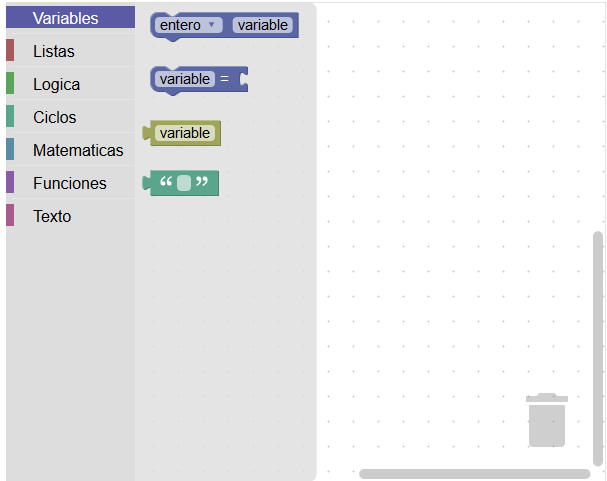
**}**

Manual de usuario - Blockly

Al iniciar Blockly, se presentará con una vista en la que se podrán seleccionar seleccionar diferentes tipos de bloques de código (1), un área para poder colocar los bloques con los que se piensa realizar el programa (2), un área de texto en la que se traducen los bloques colocados a codigo Tuby (3), un botón para generar el código que se genera a partir de los bloques y sus posiciones (4) y un botón para poder guardar el código que se generó dentro del área de texto (5).

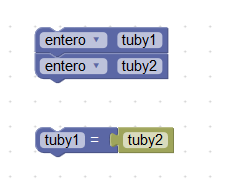


Para poder empezar a programar en Blockly, se pueden seleccionar el tipo de bloque que se quiere colocar y arrastrarlo en el área de bloques.

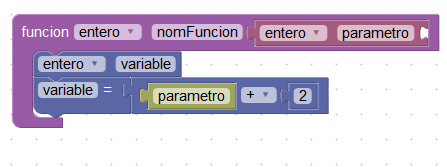
 

Se cuentan con diferentes tipos de bloques, con diferente tipo de conexiones.

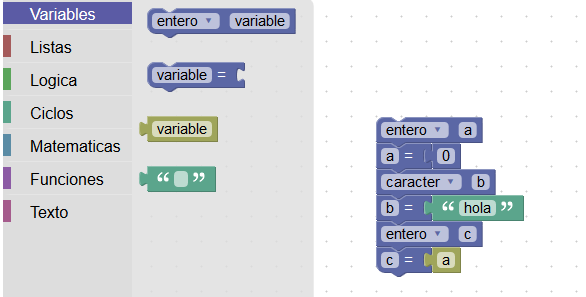
Los bloques pueden conectarse de manera horizontal y vertical.



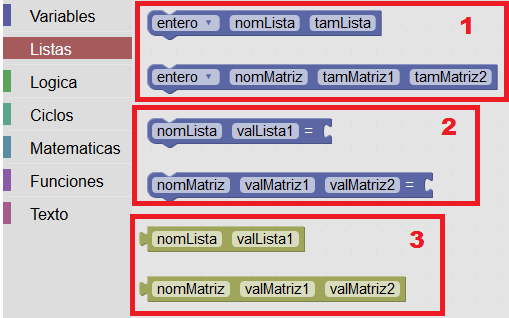
Las conexiones también pueden realizarse en las secciones de parámetros de una funcion y dentro de expresiones. Las conexiones también se pueden realizar dentro de bloques de ciclos, ifs, funciones y main.



En la sección de variables se pueden llamar los tipos de bloques necesarios para poder definir variables y asignarles valor. En caso de querer dar un valor entre comillas, se puede usar el bloque con comillas para asignar ese valor.



En la seccion de Listas se podran seleccionar los bloques para poder definir una lista o matriz (1), bloques para asignar un valor a un espacio de lista o matriz (2) o para asignarle el valor de un espacio de lista o matriz a una variable (3).

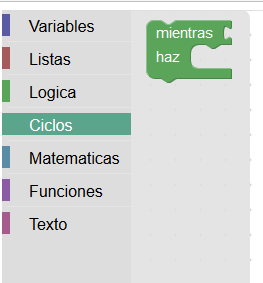
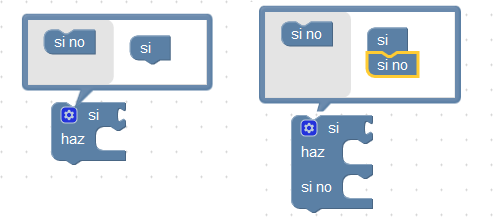


En la seccion de logic es donde se encontraran los bloques para poder hacer expresiones “==”, < y > (1), expresiones de AND y OR (2), asignar valores booleanos (3) o utilizar el metodo if o if -> else (4).

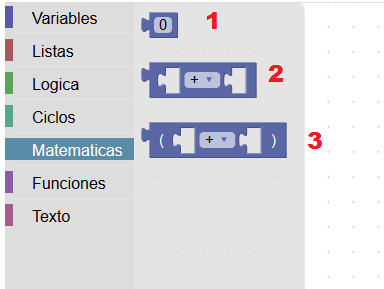


En el caso de querer utilizar el bloque de if (si) como bloque de if -> else (si -> si no). Se puede dar click en el botón de engranaje y arrastrar el sub-bloque si no abajo del sub-bloque si.

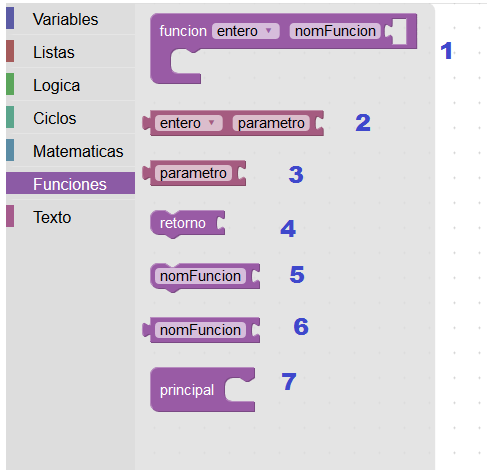
En la sección de ciclos se encuentra nuestro ciclo while (mientras) en el que se ejecuta un bloque en lo que se cumpla una expresión.



En la sección de matemáticas se podrán utilizar bloques para asignar valores numéricos a las variables o expresiones (1), y bloques para poder introducir expresiones matemáticas dentro o sin parentesis (2 y 3).



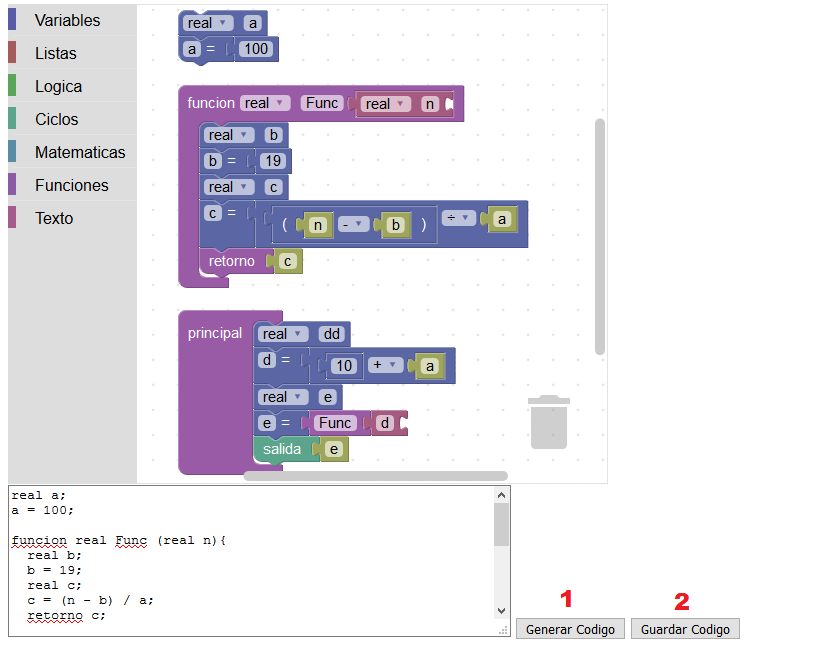
En la sección de Funciones se contará con bloques de declaración de funciones (1), para declarar los parámetros que utiliza cada función (2), para poder introducir multiples parametros (3) en bloques de llamada a función (5 y 6), bloques de retorno para regresar el valor final de la función (4) y el bloque de nuestra función principal (7).



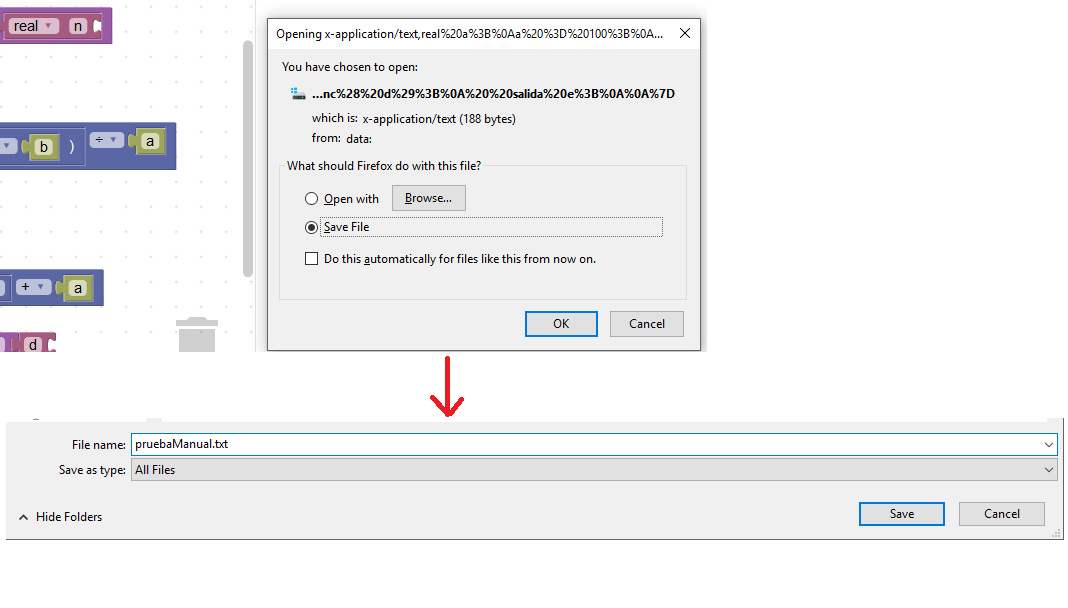
En la sección de Texto es donde se encuentran nuestros bloques para poder usar valores de entrada (3) y salida (2) al igual que asignar valores string a variables (1). NOTA: No guardar valores de caracteres que utilicen colon.



Para poder generar código dependiendo de los bloques que se utilizaron, se le presionara al botón de “Generar Código” (1) para poder generar el código en lenguaje Tuby y así poderlo revisar. Para poder guardar el código que se genero, se presiona el botón de “Guardar Codigo” (2). NOTA: Es importante haber generado el código en la sección de texto para que este pueda ser guardado.



Ya al tener la opción de guardar el archivo se podrá guardar como un archivo de texto. Es necesario guardarlo dentro de la carpeta del proyecto para que este archivo pueda correr en el compilador.



1. "Lenguas de México - Wikipedia, la enciclopedia libre." 2011. 23 Sep. 2016 <<https://es.wikipedia.org/wiki/Lenguas_de_M%C3%A9xico>> [↑](#footnote-ref-0)