



Tecnológico  
de Monterrey



**zencode**  
a peaceful introduction to code

**David Macías**  
01283419

5 de junio, 2023

# CONTENIDO

<b>DESCRIPCIÓN DEL PROYECTO .....</b>	<b>2</b>
PROPÓSITO DEL PROYECTO .....	2
ALCANCE DEL PROYECTO .....	3
ANÁLISIS DE REQUERIMIENTOS .....	4
DESCRIPCIÓN GENERAL DEL PROYECTO .....	5
BITÁCORA.....	6
<b>DESCRIPCIÓN DEL LENGUAJE .....</b>	<b>9</b>
.....	9
ERRORES .....	10
<b>DESCRIPCIÓN DEL COMPILADOR .....</b>	<b>12</b>
ESPECIFICACIONES .....	12
LÉXICO .....	12
SINTAXIS.....	14
DIAGRAMAS DE SINTAXIS .....	19
CÓDIGOS DE OPERACIÓN Y SUS ATRIBUTOS .....	29
TABLA DE CONSIDERACIONES SEMÁNTICAS .....	31
<b>ESTRUCTURAS DE DATOS .....</b>	<b>32</b>
DIRECTORIO DE FUNCIONES.....	32
TABLA DE VARIABLES .....	32
CUÁDRUPLOS.....	32
LISTA DE CONSTANTES .....	32
DIRECTORIO DE TABLAS .....	32
PILAS.....	33
<b>ADMINISTRACIÓN DE MEMORIA.....</b>	<b>34</b>
<b>PRUEBAS .....</b>	<b>36</b>
<b>APÉNDICE.....</b>	<b>38</b>

# DESCRIPCIÓN DEL PROYECTO

## PROPÓSITO DEL PROYECTO

El objetivo de este proyecto es ofrecer una introducción accesible a la programación y, al mismo tiempo, tender un puente entre las personas con escasos conocimientos de programación y el mundo de la programación. El proyecto pretende crear un compilador para este nuevo lenguaje, **ZenCode**, diseñado para facilitar el proceso de aprendizaje y permitir a los usuarios comprender los conceptos fundamentales de la programación previo a la inmersión en los conceptos y costumbres.

El lenguaje implementado por este compilador incorpora los componentes básicos que se encuentran en los lenguajes de programación funcionales, incluidas operaciones aritméticas y lógicas, variables, sentencias condicionales (condiciones `if`), sentencias iterativas (ciclos `while/do-while`) y estructuras de control iterativas (ciclos `for`, denominados "loop"). Al utilizar conceptos menos técnicos, como *decimal* (`dec`) para valores numéricos de punto flotante, `text` para datos de caracteres y bucle para procesos iterativos, el lenguaje pretende mejorar la comprensión y el apego del usuario al ambiente programación.

Además, el proyecto sirve como una introducción superficial al análisis de datos, ofreciendo a los usuarios una pequeña introducción a las consultas de datos en un formato que no requiera indexación completa, sino que pueda ser referenciado con un identificador. Para facilitar la manipulación y el análisis de datos, el lenguaje introduce un nuevo tipo llamado *datatable* (`data`). Este tipo permite nombrar columnas y ofrece funcionalidades básicas para el análisis estadístico basado en columnas, permitiendo a los usuarios realizar operaciones comunes de análisis de datos dentro del propio lenguaje.

Combinando estos aspectos, el proyecto pretende crear un entorno en el que los usuarios puedan pasar gradualmente de un estado de ausencia total de conocimientos de programación a desarrollar una base transitiva en conceptos de programación, y el análisis de datos. Nuestro objetivo es capacitar a las personas para explorar el mundo de la programación, fomentando sus habilidades y allanando el camino para un mayor aprendizaje y crecimiento en estos dominios que cada vez componen una parte más considerable y significativa de la cultura y la sociedad.

## ALCANCE DEL PROYECTO

El lenguaje de programación implementado por el compilador incorporará componentes esenciales que se encuentran en los lenguajes de programación funcionales, incluyendo operaciones aritméticas y lógicas, declaraciones de variables, condiciones *if* para declaraciones condicionales, ciclos *while* y *do-while* para control iterativo, y ciclos *loop* para procesos iterativos. Para mejorar la accesibilidad y comprensión por parte del usuario, el lenguaje utilizará conceptos relacionables, sustituyendo los términos técnicos por alternativas más familiares.

Además, el compilador introducirá un nuevo tipo llamado *datatable*, que permitirá nombrar columnas y ofrecerá funcionalidad para el análisis estadístico basado en columnas. Esta característica permitirá a los usuarios realizar operaciones básicas de análisis de datos dentro del propio lenguaje, sirviendo como plataforma introductoria para personas interesadas en explorar técnicas de análisis de datos. El alcance del proyecto incluirá el diseño y la implementación del compilador, garantizando que traduce la sintaxis y la semántica del lenguaje en código ejecutable.

La documentación del proyecto incluirá un manual de usuario que ofrecerá una guía sobre el uso del lenguaje y las funciones del compilador. En él se tratarán temas como la sintaxis y la semántica del lenguaje, el uso de variables, las sentencias condicionales, las estructuras de control iterativas, las operaciones con tablas de datos y las técnicas básicas de análisis de datos. La documentación también incluirá ejemplos y tutoriales para ayudar a los usuarios a comprender y utilizar el lenguaje con eficacia.

Sin embargo, es importante señalar que el proyecto no abarcará conceptos avanzados de programación ni técnicas complejas de análisis de datos. Por el contrario, se centrará en establecer una base sólida en los fundamentos de la programación y proporcionar un punto de entrada para que las personas con conocimientos mínimos de programación exploren el mundo de la programación y el análisis de datos.

# ANÁLISIS DE REQUERIMIENTOS

## 1. Sintaxis y semántica del lenguaje:

- Definir una sintaxis clara e intuitiva para ZenCode que se ajuste al objetivo de ser accesible para los principiantes.
- Implementar las reglas gramaticales y las construcciones del lenguaje.
- Diseñar el lenguaje para permitir la denominación de columnas y proporcionar funciones integradas para el análisis estadístico básico basado en columnas.

## 2. Funcionalidad del compilador:

- Desarrollar un léxico y un analizador sintáctico para analizar e interpretar el código fuente Zen, con una gestión de errores.
- Transformar el código analizado en representaciones intermedias adecuadas para la ejecución.
- Optimizar el código generado para lograr una ejecución y un rendimiento eficientes.

## 3. Entorno de ejecución:

- Crear un entorno de ejecución que pueda ejecutar los programas ZenCode compilados.
- Implementar un sistema de gestión de memoria para manejar variables, *arrays* y matrices de forma eficiente.
- Soportar la ejecución de operaciones aritméticas y lógicas, sentencias condicionales, estructuras de control iterativas y operaciones con tablas de datos.

## 4. Interfaz de usuario:

- Proporcionar mensajes de error y diagnósticos para ayudar a los usuarios a identificar y resolver problemas en su código.

## 5. Documentación:

- Preparar una documentación completa, incluido un manual de usuario, que explique la sintaxis, la semántica y el uso del lenguaje.
- Proporcione ejemplos y tutoriales para ayudar a los usuarios a comprender y utilizar ZenCode de forma eficaz.
- Documente las construcciones disponibles del lenguaje, las funciones incorporadas y su uso en operaciones de análisis de datos.

## 6. Pruebas y control de calidad:

- Realizar pruebas exhaustivas para garantizar la corrección, fiabilidad y rendimiento del compilador y de los programas ZenCode ejecutados.

## DESCRIPCIÓN GENERAL DEL PROYECTO

La elaboración de ZenCode tuvo dificultades iniciales cuando me enfoqué en el desarrollo del compilador utilizando el lenguaje de programación C++. Al principio, la curva de aprendizaje asociada a la comprensión de los detalles del pársers automático que generaba un programa open-source para C++ y su aplicación en el desarrollo de compiladores fue considerable. Se invirtió mucho tiempo y esfuerzo en comprender los matices del lenguaje, lo que provocó un retraso en el progreso durante las fases iniciales.

Sin embargo, persistí en mi afán de conocimiento y me esforcé con determinación por superar los obstáculos que encontré. Aprovechando los conocimientos adquiridos en las clases de Compiladores, y cambiando completamente de esquema a lo que se tenía planteado —incluso reemplazando el lenguaje con el más versátil Python— empecé a aplicar los conceptos y técnicas aprendidos al desarrollo de ZenCode. Aprovechando los conocimientos adquiridos en estas clases, fuimos ganando impulso y empezaron a haber progresos tangibles en el proyecto.

A pesar del difícil comienzo, a medida que profundicé en la comprensión y familiaridad con el propósito de las funciones y las partes individuales del proyecto, aceleré el progreso en el desarrollo del compilador ZenCode, llegando finalmente a un punto en el que podíamos implementar y probar características clave del lenguaje, su sintaxis y su semántica.

“

Creo que, a medida que avanzaba el proyecto, me di cuenta de lo equivocado que estaba cuando pensaba que comprendía cómo “piensan” las máquinas. Uno de los errores más comunes que tuve (además de la inercia de poner punto y coma al final de los estatutos cuando Python no lo ocupa) fue pensar que la máquina tenía que hacer todo en ese momento; o bien, que la función de la gramática/semántica iba a ser ya el proyecto completo, cuando todavía faltaba la máquina virtual, que es la que se encargaría de “darle vida” al proyecto. Ver toda la carrera aplicada en una sola materia valió la pena. Llevo toda la carrera esperando esta materia... y lo valió, aunque me dejara algunos días sin dormir”.

PD: La ironía del nombre fue divertida al principio...  
Esto no tuvo nada de pacífico.

  
-David Macías

# BITÁCORA

16 de abril, 2023: Semana 1

- La propuesta del proyecto está terminada. El lenguaje que utilizaremos es C++, porque es al que estamos más acostumbrados y, aunque nos gustaría experimentar con otro lenguaje (especialmente, consideramos Python y Ruby), no creo que el tiempo dé para que podamos aprender y sacarlo al mismo tiempo.

23 de abril, 2023: Semana 2

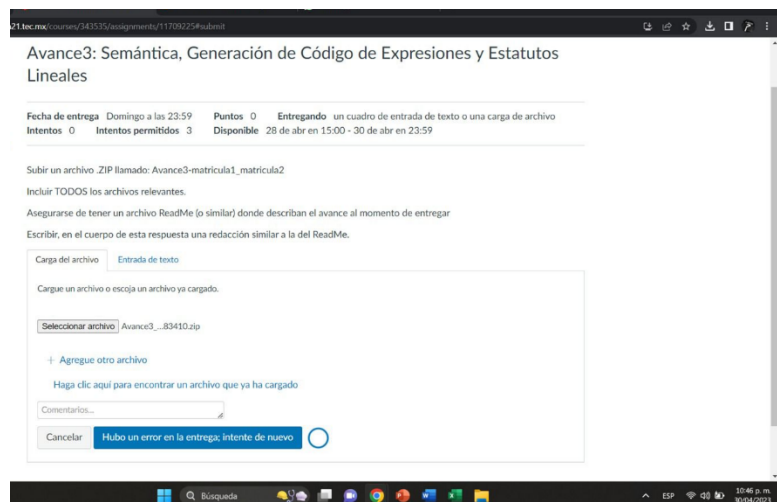
- Por las vacaciones, nos retrasamos un poco, pero el léxico ya está completamente definido, con todo y sus precedencias y algunas instrucciones para la máquina cuando lea, por ejemplo, enteros o flotantes, los convierta a ese tipo en el momento. Los estatutos están quedando, pero hay algunos con los que estamos teniendo ciertas dificultades.

27 de abril, 2023: Semana 3

- Estamos teniendo dificultades para crear el lexer/párser del lenguaje. El archivo con toda la gramática en EBNF ya está completo, pero dice que tiene varios errores shift/reduce. Estamos verificando que no tengamos ciclos infinitos o que hayamos definido algo mal, pero creo que es que algo no definimos correctamente en algún punto del archivo.

30 de abril, 2023: Semana 3

- Seguimos teniendo problemas con shift/reduce. Sería bueno si estuviéramos reduciendo el número, pero no parecen bajar de cincuenta. El cubo semántico está creado y está preparado para expandirse si fuera necesario agregar más atributos dependiendo de lo que vayamos viendo en las próximas semanas. Tampoco pudimos subir nuestro avance. Lo estuvimos intentando desde las 10:30 de la noche, y nos marcaba error.





4 de mayo, 2023: Semana 4

- Se generó el archivo lexer/párser. Estamos investigando cómo conectarlo con las funciones, o dónde definir las funciones o las instrucciones de semántica para el proyecto.

7 de mayo, 2023: Semana 4

- Debido al retraso y al fracaso con intentar comprender cómo funciona el archivo lexer/párser de Carburetta/C++, optaré por cambiar el lenguaje del proyecto de C++ a Python. El léxico está definido, así como la gramática.

11 de mayo, 2023: Semana 5

- Readaptación a Python completa.

12 de mayo, 2023: Semana 5

- Se hizo una reestructuración del proyecto y comenzó la definición de la semántica básica. Los estatutos de expresiones y asignación están definidos y en proceso.

14 de mayo, 2023: Semana 5

- Considerando el consejo de la maestra en cuanto a que no es necesario definir clases para los elementos del programa, se hizo una segunda reestructuración de los archivos donde va a ir el proyecto. La semántica básica está lista y los errores fueron corregidos.

21 de mayo, 2023: Semana 6

- La semántica de estatutos cíclicos: (*if*, *while*, *do-while*) está completa y funcionando, sin embargo, tiene unos ligeros errores que hay que corregir en cuanto a la respuesta que tiene a la ausencia de corchetes en el parseo del siguiente estatuto.

24 de mayo, 2023: Semana 7

- El estatuto *for* está completo y generando correctamente los cuádruplos (con un ligero toque personal). También, los anteriores están funcionando. De acuerdo con el profesor, vamos bien.

29 de mayo, 2023: Semana 7

- Los estatutos para funciones están listos y funcionando, aunque con ciertas combinaciones aparecen errores en la parametrización de las funciones.



2 de junio, 2023: Semana Final

- Los cuádruplos de funciones y arreglos ya están funcionando de manera correcta, y la indexación a arreglos y matrices genera los cuádruplos correctos. Queda pendiente definir la manera en la que se van a representar los elementos *pointer* de los cuádruplos para la máquina virtual.
- También, se hizo un rescoping del proyecto y se definió regresar a la idea original de hacer una clase que maneje valores como una tabla de SQL y que las columnas sean indexadas por nombre (*string*) en vez de por índice.

4 de junio, 2023: Semana Final

- La máquina virtual está completa. Se hizo un cambio completo con el administrador de memoria porque dar valores definidos desde los cuádruplos para la unidad de memoria en la que se va a alojar va a generar problemas en recursividad. Ahora, un segundo administrador se encarga de renombrar las variables con un código único basado en su función padre, su tipo y un identificador. El administrador original ahora administra en la máquina virtual y genera las direcciones de memoria como se vayan ocupando y las guarda en el "Tesseract" para traducción de identificador (*meimei*) a dirección.

## DESCRIPCIÓN DEL LENGUAJE



Imagina un lenguaje de programación diseñado para tender un puente entre el mundo de la programación y quienes acaban de iniciar su camino en este fascinante ámbito de las tecnologías. ZenCode, como bien se llama, pretende proporcionar una introducción accesible a los conceptos de programación y análisis de datos, ofreciendo un punto intermedio entre las complejidades de lenguajes como C++ y la simplicidad de un entorno para principiantes.

ZenCode adopta los componentes básicos de la programación funcional a la vez que incorpora sus propias peculiaridades. En esencia, ofrece operaciones aritméticas y lógicas familiares, lo que permite a los usuarios realizar cálculos sin esfuerzo. Pero en lugar de abrumar a los recién llegados con jerga, ZenCode opta por términos fáciles de entender.

Así, en lugar de "float" o "double", trabajarás con "decimal", ofreciendo una sensación de familiaridad a aquellos acostumbrados a los números cotidianos. Y para que el proceso de aprendizaje sea más atractivo, ZenCode sustituye el tradicional bucle "for" por el término más accesible "loop", haciendo que los procesos iterativos parezcan una progresión natural.

Una de las características destacadas de ZenCode es la introducción de un tipo conocido como "datatable". Esta innovadora adición permite a los usuarios trabajar con datos estructurados de una manera más intuitiva. Con la posibilidad de nombrar columnas y realizar análisis estadísticos, ZenCode abre las puertas a técnicas básicas de análisis de datos. Ya se trate de explorar tendencias, realizar cálculos o extraer información, el tipo datatable proporciona una experiencia fluida a los analistas de datos en ciernes.

ZenCode se esfuerza por crear un entorno que fomente la exploración y la comprensión. Al presentar una introducción suave y pacífica a la programación y el análisis de datos, ZenCode permite a los usuarios abrazar las posibilidades que se encuentran dentro de estos campos, alimentando su curiosidad y allanando el camino para un mayor crecimiento y descubrimiento.

## ERRORES

Existen una gran variedad de errores que son reconocidos por el intérprete de ZenCode. Estos están agrupados en nueve categorías:

**ZenDataTableCallError:** Este error se produce cuando hay un problema al llamar o utilizar un DataTable. Indica que puede haber un error en la sintaxis o en el uso de la DataTable, impidiendo que sea invocada o manipulada correctamente.

**ZenDataTableRedefinition:** Este error se produce cuando se intenta redefinir un DataTable que ya ha sido definido. DataTable es de uso exclusivo para una por cada función.

**ZenFunctionCallError:** Este error se produce cuando hay un error al llamar o utilizar una función Zen. Indica que puede haber un error en la sintaxis o en el uso de la función, impidiendo que sea invocada o ejecutada correctamente.


**ZenInvalidType:** Este error se produce cuando se encuentra un tipo no válido o no soportado en el programa ZenCode. Indica que el programa está intentando realizar una operación o asignar un valor que es incompatible con el tipo especificado.

**ZenRedefinedID:** Este error se produce cuando un identificador (como el nombre de una variable o función) se redefine en el programa. Significa que hay un conflicto de nombres, ya que el identificador ya ha sido utilizado y no puede ser redefinido.

**ZenRuntimeError:** Este error representa un error de ejecución general en el programa ZenCode. Aparece cuando se produce una condición inesperada o excepcional durante la ejecución del programa, indicando un problema que impide que el programa se ejecute correctamente.

**ZenSegmentationFault:** Este error indica una violación de acceso a memoria o fallo de segmentación, que suele producirse cuando un programa intenta acceder a memoria a la que no tiene permitido acceder. Sugiere un error crítico que puede llevar a la terminación o inestabilidad del programa.

**ZenTypeMismatch:** Este error se produce cuando hay un desajuste entre los tipos de datos esperados y reales en un programa ZenCode. Sugiere que el



programa está intentando realizar una operación o asignar un valor utilizando tipos de datos incompatibles.

**ZenUndefinedID:** Este error se produce cuando se hace referencia a un identificador (como el nombre de una variable o función) que no ha sido definido o declarado en el programa. Significa que el identificador al que se hace referencia no está reconocido dentro del ámbito del programa.

## DESCRIPCIÓN DEL COMPILADOR

## ESPECIFICACIONES

Fue realizado en Windows con Python y probado en diferentes plataformas, como Ubuntu y hasta Replit.com.

Para la compilación son necesarias las librerías `PLY`, `RE`, `NumPy` y `Statistics` de Python, aunque la gran mayoría de intérpretes ya incluyen las últimas tres. `PLY` viene incluida en los archivos del programa.

# LÉXICO

AND =	'&&'
ANG_L =	'<'
ANG_R =	'>'
ASSIGN =	'<<'
ASTRK =	'\''
BINOMIAL =	'binomial'
BOOL =	'bool'
BOX_L =	'['
BOX_R =	']'
BRACE_L =	{'}
BRACE_R =	}
CARET =	'^'
CHAR =	'char'
CHARACTER =	"(\\' \\\\ \\/ \" \\\\\\\\\\\\\\\\ \\\\n \\\\t [^'\\\\\\\\\\\\\\\\n\\\\t\\\\\\\\])"
COLON =	':'
COMMA =	','
COMMENT:	'\\#.*?(?=\\# \$) \\#.*'
CORR =	'CORR'
CREAD =	'cread'
CREATE =	'create'
CWRITE =	'cwrite'
DATA =	'data'
DEC =	'dec'
DECIMAL =	'\\d+\\.\\d+'
DO =	'do'
ELSE =	'else'
END =	'end'

EQUAL =	'='
FALSE =	'False'
FIT =	'fit'
FUNCTION =	'function'
ID:	'[a-z][a-zA-Z0-9_]*'
IF =	'if'
IN =	'in'
INT =	'int'
INTEGER =	'\d+'
LIST =	'list'
LOOP =	'loop'
MAIN =	'main'
MATRIX =	'matrix'
MAX =	'MAX'
MEAN =	'MEAN'
MIN =	'MIN'
MODULO =	'%'
NDASH =	'-'
NDIST =	'ndist'
NEQUAL =	'><'
NULL =	'Null'
OR =	'\ '
PARNL =	'('
PARNR =	')'
PASS =	'pass'
PLUS =	'+'
RANGE =	'range'
RETURN =	'return'
SD =	'SDEV'
SET =	'set'
SLASH =	'/'
SMCLN =	','
STRING =	'"([^\\"\\]*(\\\[^\\"\\]*)*)"'
SUM =	'SUM'
TEXT =	'text'
TILDE =	'~'
TRUE =	'True'
VALUES =	'values'
VAR =	'VAR'
VOID =	'void'

WHILE = 'while'  
WITH = 'with'  
WRITE = 'write'

## SINTAXIS

program : auxa auxb mains

auxa : vars auxa  
|  $\varepsilon$

auxb : fonctiondef auxb  
|  $\varepsilon$

statement : assign  
| conds  
| console  
| datadist  
| dataset  
| dowhiles  
| loops  
| vfunction  
| whiles

vars : SET type var SMCLN vars  
| CREATE datatable SMCLN vars  
|  $\varepsilon$

type : INT  
| CHAR  
| DEC  
| BOOL  
| TEXT

datatable : DATA ID BOX\_L BRACE\_L type COLON STRING

auxc : COMMA BRACE\_L type COLON STRING n0502 BRACE\_R auxc  
|  $\varepsilon$



var : auxd  
| auxf  
| auxh

auxd : ID auxe

auxe : COMMA ID auxe  
|  $\epsilon$

auxf : LIST ID BOX\_L INTEGER BOX\_R auxg

auxg : COMMA ID BOX\_L INTEGER BOX\_R auxg  
|  $\epsilon$

auxh : MATRIX ID BOX\_L INTEGER COMMA INTEGER BOX\_R auxi

auxi : COMMA ID BOX\_L INTEGER COMMA INTEGER BOX\_R auxi  
|  $\epsilon$

assign : ID ASSIGN expression SMCLN  
| direction ASSIGN expression SMCLN

conds : IF PARNT\_L condition PARNT\_R auxk  
| IF PARNT\_L condition PARNT\_R auxk ELSE auxk

auxk : statement  
| BRACE\_L statement auxl BRACE\_R

auxl : statement auxl  
|  $\epsilon$

condition : auxm comparison logicop condition  
| auxm comparison

auxm : TILDE  
|  $\epsilon$

comparison : expression compop expression

whiles : WHILE PARNT\_L condition PARNT\_R auxk

dowhiles : DO auxk WHILE PARNT\_L condition PARNT\_R SMCLN

loops : LOOP ID IN RANGE PARNT\_L expression COLON expression COLON  
condition PARNT\_R auxk

console : CREAD ASSIGN expression SMCLN  
| CWRITE ASSIGN auxn SMCLN

auxn : expression  
| STRING

constant : INTEGER  
| DECIMAL  
| CHARACTER  
| TRUE  
| FALSE  
| NULL

function : ID PARNT\_L params PARNT\_R

vfunction : ID PARNT\_L params PARNT\_R SMCLN

args : type ID auxp  
|  $\epsilon$

auxp : COMMA args auxp  
|  $\epsilon$

params : expression auxq  
|  $\epsilon$

auxq : COMMA expression auxq  
|  $\epsilon$

direction : ID BOX\_L expression BOX\_R  
| ID BOX\_L expression COMMA expression BOX\_R  
| ID COLON STRING BOX\_L expression BOX\_R

expression : term PLUS expression  
              | term NDASH expression  
              | term

term : base ASTRK term  
      | base SLASH term  
      | base

base : factor MODULO factor  
      | factor CARET factor  
      | factor

factor : ID  
        | constant  
        | NDASH constant  
        | direction  
        | function  
        | datacalc  
        | PARNT\_L expression PARNT\_R  
        | BRACE\_L condition BRACE\_R

logicop : AND  
          | OR

compop : ANG\_L  
          | ANG\_L EQUAL  
          | ANG\_R  
          | ANG\_R EQUAL  
          | EQUAL  
          | NEQUAL

dataset : WRITE IN ID BOX\_L INTEGER BOX\_R VALUES PARNT\_L auxr auxs

auxr : expression  
      | PASS

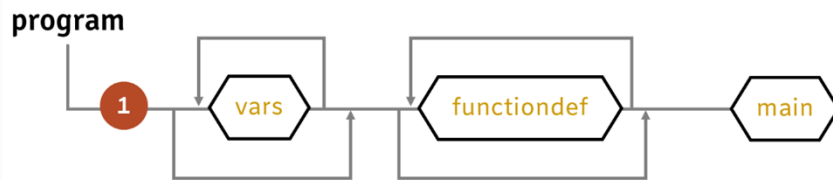
auxs : COMMA auxr auxs  
      |  $\epsilon$

datacalc : MAX PARNT\_L ID COLON STRING PARNT\_R  
| MIN PARNT\_L ID COLON STRING PARNT\_R  
| SUM PARNT\_L ID COLON STRING PARNT\_R  
| MEAN PARNT\_L ID COLON STRING PARNT\_R  
| VAR PARNT\_L ID COLON STRING PARNT\_R  
| SD PARNT\_L ID COLON STRING PARNT\_R  
| CORR PARNT\_L ID COLON STRING COMMA STRING PARNT\_R

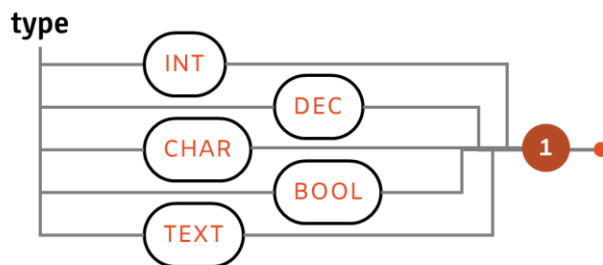
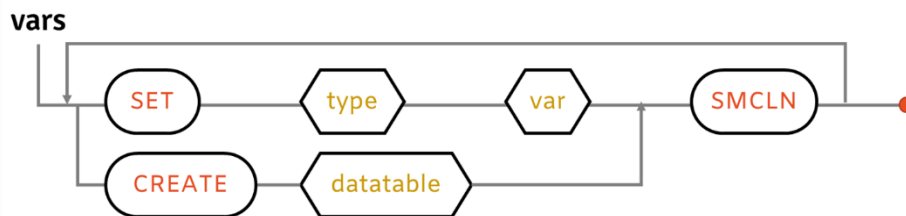
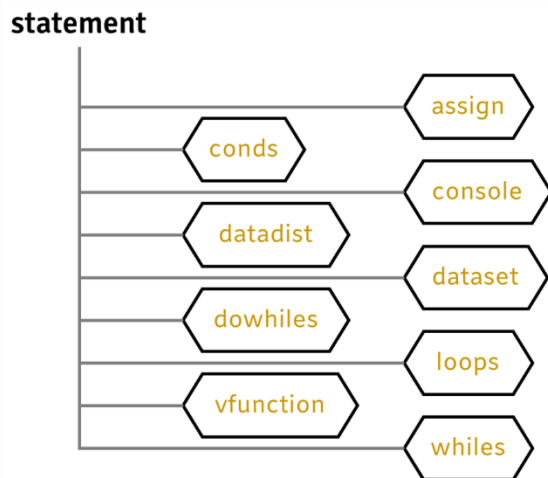
datadist : NDIST FIT ID COLON STRING WITH PARNT\_L expression COMMA  
expression PARNT\_R SMCLN  
| BINOMIAL FIT ID COLON STRING WITH PARNT\_L expression  
COMMA expression PARNT\_R SMCLN

mains : MAIN BRACE\_L vars statement auxl END BRACE\_R

## DIAGRAMAS DE SINTAXIS

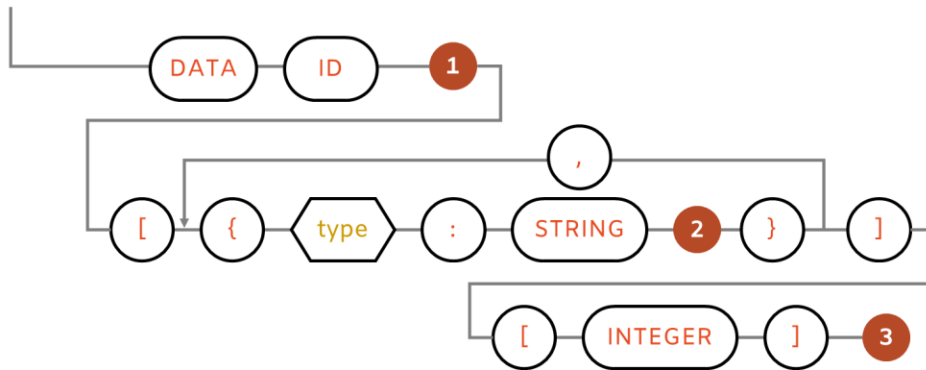


(1) goto main (pendiente), creación del Directorio de Funciones (FD).



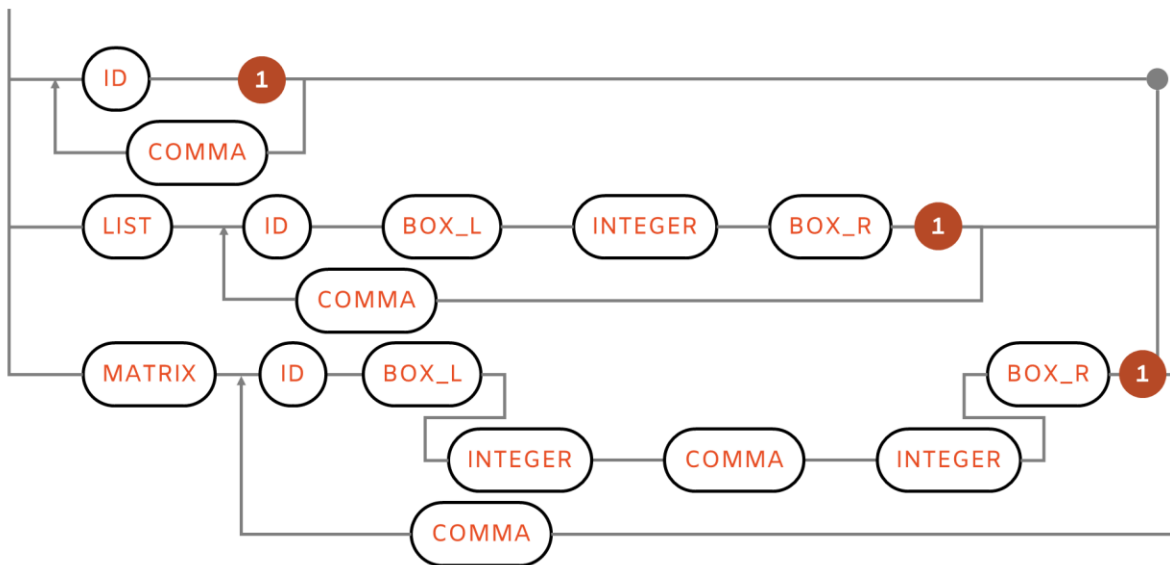
(1) current\_type = type

datatable



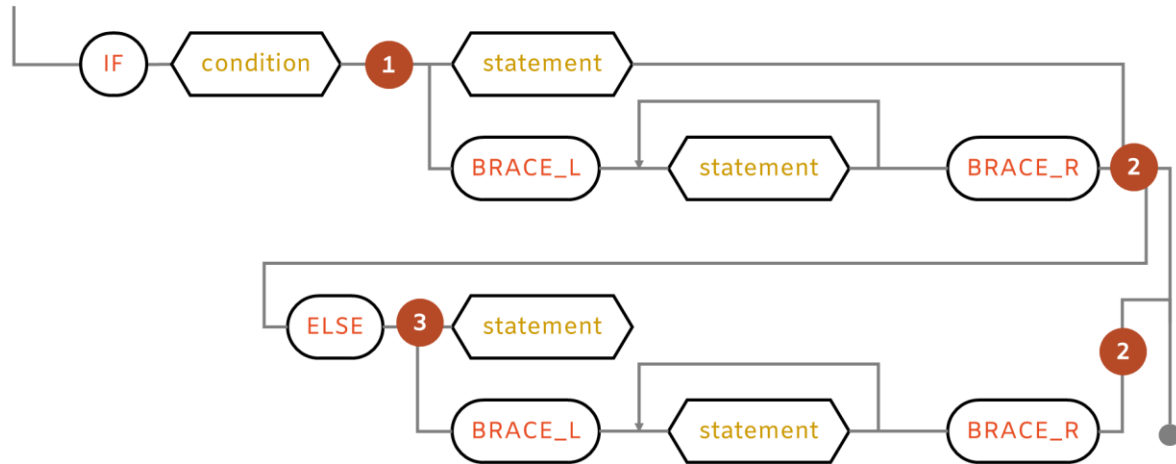
- (1)** Verificar si existe. Si no, crear tabla.
- (2)** Guardar nombre de columna en directorio de columnas (dentro de FD)
- (3)** Guardar cantidad de filas y columnas

var



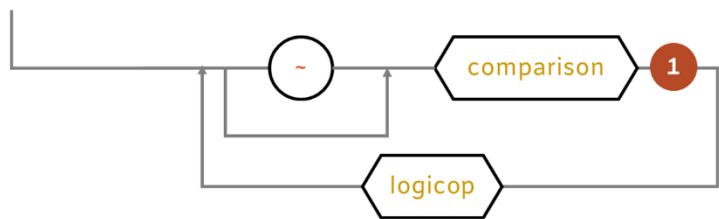
- (1)** Verificar si existe. Si no, añadir a la tabla de variables (VT) de la función.

**conds** (conditional statement)



- (1) Comprobar que la condición sea booleana. Goto-f (incompleto)
- (2) Completar el goto que queda pendiente.
- (3) Goto (incompleto)

**condition**



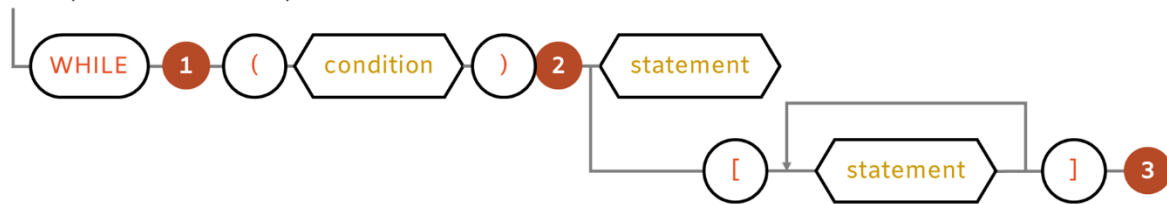
- (1) Verificar que los operandos sean booleanos y resolver operadores lógicos (and, or, not) pendientes.

**comparison**



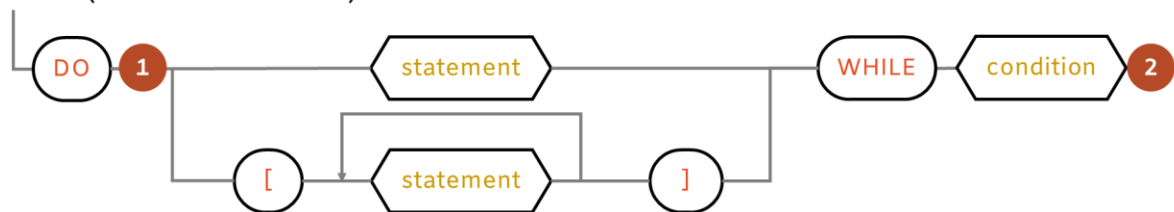


### whiles (*while* statement)



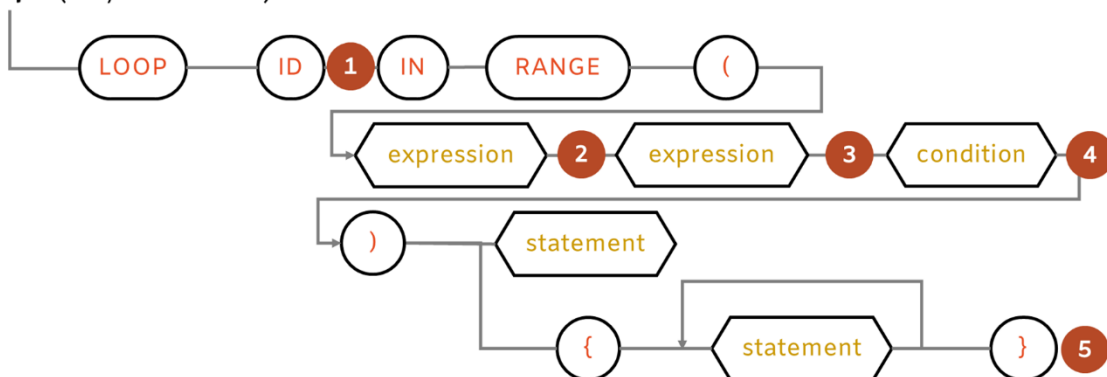
- (1) Añadir posición al JS
- (2) Verificar si la condición es booleana y crear goto-f (incompleto)
- (3) Goto a la comparación inicial en JS

### dowhiles (*do-while* statement)



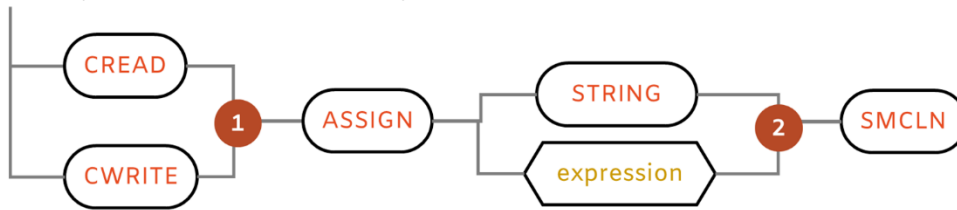
- (1) Añadir posición al JS
- (2) goto-t al inicio

### loops (*loop* statement)



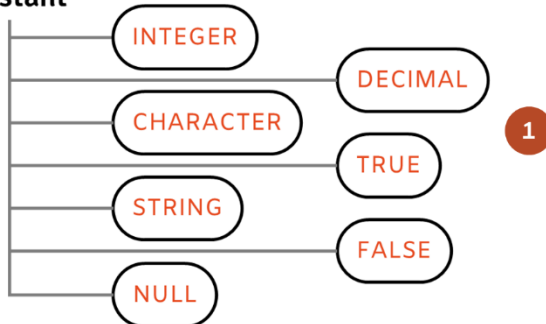
- (1) Verificar que el ID exista y sea entero.
- (2) Asignar Valor al ID y goto a tres estados delante
- (3) Sumar valor especificado al iterador y asignarlo al iterador
- (4) Verificar si la función se cumple y goto-t (incompleto)
- (5) Completa el goto-t y define un goto de vuelta a la reasignación

**console** (interaction with console)



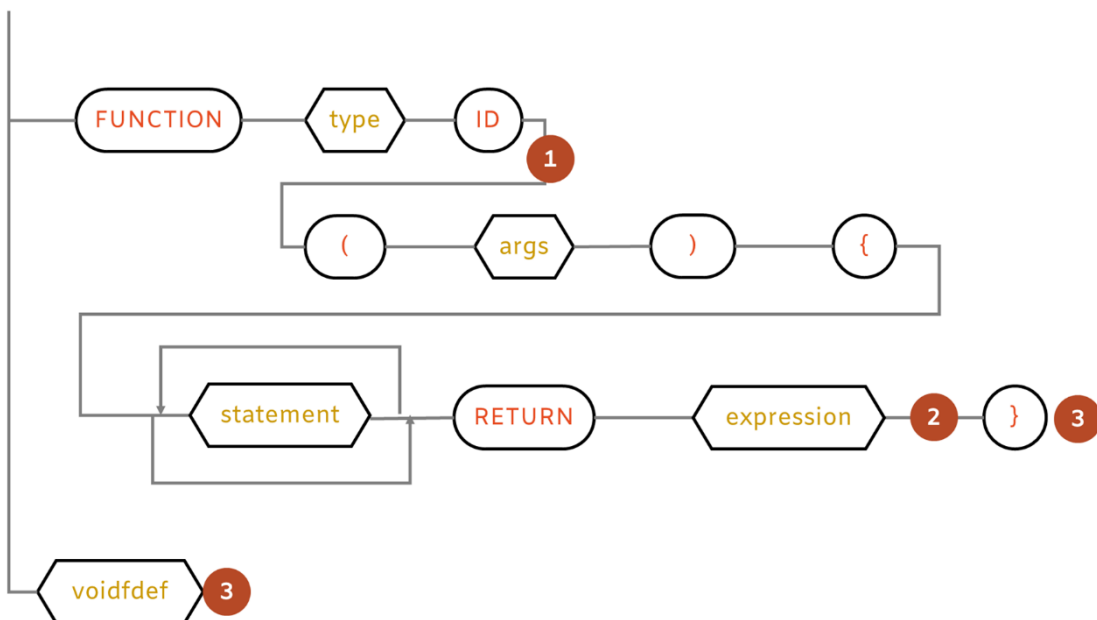
- (1) Entrar en estado read/write.
- (2) Utilizar valor y extraerlo del *stack* de operandos (OPS).

**constant**



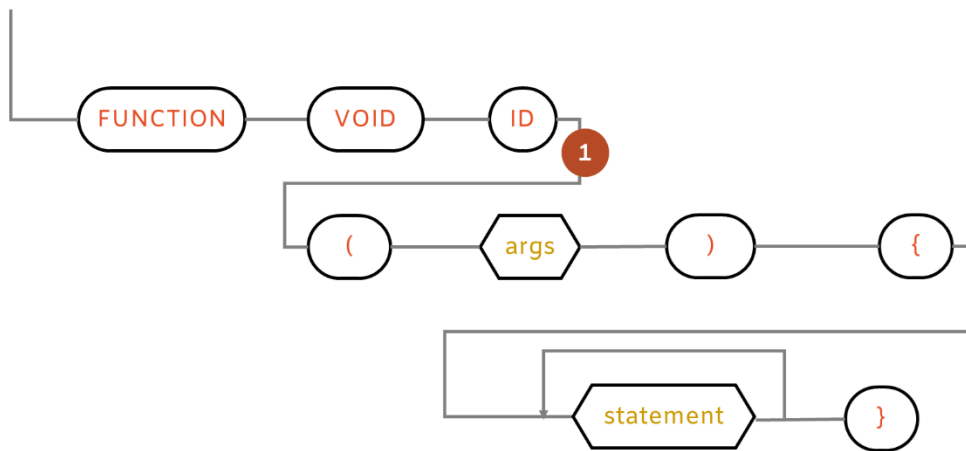
- (1) Asignar dirección en la tabla de constantes.

**functiondef**



- (1) Verificar que la función no haya sido previamente definida
- (2) Verificar que la expresión sea del mismo tipo que la función y return.
- (3) Llenar el FD con información de recursos y borrar la VT

voidfdef



(1) Verificar que la función no haya sido previamente definida

function



(1) Verificar que la función haya sido previamente definida y ERA

(2) Si la cantidad de parámetros coincide, gosub. Luego, asignar el valor del return de la función a una temporal.

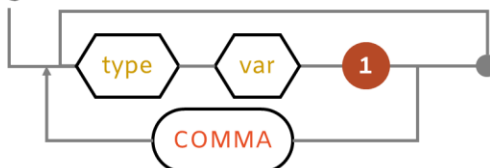
vfunction



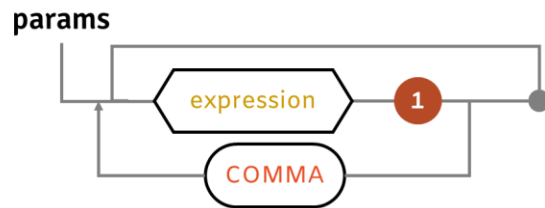
(1) Verificar que la función haya sido previamente definida y ERA

(2) Si la cantidad de parámetros coincide, gosub.

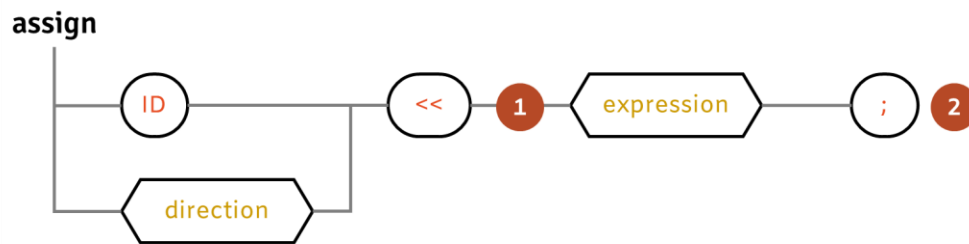
args



(1) Si está en declaración de función, añadir valor a la tabla de variables de la FD y añadir a la tabla de parámetros.

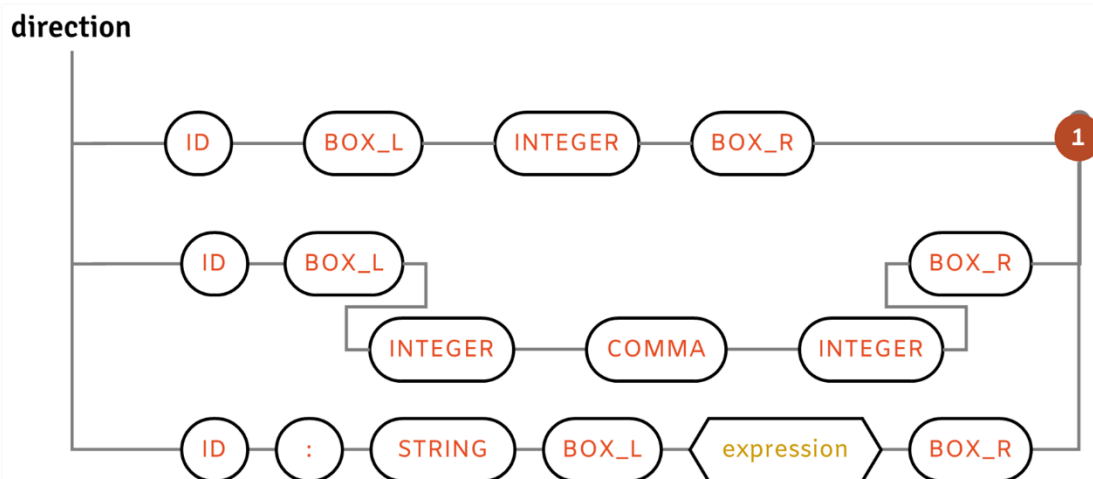


(1) Si el tipo coincide con el valor del parámetro  $x$ , generar cuádruplo de asignación de parámetros.



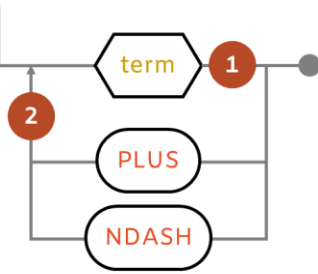
(1) Añadir '<<' al *stack* de operadores (OS)

(2) Si el tipo de la expresión puede ser casteado a la variable, asignar



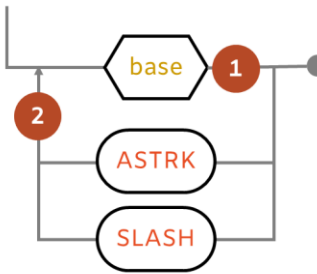
(1) Verificar que el índice esté dentro de los límites establecidos y regresar ubicación

### expression



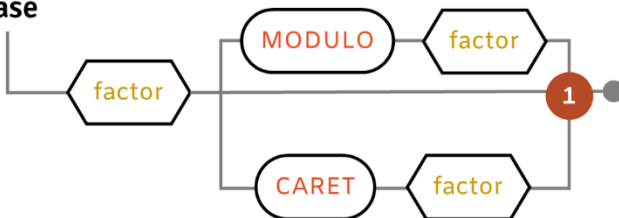
- (1) Si el top del OS es una suma o resta, realizar operación
- (2) Agregar al OS

### term



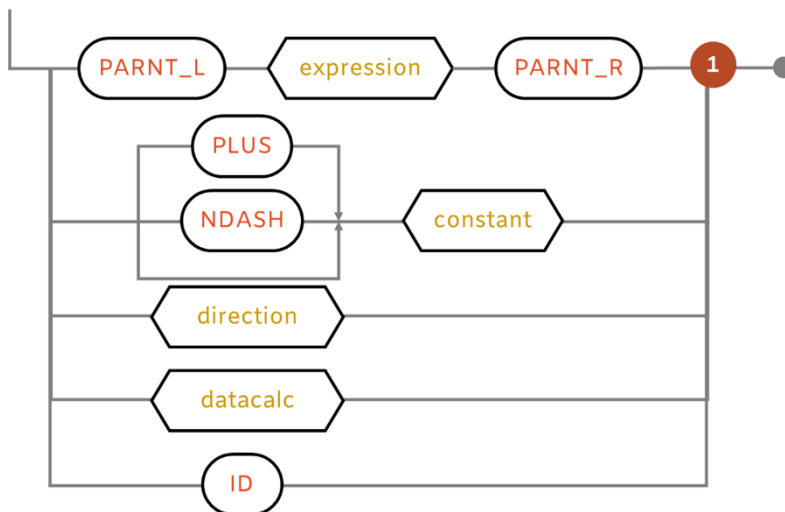
- (1) Si el top del OS es un asterisco o una división, realizar operación.
- (2) Agregar al OS

### base



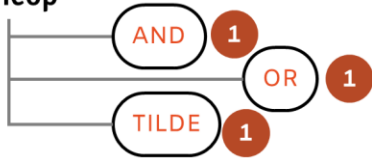
- (1) Si el operador top en el OS es módulo o exponencial, realizar la operación.

### factor



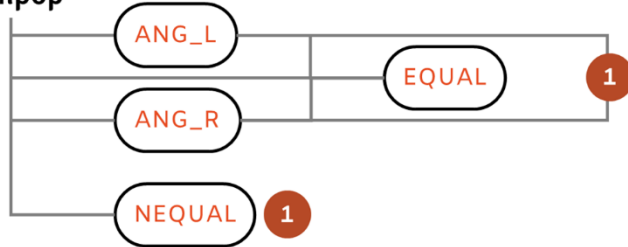
- (1) Añadir valor al OPS

logicop



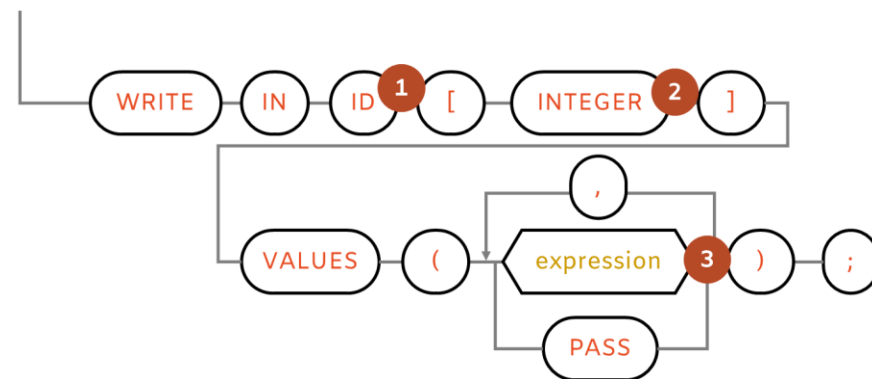
(1) Añadir valor al OS

compop



(1) Añadir valor al OS

direction

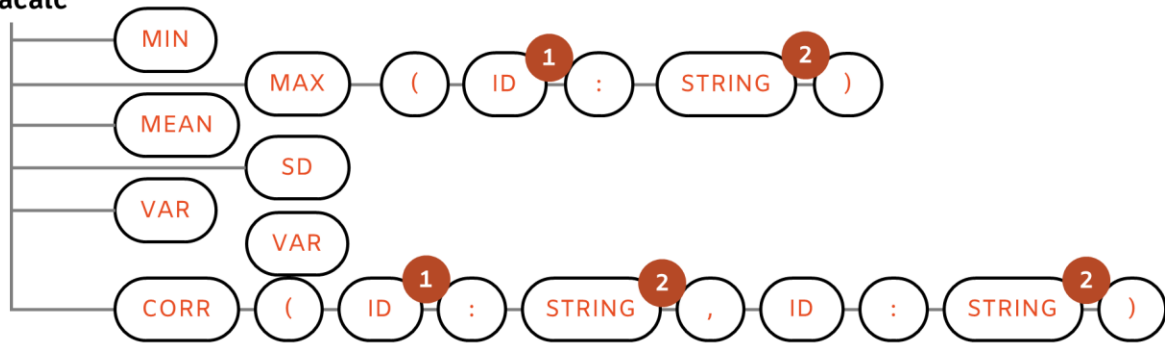


(1) Verificar que ID existe

(2) Verificar que el entero esté dentro de los límites de filas establecidos.

(3) Verificar que la longitud de la tabla no se haya superado todavía y que los tipos coinciden, y agregar valor a la tabla.

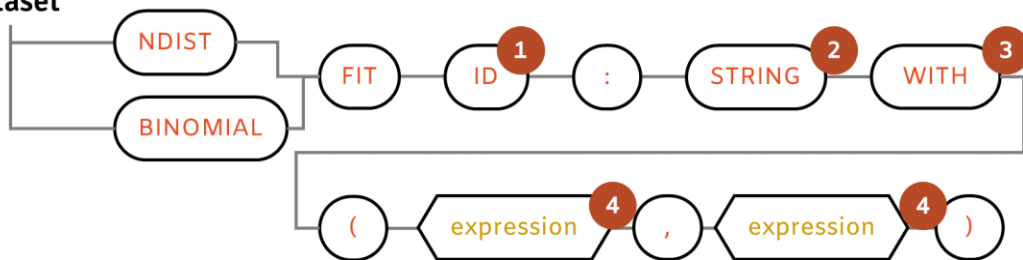
datacalc



(1) Verificar que ID existe

(2) Verificar que el nombre de la tabla exista y tenga un valor apropiado

dataset



(1) Verificar que ID existe

(2) Verificar que el nombre de la tabla exista y tenga un valor apropiado

(3) Guardar medidas de tabla para iteración.

(4) Verificar que la expresión tenga un valor numérico y compatible.



## CÓDIGOS DE OPERACIÓN Y SUS ATRIBUTOS

<i>OC</i>	<i>FUNCIÓN</i>	<i>INSTRUCCIÓN 1</i>	<i>INSTRUCCIÓN 2</i>	<i>INSTRUCCIÓN 3</i>
<b>0</b>	asignación	valor por asignar		ubicación
<b>1</b>	suma	sumando 1	sumando 2	ubicación
<b>2</b>	resta	minuendo	sustraendo	ubicación
<b>3</b>	multiplicación	multiplicando 1	multiplicando 2	ubicación
<b>4</b>	división	dividendo	divisor	ubicación
<b>5</b>	módulo/residuo	dividendo	divisor	ubicación
<b>6</b>	exponenciación	base	exponente	ubicación
<b>7</b>	mayor que	valor 1	valor 2	ubicación
<b>8</b>	menor que	valor 1	valor 2	ubicación
<b>9</b>	igual que	valor 1	valor 2	ubicación
<b>10</b>	diferente de	valor 1	valor 2	ubicación
<b>11</b>	mayor/igual que	valor 1	valor 2	ubicación
<b>12</b>	menor/igual que	valor 1	valor 2	ubicación
<b>13</b>	goto			estado
<b>14</b>	goto si verdadero	valor		estado
<b>15</b>	goto si falso	valor		estado
<b>16</b>	lectura consola			ubicación
<b>17</b>	escritura consola			ubicación
<b>18</b>	NOT lógico	valor 1	valor 2	ubicación
<b>19</b>	OR lógico	valor 1	valor 2	ubicación
<b>20</b>	AND lógico	valor 1	valor 2	ubicación
<b>21</b>	ARX*			función muestra
<b>22</b>	parámetro#	valor		futura ubicación
<b>23</b>	go a submétodo			estado
<b>24</b>	return			ubicación

25	verificar validez del índice	elemento	límite inferior	límite superior
26	verificar validez del índice (DT)	elemento	límite inferior	límite superior
27	verificar longitud de tabla (DT)	elemento		objetivo
28	valor máximo en columna (DT)	columna#	filas	columnas
29	valor mínimo en columna (DT)	columna#	filas	columnas
30	sumatoria de valores de la columna (DT)	columna#	filas	columnas
31	media de valores de la columna (DT)	columna#	filas	columnas
32	varianza de valores de la columna (DT)	columna#	filas	columnas
33	des. estándar de valores de la columna (DT)	columna#	filas	columnas
34	correlación de valores de dos columnas (DT)	columna1#	filas	columnas
34	...	columna2#	filas	columnas
35	preparar tabla para edición	índice/valor 1	longitud/valor 2	dirección inicial/método
36	asignar distribución normal a columna	columna#	filas	columnas
37	asignar distribución binomial a columna	columna#	filas	columnas
49	fin de edición			
99	fin de submétodo			
999	fin de programa			

\* Activation Record Expansion (Expansión de Registro de Activación)

## TABLA DE CONSIDERACIONES SEMÁNTICAS

### ENTEROS

	<<	+	-	*	/	%	^	com*	log**
<b>int</b>	int	int	int	int	int	int	int	bool	-
<b>dec</b>	Int	dec	dec	dec	dec	-	dec	bool	-
<b>char</b>	int	int	int	-	-	-	-	-	-
<b>bool</b>	int	int	int	int	int	-	-	bool	-

### DECIMALES

	<<	+	-	*	/	%	^	com*	log**
<b>int</b>	dec	dec	dec	dec	dec	-	dec	bool	-
<b>dec</b>	dec	dec	dec	dec	dec	-	dec	bool	-
<b>char</b>	-	-	-	-	-	-	-	-	-
<b>bool</b>	dec	dec	dec	dec	dec	-	-	bool	-

### CARACTERES

	<<	+	-	*	/	%	^	com*	log**
<b>int</b>	char	char	char	int	int	-	-	-	-
<b>dec</b>	-	-	-	-	-	-	-	-	-
<b>char</b>	char	-	-	-	-	-	-	bool	-
<b>bool</b>	-	-	-	-	-	-	-	-	-

### BOOLEANOS

	<<	+	-	*	/	%	^	com*	log**
<b>int</b>	bool	-	-	-	-	-	-	bool	-
<b>dec</b>	-	-	-	-	-	-	-	bool	-
<b>char</b>	-	-	-	-	-	-	-	-	-
<b>bool</b>	bool	-	-	-	-	-	-	bool	bool

\*com: comparativos \*\* log: lógicos

# ESTRUCTURAS DE DATOS

## DIRECTORIO DE FUNCIONES

El directorio de funciones es una hipertabla hecha como una lista de tuplas en Python. Contiene, en este orden, el **nombre de la función**, su **tipo**, su **primer estado** en el esquema de cuádruplos, su **función madre** o la función desde donde se manda a llamar, una lista con sus **parámetros**, y otra más con su **tabla de variables**, así como un espacio para que Koan inserte una tupla con sus **recursos** cuando la ejecución de la semántica termine.

## TABLA DE VARIABLES

La tabla de variables también es una hipertabla que contiene generalmente el **nombre asignado** por el usuario, el **tipo** de la variable y su *meimei* o **identificador de compilación**. Variables estructuradas tienen, en su dirección principal, un cuarto campo que contiene una lista con sus **dimensiones**.

## CUÁDRUPLOS

Los cuádruplos están hechos como tuplas, principalmente para evitar que, durante el proceso de elaboración, hubiera alguna instrucción accidental que sobrescribiera los cuádruplos. Estos solamente son editables a través de una función en la semántica. Cada uno, como su nombre lo menciona, contiene cuatro campos (no son didácticos como las estructuras anteriores) y son almacenados en una lista llamada *schema*.

## LISTA DE CONSTANTES

La lista de constantes es una lista que contiene todas las constantes que son detectadas en la elaboración del código intermedio. Estas se asignan después en las direcciones posteriores a la 1500000, por lo que no necesitan ser administrados más que para verificar si ya se ha usado anteriormente.

## DIRECTORIO DE TABLAS

El directorio de tablas es una hipertabla que contiene los nombres de las columnas de las tablas generadas. Considerando que solo se permite crear una tabla por función, y una global, el directorio guarda los nombres que se asignen

a las columnas y su posición para la traducción posterior. Este directorio no pasa a la máquina virtual.

## PILAS

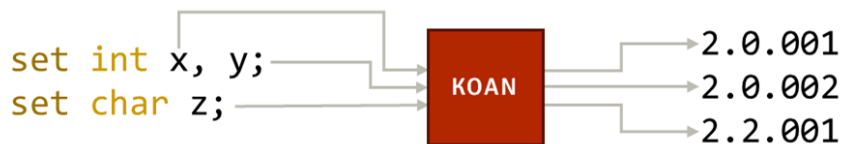
La compilación utiliza seis pilas para procesar la compilación: la **pila dimensional** contiene las dimensiones de las variables estructuradas que están siendo utilizadas en el momento. La **pila de saltos** contiene las direcciones que están esperando información sobre adónde deben “saltar” sea que suceda algo o no, o viceversa, envían su dirección en espera de que sea utilizada más adelante. La **pila de operandos** es una pila de tuplas que contiene la dirección del operando que está esperando a ser utilizado, así como su tipo. La **pila de operadores** es similar a la anterior, solamente que esta contiene los operandos que están esperando al parseo de otra instrucción para ser efectuados. La **pila de parámetros** recoge los parámetros de una función antes de comenzar y va comparando uno a uno los tipos del parámetro con lo que se espera. Su función está orientada al llamado de funciones como parámetros intermedios. Y, finalmente, la **pila de límites de parámetros** es el apoyo de la pila anterior para el llamado de funciones como parámetros. Esta contiene la longitud que tenía la pila de parámetros antes de comenzar a analizar los parámetros de otra función interna.

# ADMINISTRACIÓN DE MEMORIA

En el proceso de administración de memoria dentro de la compilación ZenCode, intervienen dos administradores: "Koan" y "MasterMind", junto con una estructura denominada "Tesseract".

El primer administrador, Koan, se inspira en un proceso de transición de la filosofía Zen. Su función es asignar variables utilizando IDs, llamados meimei (del japonés para "nombrar") en lugar de direcciones directas de memoria. Estos ID contienen información sobre la función de origen de la variable, su tipo y el orden en que se asignó. Mediante el uso de meimeis, Koan abstrae las direcciones físicas de memoria y proporciona un enfoque más flexible y organizado para la gestión de variables.

```
# current function ID: 2
```

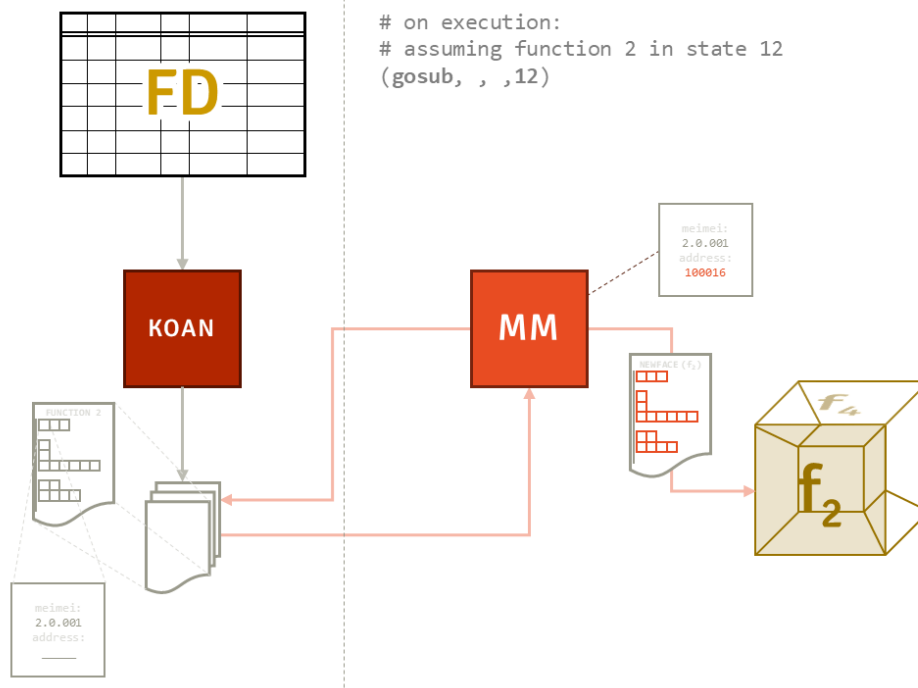


Una vez que la máquina recibe las instrucciones ZenCode, entra en acción el segundo administrador, MasterMind. MasterMind utiliza una estructura denominada "Tesseract", que puede visualizarse como un cubo formado por múltiples hipertablas. Cada cara del tesseracto representa una función activa diferente durante la ejecución del programa.

Cuando se procesan las instrucciones, Koan establece las instrucciones para el tesseracto como una plantilla, asignando las variables necesarias y sus respectivos ID. MasterMind entonces abre esta plantilla y comienza a reemplazar los IDs con direcciones de memoria reales basadas en la información generada por el Tesseract y las ubicaciones de memoria que MasterMind tiene consideradas como activas.

Durante la ejecución de una función, solo son accesibles las caras global y local del Tesseract. El Tesseract contiene todos los valores necesarios para estas funciones, lo que permite recuperar y manipular los datos con eficacia. Una vez que una función finaliza su ejecución, el Tesseract abandona la cara

correspondiente, que contiene todos los valores asociados a esa función, y vuelve a la cara abierta anteriormente.



Este sistema de gestión garantiza que el compilador ZenCode pueda manejar eficazmente la asignación de variables y la administración de memoria utilizando el concepto de IDs, la estructura Tesseract y la coordinación entre Koan y MasterMind. Este proporciona un enfoque estructurado y eficiente para la gestión de memoria dentro del proceso de compilación ZenCode. Al final, la memoria como tal es un diccionario que es llamado únicamente dándole funciones como identificador; las cuales son administradas por MasterMind, que las guarda en el Tesseract de acuerdo con cómo le haya especificado Koan.



# PRUEBAS

```
zen > zenhashi.py
6 import numpy
7 import statistics
8
9 # ---Memory-----
10 zenmind = {}
11
12 # ---Directories and Stacks-----
13 const_list = []
14 function = []
15 mastermind = [MasterMind(1), Mast
16 param_pusher = []
17 pipe = []
18 return_stack = []
19 schema = []
20 tesseract = []
21
22 # ---Pins-----
23 nextface = -1
24 part_one = True
25 show_vb = True
26 to_main = True
27
28 # ---Helper Functions-----
29 def addr(element):
30     if isinstance(element, str):
31         if element[0] == '&':
32             me, im, ei = element[1:1] cr
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
```

```
zen > zenhashi.py
6 import numpy
7 import statistics
8
9 # ---Memory-----
10 zenmind = {}
11
12 # ---Directories and Stacks-----
13 const_list = []
14 function = []
15 mastermind = [MasterMind(1), Mast
16 param_pusher = []
17 pipe = []
18 return_stack = []
19 schema = []
20 tesseract = []
21
22 # ---Pins-----
23 nextface = -1
24 part_one = True
25 show_vb = True
26 to_main = ~True
27
28 # ---Helper Functions-----
29 def addr(element):
30     if isinstance(element, str):
31         if element[0] == "6":
32             me im ai - elementf1.1 ce
33
34 > write in console:
35 184
36 > goto:215
37 > calc 1 + 1
38 > value kept in: 500096
39 > assigning 2 in 1.0.031
40 > check if 2 = 3
41 > value kept in: 800004
42 > F ignoring goto:226
43 > check if 1 is between 0 and 2
44 > calc 1 * 3
45 > value kept in: 500097
46 > check if 2 is between 0 and 3
47 > calc 3 + 2
48 > value kept in: 500098
49 > calc 5 + 100000
50 > value kept in: 500099
51 > write in console:
52 210
53 > goto:215
54 > calc 1 + 2
55 > value kept in: 500096
56 > assigning 3 in 1.0.031
57 > check if 3 = 3
58 > value kept in: 800004
59 > T then goto:226
60 > write in console:
61 }
62 > goto:208
63 > calc 1 + 1
64 > value kept in: 500095
65 > assigning 2 in 1.0.032
66 > check if 2 = 2
67 > value kept in: 800003
68 > T then goto:228
69 > write in console:
70 }
71 > done!
```

```
tests > mmult.zen
2 # Multiplicación de Matrices #
3
4 main {
5     set int matrix a[2,4], b[4,3], m[2,3];
6     set int i, j, k, x, y, z;
7
8     a[0,0] << 1;
9     a[0,1] << 2;
10    a[0,2] << 3;
11    a[0,3] << 4;
12    a[1,0] << 5;
13    a[1,1] << 6;
14    a[1,2] << 7;
15    a[1,3] << 8;
16    b[0,0] << 1;
17    b[0,1] << 2;
18    b[0,2] << 3;
19    b[1,0] << 4;
20    b[1,1] << 5;
21    b[1,2] << 6;
22    b[2,0] << 7;
23    b[2,1] << 8;
24    b[2,2] << 9;
25    b[3,0] << 10;
26    b[3,1] << 11;
27    b[3,2] << 12;
28    m[0,0] << 0;
29    m[0,1] << 0;
30
31 Zen Beta-2 Compiler...
32 File: tests/mmult
33
34 Matriz resultante:
35 {
36     {
37         70
38         80
39         90
40     }
41     {
42         158
43         184
44         210
45     }
46 }
47
48 >
```

# APÉNDICE

Origen de los Nombres:

**Koan** viene de una tradición zen que es una conversación de transición entre un maestro zen y un alumno. La idea es que, en esta conversación, el alumno deje de lado su razonamiento y pase a una respuesta intuitiva. Así, esta clase toma los nombres lógicos o funcionales y los convierte en nombres intuitivos.

**Meimei** es japonés para “nombrar” o “bautizar”, pero también es un acrónimo para la estructura del nombre: **m**emory **e**lement – **i**nitial **m**indscape (el nombre para el *variable type* en la memoria) – **e**lement’s **i**d.

**Hashi** (橋) es japonés para “puente” y un tipo de estructura zen. *Hashi* es el nombre del intérprete.