

ChocoPy

Nota: Para probar chocopy, utilizar el TryChocoPy que aparece aqui <https://chocopy.org/>

1. Introducción

- Es estáticamente tipado
- Diseñado para ser implementado como compilador
- Chocopy consiste en variables, funciones, clases y sentencias
- Chocopy no soporta diccionarios, funciones de primera clase(first-class function) y reflectividad introspectiva(reflective introspective)
- Solo un tipo durante toda la vida

2. Tour por ChocoPy

a. Notación:

- i. {id} = Nombre de la variable
- ii. {type} = Tipo de variable
- iii. {literal}= Valor inicial
- iv. {expr}= Cualquier valor del tipo específico
- v. {return type}= ->{type}

b. Variables

- i. {id}:{type}={literal}
- ii. Variables, funciones y clases deben tener distinto nombre

c. Funciones

- i. Aparecen en la parte superior de un programa o dentro de funciones o métodos
- ii. def{id} ({id}+{type},...,{id}+{type}){return type}
- iii. {return type}= ->{type}
- iv. Si el return type es vacío se debe retornar None
- v. Una función crea un nuevo scope

d. Clases

- i. class {id}{id}: {declarations}
- ii. Existe una clase por default llamada object

e. Valores

- i. Integers(int)=
 1. 32 bits= Hay que tener en cuenta el límite inferior y límite superior (-2147483648 a 2,147,483,647)
 2. Son inmutables
- ii. Booleans(bool)= True or False (No admite 0 o 1)
- iii. Strings(str)=
 1. Se delimita por " hola ". (No admite 'hola')
 2. Soporta len, index s[i] y concatenación s1+s2
 3. Solo admite entre 32 y 126 del rango ascii

- iv. Listas=
 - 1. Son mutables
 - 2. [1,2,3]
 - 3. Soporta las mismas operaciones que la string
- v. Objetos= Se debe referencia por ejemplo x=cow()
- vi. None= Solo se asigna a un objeto, clase o lista
- vii. Lista vacía []= puede ser asignado a un objeto o lista

f. Expresiones

- i. Literales e identificadores= Son de tipo str,bool e int
- ii. Expresiones de listas=[]
- iii. Expresiones aritméticas= int: {expr}+{expr}, {expr}-{expr},{expr}*{expr},{expr}/{expr},{expr}%{expr} y -{expr}. No soporta división porque da punto flotante y eso no es válido en Chocopy.
- iv. Expresiones lógicas= Soporta not{expr},{expr} and {expr} y {expr} or {expr}
- v. Expresiones de relación= int:{expr}<{expr}, {expr}<={expr}, {expr}>{expr}. También soporta {expr}=={expr} y {expr}!={expr} las cuales pueden ser de tipo int,bool y string
- vi. Expresiones condicionales= Maneja la estructura {expr1} if {expr0} else { expr2}
- vii. Concatenación de expresiones= {expr} + {expr} se puede usar para concatenar 2 string o listas
- viii. Expresiones de acceso= {expr}.{id} ejemplo x.y

g. Oraciones

- i. Condicionales y loops= Igual que en python.Soporta while y for
- ii. Oraciones de asignación=
 - 1. {id}={expr}
 - 2. {expr}.{id}={expr}
 - 3. {expr}[{expr}]= {expr}
 - 4. x=y.f=x[0]=1 Asigna el valor de 1 a toda la expresión
- iii. Clases predefinidas y funciones=
 - 1. print= Output str,int o bool el resto se aborda
 - 2. input= Retorna una str

3. Estructura léxica

- a. Estructura de línea
 - i. Líneas físicas=Son secuencias que terminan con fin de línea
 - ii. Líneas lógicas= ???
 - iii. Comentarios=Comienzan con # y terminan con una línea física
 - iv. Lineas vacias= Son ignoradas
 - v. Indentación=???
 - vi. Identificadores=Letras(Mayusculas o minusculas) de A a la Z, se puede barra al piso (_) y números excepto por el primer carácter

- vii. Keywords= No deben ser utilizados como variables. Estas palabras son: False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, or, pass, raise, return, try, while, with, yield, int, str.

Convenciones para el AFD

No admite caracteres fuera del rango solo ASCII del 32 a 126

Tipos de Datos		
A	Letras	A...Z,a...z
B	Números	0...9
C	Símbolo ASCII(32 al 126)	&,% ,etc..
F	Espacio	

Tipos de token

- <tipo_de_token,lexema,filas,columna>
- <tipo_de_token,filas,columna>

Tokens	
Convención de nombres	Token
Keywords	Se autoidentifica
tk_par_izq	(
tk_dos_puntos	:
tk_par_der)
tk_asig	=
tk_cadena	"Animal"
tk_ejecuta	->

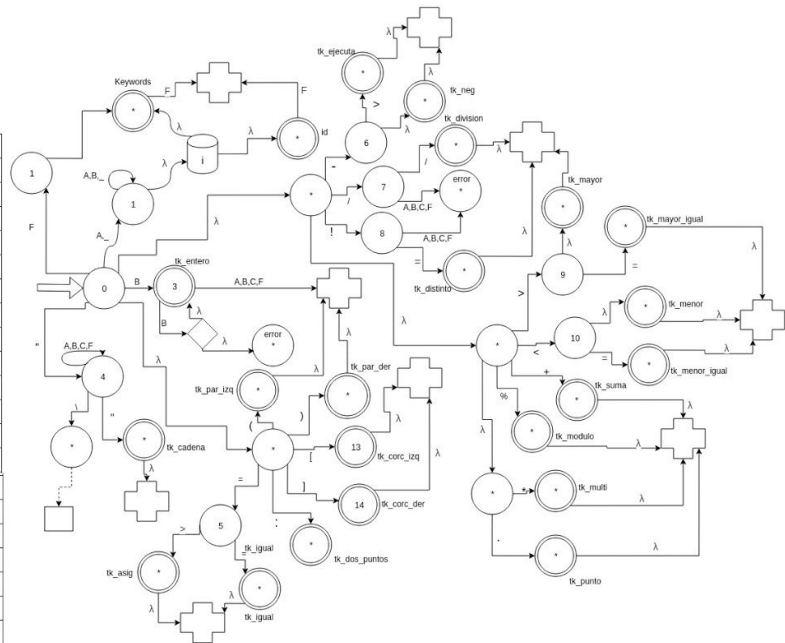
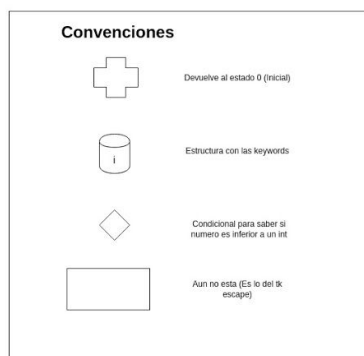
tk_escape	\n,\\,\'",\t
tk_punto	.
tk_distinto	!=
tk_igual	==
tk_menor	<
tk_mayor	>
tk_mayor_igual	>=
tk_menor_igual	<=
tk_suma	+
tk_neg	-
tk_multi	*
tk_division	//
tk_modulo	%
tk_corch_der]
tk_corch_izq	[
tk_entero	123 o cualquier número
tk_coma	,
id	Identificador
tk_igual_igual	==

Fake AFD

Se construyó un AFD que no cumple con todas las reglas para serlo (Fake AFD) pero permite visualizar a gran escala la idea que se utilizó para construir el código respectivo y también para facilitar el desarrollo dentro de los integrantes del equipo. Esta un poco desactualizado respecto al código porque cuando estuvimos programando nos dimos cuenta de cosas que faltaban pero se nos olvidó actualizarlo

Tipos de Datos		
A	Letras	A...Z,a...z
B	Números	0...9
C	Símbolo ASCII(32 al 126)	&,% etc...
F	Espacio	

Tokens	
Convención de nombres	Token
Keywords	Se autoidentifica
tk_par_izq	(
tk_par_der)
tk_asig	=
tk_cadena	"Animal"
tk_ejecuta	->
tk_escape	\n,\t,\",\'
__init__	__init__
tk_punto	.
tk_distinto	!=
tk_igual	==
tk_menor	<
tk_mayor	>
tk_mayor_igual	>=
tk_menor_igual	<=
tk_suma	+
tk_neg	-
tk_multi	*
tk_division	//
tk_modular	%
tk_cor_der]
tk_cor_izq	[
tk_entero	123 o cualquier número



https://drive.google.com/file/d/1AmtCJVbl4ghdgw_O7iof6li4slfEy1ld/view?usp=sharing

Nota:

- No se considera la palabra `__init__` como reservada debido a que está a comparación las palabras reservadas se puede utilizar como nombre de variable. Es decir el siguiente código donde `__init__` es nombre de variable es perfectamente válido en chocopy

```
__init__:int=1
```

```
print(__init__)
```

- Falta en el analizador léxico por implementar el apartado de los comentarios multilineas y las listas.