

SINTÁXIS



Sintáxis y Semántica

Un lenguaje de programación es una notación formal para describir algoritmos a ser ejecutados en una computadora

- Lenguaje de programación
 - Sintaxis
 - Semántica



Sintáxis y Semántica

■ Definiciones.

- Sintáxis: Conjunto de reglas que definen como componer letras, dígitos y otros caracteres para formar los programas
- Semántica: Conjunto de reglas para dar significado a los programas sintácticamente válidos.

v: array [1..10] of integer; ----- en Pascal
y
int v[10]; ----- en C



Sintáxis y Semántica

- ¿Cuál es la utilidad de definir y conocer la sintáxis y la semántica de un lenguaje? ¿Quiénes se benefician?
 - Programadores
 - Implementador (Compilador)
- La definición de la sintáxis y la semántica de un lenguaje de programación proporcionan mecanismos para que una persona o una computadora pueda decir:
 - Si el programa es válido y
 - Si lo es, qué significa



Sintáxis

■ **Características de la sintáxis**

- La sintáxis debe ayudar al programador a escribir programas correctos sintácticamente
- La sintáxis establecen reglas que sirven para que el programador se comuniquen con el procesador
- La sintáxis debe contemplar soluciones a características tales como:
 - Legibilidad
 - Verificabilidad
 - Traducción
 - Falta de ambigüedad



Sintáxis

La sintáxis establece reglas que definen cómo deben combinarse las componentes básicas, llamadas "**word**", para formar sentencias y programas.

- **Elementos de la sintáxis**
 - Alfabeto o conjunto de caracteres
 - identificadores
 - Operadores
 - Palabra clave y palabra reservada
 - Comentarios y uso de blancos



Sintáxis

■ Alfabeto o conjunto de caracteres

El código ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

Caracteres de control ASCII		
DEC	HEX	Símbolo ASCII
00	00h	NULL (carácter nulo)
01	01h	SOH (inicio encabezado)
02	02h	STX (inicio texto)
03	03h	ETX (fin de texto)
04	04h	EOT (fin transmisión)
05	05h	ENQ (enquiry)
06	06h	ACK (acknowledgement)
07	07h	BEL (timbre)
08	08h	BS (retroceso)
09	09h	HT (tab horizontal)
10	0Ah	LF (salto de línea)
11	0Bh	VT (tab vertical)
12	0Ch	FF (form feed)
13	0Dh	CR (retorno de carro)
14	0Eh	SO (shift Out)
15	0Fh	SI (shift In)
16	10h	DLE (data link escape)
17	11h	DC1 (device control 1)
18	12h	DC2 (device control 2)
19	13h	DC3 (device control 3)
20	14h	DC4 (device control 4)
21	15h	NAK (negative acknowle.)
22	16h	SYN (synchronous idle)
23	17h	ETB (end of trans. block)
24	18h	CAN (cancel)
25	19h	EM (end of medium)
26	1Ah	SUB (substitute)
27	1Bh	ESC (escape)
28	1Ch	FS (file separator)
29	1Dh	GS (group separator)
30	1Eh	RS (record separator)
31	1Fh	US (unit separator)
127	7Fh	DEL (delete)

Caracteres ASCII imprimibles								
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	\$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(72	48h	H	104	68h	h
41	29h)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	
61	3Dh	=	93	5Dh]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_	elCodigoASCII.com.ar		

ASCII extendido											
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	à	192	C0h	Ł	224	E0h	Ó
129	81h	ü	161	A1h	á	193	C1h	ł	225	E1h	ô
130	82h	é	162	A2h	â	194	C2h	Ł	226	E2h	ó
131	83h	ä	163	A3h	ã	195	C3h	ł	227	E3h	ô
132	84h	ä	164	A4h	ä	196	C4h	Ł	228	E4h	ö
133	85h	ä	165	A5h	å	197	C5h	Ł	229	E5h	õ
134	86h	ä	166	A6h	*	198	C6h	Ł	230	E6h	µ
135	87h	ç	167	A7h	*	199	C7h	Ł	231	E7h	þ
136	88h	ë	168	A8h	ž	200	C8h	Ł	232	E8h	ð
137	89h	ë	169	A9h	ž	201	C9h	Ł	233	E9h	ù
138	8Ah	è	170	AAh	~	202	CAh	Ł	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	Ł	235	EBh	Ü
140	8Ch	ï	172	ACH	¾	204	Ch	Ł	236	ECh	Ý
141	8Dh	ï	173	ADh	¾	205	CDh	Ł	237	EDh	ÿ
142	8Eh	À	174	Aeh	¾	206	CEh	Ł	238	Eeh	-
143	8Fh	À	175	AFh	¾	207	CFh	Ł	239	EFh	-
144	90h	É	176	B0h	¾	208	D0h	Ł	240	F0h	-
145	91h	æ	177	B1h	¾	209	D1h	Ł	241	F1h	±
146	92h	Æ	178	B2h	¾	210	D2h	Ł	242	F2h	‰
147	93h	ø	179	B3h	¾	211	D3h	Ł	243	F3h	‰
148	94h	ø	180	B4h	¾	212	D4h	Ł	244	F4h	‰
149	95h	ø	181	B5h	¾	213	D5h	Ł	245	F5h	‰
150	96h	ù	182	B6h	¾	214	D6h	Ł	246	F6h	‰
151	97h	ù	183	B7h	¾	215	D7h	Ł	247	F7h	‰
152	98h	ý	184	B8h	¾	216	D8h	Ł	248	F8h	‰
153	99h	Ö	185	B9h	¾	217	D9h	Ł	249	F9h	‰
154	9Ah	U	186	BAh	¾	218	DAh	Ł	250	FAh	‰
155	9Bh	ø	187	Bbh	¾	219	DBh	Ł	251	FBh	‰
156	9Ch	€	188	BCh	¾	220	DC	Ł	252	FC	‰
157	9Dh	ø	189	BDh	¾	221	DDh	Ł	253	FDh	‰
158	9Eh	x	190	BEh	¾	222	DEh	Ł	254	FEh	‰
159	9Fh	f	191	BFh	¾	223	DFh	Ł	255	FFh	‰

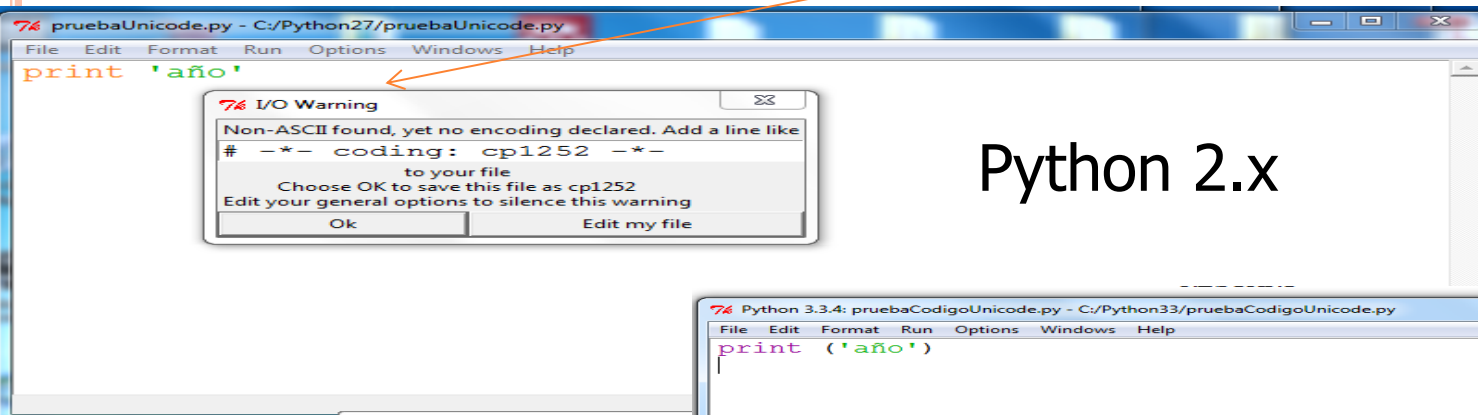
Sintáxis

■ Alfabeto o conjunto de caracteres

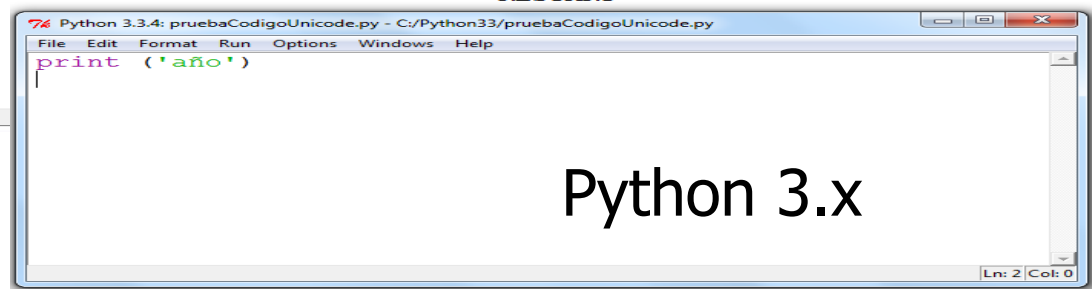
Importante: Tener en cuenta con qué conjunto de caracteres se trabaja sobre todo por **el orden** a la hora de comparaciones.

La secuencia de bits que compone cada carácter la determina la implementación.

-*- coding: utf-8 -*-



Python 2.x



Python 3.x

Sintáxis

■ Identificadores

- Elección más ampliamente utilizada: Cadena de letras y dígitos, que deben comenzar con una letra
- Si se restringe la longitud se pierde legibilidad

■ Operadores

- Con los operadores de suma, resta, etc. la mayoría de los lenguajes utilizan +, -. En los otros operadores no hay tanta uniformidad

■ Comentarios

- Hacen los programas más legibles

"El código es leído muchas más veces de lo que es escrito". Guido Van Roussen.



Sintáxis

■ Palabra clave y palabra reservada

Array do else if

- Palabra **clave** o **keywords**, son palabras claves que tienen un significado dentro de un contexto.
- Palabra **reservada**, son palabras claves que además no pueden ser usadas por el programador como identificador de otra entidad.
- Ventajas de su uso:
 - Permiten al compilador y al programador expresarse claramente
 - Hacen los programas más legibles y permiten una rápida traducción
- Soluciones para evitar confusión entre palabras claves e identificadores
 - Usar palabras reservadas
 - Identificarlas de alguna manera (Ej. Algol) usa 'PROGRAM 'END
 - Libre uso y determinar de acuerdo al contexto.

Ej: if=1 then if=0;



Sintáxis

Python: las palabras reservadas y sus versiones...

En las versiones 2.x el lenguaje cuenta con 31 palabras reservadas:

```
and as assert break class continue def del elif else except
exec finally for from global if import in is lambda not or
pass print raise return try while with yield
```

En la versión 3.x se quitaron de la lista **exec** y **print** han sido removidas, ya que ahora se presentan como funciones incorporadas por defecto.

Se han aadido: los térmiinos **nonlocal**, **True**, **False** y **None**

Por lo tanto, la lista de *keywords* en Python 3 resulta ser la siguiente.

```
False None True and as assert break class continue def del
elif else except finally for from global if import in is
lambda nonlocal not or pass raise return try while with yield
```

Esto lleva a que...



Sintaxis

**En una versión
de error y en
otra no**

```
Python 2.7.9 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more
>>> print=9
SyntaxError: invalid syntax
>>> |
```

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print=9
>>> |
Ln:131 Col:4
```

```
Python 2.7.9 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more
>>> True='hola'
>>> print True
hola
>>> |
```

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print=9
>>> True='hola'
SyntaxError: can't assign to keyword
>>> |
Ln:131 Col:4
```

Sintáxis

■ Estructura sintáctica

■ **Vocabulario o words**

- Conjunto de caracteres y palabras necesarias para construir expresiones, sentencias y programas. Ej: identificadores, operadores, palabras claves, etc.

Las words no son elementales se construyen a partir del alfabeto

■ **Expresiones**

- Son funciones que a partir de un conjunto de datos devuelven un resultado.
- Son bloques sintácticos básicos a partir de los cuales se construyen las sentencias y programas

■ **Sentencias**

- Componente sintáctico más importante.
- Tiene un fuerte impacto en la facilidad de escritura y legibilidad
- Hay sentencias simples, estructuradas y anidadas.



Sintáxis

■ Reglas léxicas y sintácticas.

- Diferencias entre mayúsculas y minúsculas
- Símbolo de distinto. En C != en Pascal <>

- Reglas léxicas: Conjunto de reglas para formar las **"word"**, a partir de los caracteres del alfabeto
- Reglas sintácticas: Conjunto de reglas que definen como formar las **"expresiones"** y **"sentencias"**

- El If en C no lleva "then", en Pascal si

La diferencia entre léxico y sintáctico es arbitrario, dan la apariencia externa del lenguaje



Sintáxis

■ Tipos de Sintáxis

■ **ABSTRACTA**

- Se refiere básicamente a la estructura

■ **CONCRETA**

- Se refiere básicamente a la parte léxica

■ **PRAGMÁTICA**

- Se refiere básicamente al uso práctico



Sintáxis

Ejemplo de sintáxis concreta y abstracta:.

while (x != y)

Uso de paréntesis

{

Forma de
encerrar un
bloque

};

(En C)

while x <> y do

begin

end

Símbolo de distinto

(En Pascal)

- Son diferentes respecto a la **sintáxis concreta**, porque existen diferencias léxicas entre ellas
- Son iguales respecto a la **sintáxis abstracta**, ya que ambas tienen la misma estructura

while condición
bloque



Sintáxis

Ejemplo de sintáxis pragmática:.

Ej1.


<> es mas legible que !=

Ej2.

En C y Pascal {} o begin-end pueden omitirse si el bloque esta compuesto por una sola sentencia

while (x!=y) x=y+1

Pragmáticamente puede conducir a error ya que si se necesitara agregar una sentencia debe agregarse el begin end o las {}.



Sintáxis

■ **Cómo definir la sintáxis**

- Se necesita una descripción finita para definir un conjunto infinito (conjunto de todos los programas bien escritos)
- Formas para definir la sintaxis:
 - Lenguaje natural. Ej.: Fortran
 - Utilizando la gramática libre de contexto, definida por Backus y Naun: BNF. Ej: Algol
 - Diagramas sintácticos son equivalentes a BNF pero mucho mas intuitivos



Sintáxis

■ BNF (Backus Naun Form)

- Es una notación formal para describir la sintaxis
- Es un metalenguaje
- Utiliza metasímbolos
 - $< > ::= |$
- Define las reglas por medio de “producciones”

Ejemplo:

$< \text{digito} > ::= 0|1|2|3|4|5|6|7|8|9$

No terminal

Se define como

Metasímblo

Terminales



Sintáxis

■ Gramática

- Conjunto de reglas finita que define un conjunto infinito de posibles sentencias válidas en el lenguaje.
- Una gramática esta formada por una 4-tupla

$$\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{S}, \mathbf{P})$$

Conjunto de
símbolos no
terminales

Conjunto de
símbolos
terminales

Símbolo distinguido
de la gramática que
pertenece a N

Conjunto de
producciones



Sintáxis

■ Árboles sintácticos

“Juan un canta manta”

- Es una oración sintácticamente incorrecta
- No todas las oraciones que se pueden armar con los terminales son válidas
- Se necesita de un **Método de análisis (reconocimiento)** que permita determinar si un string dado es valido o no en el lenguaje:
Parsing.
- El **parse**, para cada sentencia construye un **“árbol sintáctico o árbol de derivación”**

Sintáxis

- **Árboles sintácticos**

- Dos maneras de construirlo:

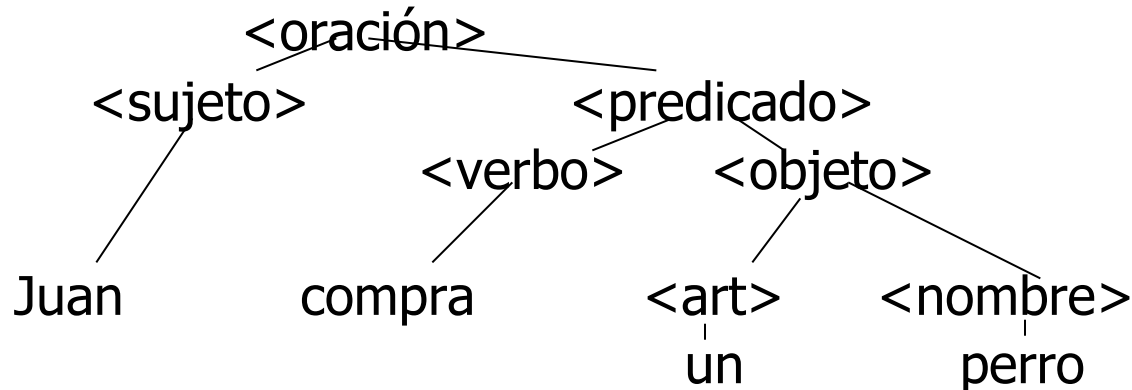
- **Método bottom-up**

- De izquierda a derecha
- De derecha a izquierda

- **Método top-down**

- De izquierda a derecha
- De derecha a izquierda

Ejemplo: árbol sintáctico de "oración". Top-down de izquierda a derecha



Sintáxis

■ **Árbol de derivación:**

- Ejemplo top-down de izquierda a derecha

<oración>	=>	<sujeto><predicado>
	=>	Juan <predicado>
	=>	Juan <verbo><objeto>
	=>	Juan compra <objeto>
	=>	Juan compra art><sustan>
	=>	Juan compra un <sustan>
	=>	Juan compra un perro

- Los compiladores utilizan el parse canónico que es el bottom-up de izquierda a derecha

- Otro ejemplo:
 - Expresiones simples de uno y dos términos
 - Posibles operaciones: $+$ / $*$ y $-$
 - Solo los operandos A, B y C
 - Ejemplo de expresiones válidas:
 - A
 - A+B
 - A-C
 - etc.



Sintaxis

■ Producciones recursivas:

- Son las que hacen que el conjunto de sentencias descripto sea infinito

- Ejemplo de producciones recursivas:

$\langle \text{natural} \rangle ::= \langle \text{digito} \rangle \mid \langle \text{digito} \rangle \langle \text{digito} \rangle \mid$
..... $\mid \langle \text{digito} \rangle \dots \dots \dots \langle \text{digito} \rangle$

- Si lo planteamos recursivamente

$GN = (N, T, S, P)$

$N = \{ \langle \text{natural} \rangle, \langle \text{digito} \rangle \}$ $T = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

$S = \langle \text{natural} \rangle$

$P = \{ \langle \text{natural} \rangle ::= \langle \text{digito} \rangle \mid \langle \text{digito} \rangle \langle \text{natural} \rangle,$
 $\langle \text{digito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \}$

- Cualquier gramática que tiene una producción recursiva describe un **lenguaje infinito**.

Sintáxis

■ Producciones recursivas:

- Regla recursiva por la izquierda
 - La asociatividad es por la izquierda
 - El símbolo no terminal de la parte izquierda de una regla de producción aparece al comienzo de la parte derecha
- Regla recursiva por la derecha
 - La asociatividad es por la derecha
 - El símbolo no terminal de la parte izquierda de una regla de producción aparece al final de la parte derecha



Sintaxis

■ Gramáticas ambiguas:

- Una gramática es ambigua si una sentencia puede derivarse de mas de una forma

$G = (N, T, S, P)$

$N = \{ \langle \text{id} \rangle, \langle \text{exp} \rangle, \langle \text{asig} \rangle \}$

$T = \{ A, B, C, +, *, -, /, := \}$

$S = \langle \text{asig} \rangle$

$P1 = \{$

$\langle \text{asig} \rangle ::= \langle \text{id} \rangle := \langle \text{exp} \rangle$

$\langle \text{exp} \rangle ::= \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \underbrace{\langle \text{exp} \rangle * \langle \text{exp} \rangle}_{\text{recursión}} \mid \langle \text{exp} \rangle - \langle \text{exp} \rangle \mid \langle \text{exp} \rangle / \langle \text{exp} \rangle \mid \langle \text{id} \rangle$

$\langle \text{id} \rangle ::= A \mid B \mid C$

$\}$

recursión



Sintaxis

■ Subgramáticas:

- Sea la gramática para identificadores $GI = (N, T, S, P)$

$N = \{ \langle id \rangle, \langle letra \rangle, \langle digito \rangle, \langle otro \rangle \}$

$T = \{ A, \dots, Z, 0, \dots, 1 \}$

$S = \langle id \rangle$

$P = \{ \begin{array}{l} \langle id \rangle ::= \langle letra \rangle \mid \langle letra \rangle \langle otro \rangle, \\ \langle otro \rangle ::= \langle letra \rangle \mid \langle digito \rangle \mid \langle letra \rangle \langle otro \rangle \mid \langle digito \rangle \langle otro \rangle, \\ \langle letra \rangle ::= A \mid B \mid C \mid \dots \mid Z \\ \langle digito \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9 \end{array} \}$

- Para definir la gramática GE, de expresiones, se puede utilizar la gramática de números y de identificadores.

GE se definiría utilizando las **subgramáticas** GN y GI

“La filosofía de composición es la forma en que trabajan los compiladores”

Sintáxis

■ Gramáticas libres de contexto y sensibles al contexto :

int e; a := b + c;

- Según nuestra gramática son sentencias sintácticamente válidas, aunque puede suceder que a veces no lo sea semánticamente.
 - El identificador está definido dos veces
 - No son del mismo tipo
- Una gramática libre de contexto es aquella en la que no realiza un análisis del contexto.
- Una gramática sensible al contexto analiza este tipo de cosas. (Algol 68).

Sintáxis

- **Otras formas de describir la sintaxis libres de contexto:**

- **EBNF.** Esta gramática es la **BNF extendida**
- Los metasimbolos que incorporados son:

[] elemento optativo puede o no estar

(|) selección de una alternativa

{ } repetición

*** 0 o mas veces**

+ una o mas veces



Sintaxis

■ Ejemplo con EBNF:

Definición números enteros en BNF y en EBNF

BNF

$\langle \text{enterosig} \rangle ::= + \langle \text{entero} \rangle \mid - \langle \text{entero} \rangle \mid \langle \text{entero} \rangle$

$\langle \text{entero} \rangle ::= \langle \text{digito} \rangle \mid \langle \text{entero} \rangle \langle \text{digito} \rangle$



Recursión

EBNF

$\langle \text{enterosig} \rangle ::= [(+|-)] \langle \text{digito} \rangle \{ \langle \text{digito} \rangle \}^*$

Eliminó la recursión y es mas fácil de entender

Sintáxis

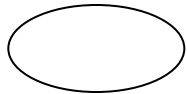
■ Diagramas sintácticos (CONWAY):

- Es un grafo sintáctico o carta sintáctica
- Cada diagrama tiene una entrada y una salida, y el camino determina el análisis.
- Cada diagrama representa una regla o producción
- Para que una sentencia sea válida, debe haber un camino desde la entrada hasta la salida que la describa.
- Se visualiza y entiende mejor que BNF o EBNF

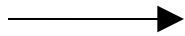


Sintáxis

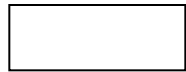
■ Diagramas sintácticos (CONWAY):



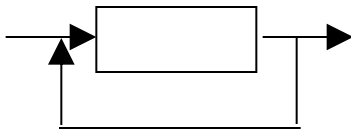
Terminales



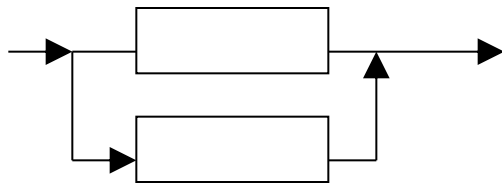
Flujo



No terminales



Repetición



Selección

Ej:

Programa



Sintáxis

■ Pensar:

Como definir una gramática para una expresión con operandos del tipo identificador y números y que refleje el orden de prioridades de las operaciones

