CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

2025

CONVOCATORIA COLABORADORES PARA PROYECTO "**EL BARRIO VA A LA UNIVERSIDAD**"

- ***18 años*** trabajando con la comunidad, capacitando a niños y jóvenes de diferentes barrios de la ciudad de La Plata y alrededores
- Acercando a los sectores más vulnerables a la Universidad, con el objetivo de que estos sectores puedan incorporar en sus imaginarios la posibilidad de continuar sus estudios en la Universidad.

¿Cómo se puede colaborar?

- Capacitaciones.
- Producción
- del material

INTRODUCCION Y EVALUACION DE LENGUAJES

INTRODUCCIÓN

Los lenguajes de Programación son el corazón de la Ciencia de Informática.

Son herramientas que usamos para comunicarnos con las máquinas y también con las personas.

¿CUÁL ES LA IDEA?

“El valor de un lenguaje se debe juzgar según la forma en que afecta la producción de Software y a la facilidad con la que puede integrarse a otras herramientas”

Se debe **introducir, analizar** y **evaluar** los conceptos más importantes de los lenguajes de programación.

QUÉ CONSEGUIREMOS

- Adquirir habilidad de **apreciar y evaluar** lenguajes, identificando los **conceptos** más importantes de cada uno de ellos y sus **límites y posibilidades**

- Habilidad para **elegir,** para **diseñar, implementar o utilizar** un lenguaje

- Enfatizar la **abstracción** como la mejor forma de manejar la complejidad de objetos y fenómenos

PARA QUÉ ESTUDIAR CONCEPTOS DE LENGUAJES

- Aumentar la capacidad para producir software.
- Mejorar el uso del lenguaje
- Elegir mejor un lenguaje
- Facilitar el aprendizaje de nuevos lenguajes
- Facilitar el diseño e implementación de lenguajes

CRITERIOS PARA EVALUAR LOS LENGUAJES DE PROGRAMACIÓN

Para poder evaluar los lenguajes necesitamos establecer criterios de evaluación.

CRITERIOS PARA EVALUAR LOS LENGUAJES DE PROGRAMACIÓN

- Simplicidad y legibilidad
- Claridad en los bindings
- Confiabilidad
- Soporte
- Abstracción
- Ortogonalidad
- Eficiencia

Analicemos algunas respuestas de docentes respecto al lenguaje que enseñan..

CRITERIOS PARA EVALUAR LOS LENGUAJES DE PROGRAMACIÓN

Textos obtenidos de trabajos finales de promoción

Pregunta a docentes de Cátedra de Lenguaje C:
Suele observarse que el código de C no es legible ¿Cree que esto se debe a la naturaleza del lenguaje o a malos hábitos de programación?¿Cuál cree que sea la causa?

Rta 1:
La sintaxis de C permite que pueda escribirse código muy compacto. Los alumnos deberían salir preparados para leer código de otros sin problemas. Muchos de los proyectos más grandes escritos en C tienen estrictas normas de estilo y esto permite que a pesar de ser compacto, el código sea suficientemente expresivo para todos.

Rta 2:
A la naturaleza del lenguaje sobre todo. El permitir código conciso y.....

Pregunta a docentes de Cátedra lenguaje Ruby:
¿Qué opina de la sintaxis de Ruby?

Rta:
La sintaxis de Ruby es, a mi parecer, simple y elegante. ..... y por otro lado es muy fácil desarrollar con él porque es muy conciso y preciso a la hora de implementar cualquier clase de funcionalidad

CRITERIOS PARA EVALUAR LOS LENGUAJES DE PROGRAMACIÓN

Pregunta a docentes de Cátedra lenguaje Ruby:
¿Usted considera que Ruby es un buen ejemplo de un lenguaje que presenta Ortogonalidad?. ¿A qué se debe?
Rta:
A mi parecer sí. El simple hecho de que toda sentencia del lenguaje sea una expresión (incluso las estructuras de control) permite realizar combinaciones y/o composiciones sin ninguna clase de limitación. Otro factor que favorece la ortogonalidad es que casi todo en este lenguaje son objetos; no recuerdo fácilmente casos en los cuales haya tenido problemas por encontrarme con algo que no sea/se comporte como un objeto.

Pregunta a docentes de Cátedra lenguaje Python:
¿Piensa que el lenguaje Python es indicado para iniciar a los alumnos en la programación?
Rta:
Si, Python es un lenguaje simple y fácil de enseñar. Es un lenguaje de tipado dinámico pero es fuertemente tipado, Es un lenguaje ortogonal porque se pueden combinar sus componentes. Su código es legible, confiable por ser fuertemente tipado y proveer manejo de excepcioenes,

En las respuestas se pueden observar criterios de evaluación aplicados, relacionados con: Simplicidad, Ortogonalidad, Confiabilidad, Expresividad, etc.

SIMPLICIDAD Y LEGIBILIDAD

- Los lenguajes de programación deberían:
  - Poder producir programas fáciles de escribir y de leer.
  - Resultar fáciles a la hora de aprenderlo o enseñarlo
  - La estructura subyacente del algoritmo y los datos que el programa representa deben quedan en manifiesto alinspeccionar el texto del programa.
- Ejemplo de cuestiones que atentan contra esto:
  - Muchas componentes elementales
  - Conocer subconjuntos de componentes
  - El mismo concepto semántico – distinta sintaxis (Por ej. Incrementar en uno una variable en C)
  - Distintos conceptos semánticos - la misma notación sintáctica (Por ej. el + de Python)
  - Abuso de operadores sobrecargados

CLARIDAD EN LOS BINDINGS

- Los elementos de los lenguajes de programación pueden ligarse a sus atributos o propiedades en diferentes momentos:
  - Definición del lenguaje
  - Implementación del lenguaje
  - En escritura del programa
  - Compilación
  - Cargado del programa
  - En ejecución
- La ligadura en cualquier caso debe ser clara

CONFIABILIDAD

- La confiabilidad está relacionada con la seguridad
  - Chequeo de tipos
    - Cuanto antes se encuentren errores menos costoso resulta realizar los arreglos que se requieran.
  - Manejo de excepciones
    - La habilidad para interceptar errores en tiempo de ejecución, tomar medidas correctivas y continuar.

SOPORTE

- Debería ser accesible para cualquiera que quiera usarlo o instalarlo
  - Lo ideal sería que su compilador o intérprete sea de dominio público

- Debería poder ser implementado en diferentes plataformas

Deberían existir diferentes medios para poder familiarizarse con el lenguaje: tutoriales, cursos textos, etc.

ABSTRACCIÓN

- Capacidad de definir y usar estructuras u operaciones complicadas de manera que sea posible ignorar muchos de los detalles.

- Concepto clave para manejar la complejidad, abstracción de procesos y de datos.

Le ahorran al programador tiempo de desarrollo al proporcionar algoritmos ya implementados.

ORTOGONALIDAD

- Ortogonalidad refiere a la posibilidad que ofrece el lenguaje de poder combinar sus elementos sin producir errores.

- Significa que un conjunto pequeño de constructores primitivos, puede ser combinado en número relativamente pequeño a la hora de construir estructuras de control y datos. Cada combinación es legal y con sentido.

El programador comprende mejor si tiene un pequeño número de primitivas y un conjunto consistente de reglas de combinación.

ORTOGONALIDAD

3.4 Devolución de valores

Una función en C sólo puede devolver un valor. Para devolver dicho valor, se utiliza la palabra reservada *return* cuya sintaxis es la siguiente:

```
return <expresión>;
```

Donde *<expresión>* puede ser cualquier tipo de dato salvo un array o una función. Además, el valor de la expresión debe coincidir con el tipo de dato declarado en el prototipo de la función. Por otro lado, existe la posibilidad de devolver múltiples valores mediante la utilización de punteros o estructuras. Dentro de una función pueden existir varios *return* dado que el programa devolverá el control a la sentencia que ha llamado a la función en cuanto encuentre la primera sentencia *return*. Si no existen *return*, la ejecución de la función continúa hasta la llave del final del cuerpo de la función *(})*. Hay que tener en cuenta que existen funciones que no devuelven ningún valor. El tipo de dato devuelto por estas funciones puede ser *void*, considerado como un tipo especial de dato. En estos casos, la sentencia *return* se puede escribir como *return* o se puede omitir directamente. Por ejemplo:

```
void imprime_cabecera(){
```

EFICIENCIA

- La eficiencia se relaciona con:
  - Tiempo y Espacio
    - capacidad de un programa para realizar una tarea de manera correcta y consumiendo pocos recursos, de memoria, espacio en disco, tiempo de ejecución, tráfico de red, etc.
  - Esfuerzo humano
    - que mejore el rendimiento del programador.
  - Optimizable
    - capacidad del lenguaje de programación de venir optimizado para tareas específicas

SINTAXIS

SINTAXIS Y SEMÁNTICA

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

- Lenguaje de programación → Sintaxis
- Lenguaje de programación → Semántica

SINTAXIS Y SEMÁNTICA

- **Sintaxis:** 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
```

SINTAXIS Y SEMÁNTICA

- ¿Cuál es la utilidad de definir y conocer la sintaxis y la semántica de un lenguaje? ¿Quiénes se benefician?
  - Programadores
  - Implementador (Compilador)
- La definición de la sintaxis 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.

SINTAXIS Y SEMÁNTICA

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

SINTAXIS

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

Elementos de la sintaxis

- Alfabeto o conjunto de caracteres
- Identificadores
- Operadores
- Comentarios y uso de blancos
- Palabra clave y palabra reservada

SINTAXIS

- **Alfabeto o conjunto de caracteres**
  - **Conjunto finito de símbolos admitidos en el lenguaje.**
  - **La secuencia de bits que compone cada carácter la determina la implementación.**

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

SINTAXIS

- **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	Simbolo 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	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
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

Unicode

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

UCS
UTF-16
UTF-7
UTF-32
UTF-8
SCSU

SINTAXIS

- **Alfabeto o conjunto de caracteres**

Latin-1 (ISO-8859-1: Western European)

1	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	nul	stx	sot	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
0x1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
0x2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
0x8	pad	hop	bph	nbh	ind	nel	ssa	esa	hts	htj	vts	pld	plu	ri	ss2	ss3
0x9	dcs	pu1	pu2	sts	cch	mw	spa	epa	sos	sgci	sci	csi	st	osc	pm	apc
0xa	nbsp	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	shy	®	¯
0xb	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
0xc	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
0xd	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
0xe	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
0xf	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Greek (ISO-8859-7)

G	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	nul	stx	sot	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
0x1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
0x2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
0x8	pad	hop	bph	nbh	ind	nel	ssa	esa	hts	htj	vts	pld	plu	ri	ss2	ss3
0x9	dcs	pu1	pu2	sts	cch	mw	spa	epa	sos	sgci	sci	csi	st	osc	pm	apc
0xa	nbsp	‘	’	£	xxx	xxx	¦	§	¨	©	xxx	«	¬	shy	xxx	—
0xb	°	±	²	³	΄	΅	Α	·	Ε	Η	Ι	»	Ο	½	Υ	Ω
0xc	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
0xd	Π	Ρ	xxx	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ	ά	έ	ή	ί
0xe	ΰ	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
0xf	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	ι	ϋ	ό	ύ	ώ	xxx

Cyrillic (ISO-8859-5)

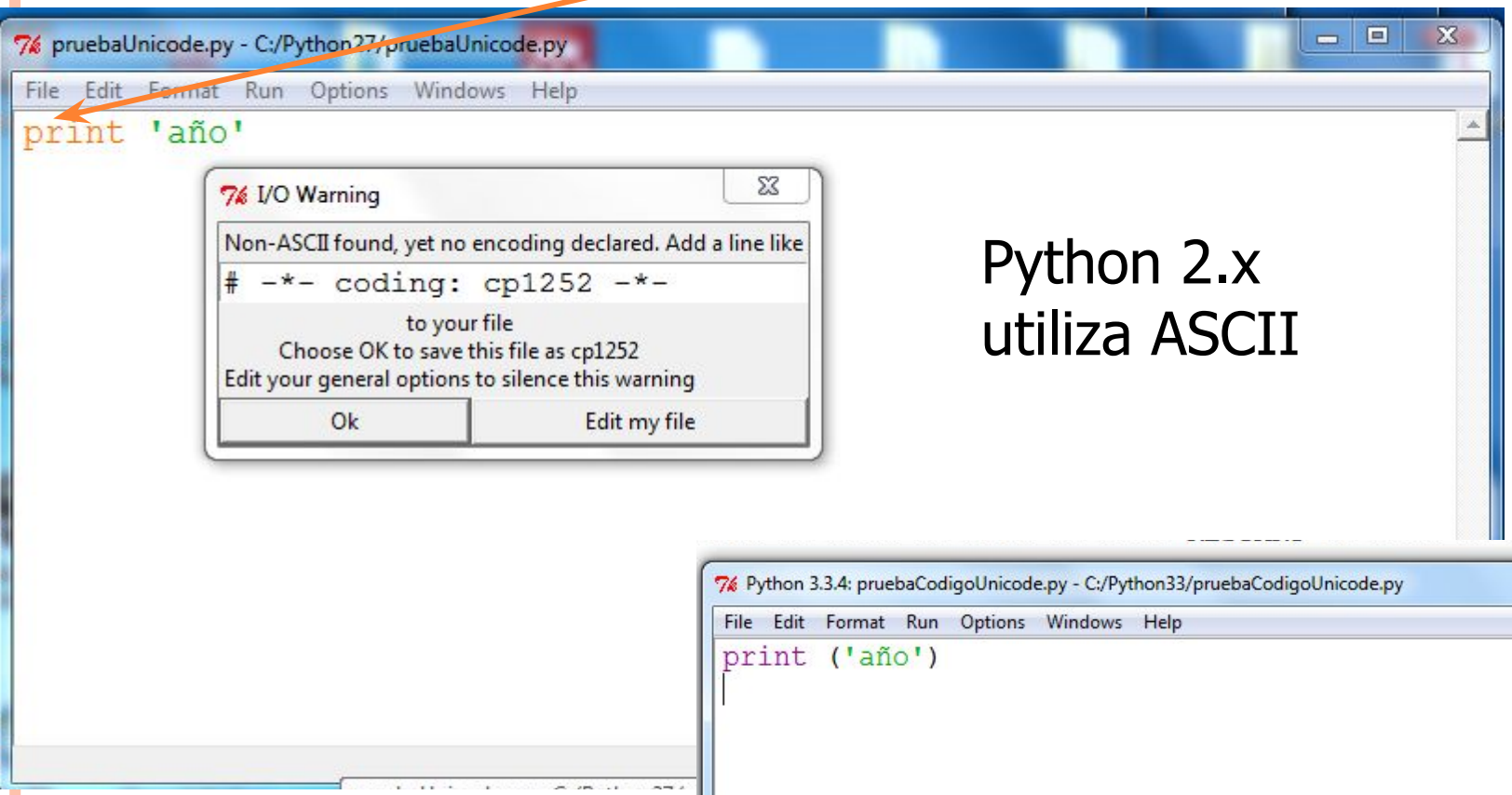
C	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	nul	stx	sot	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
0x1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
0x2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
0x8	pad	hop	bph	nbh	ind	nel	ssa	esa	hts	htj	vts	pld	plu	ri	ss2	ss3
0x9	dcs	pu1	pu2	sts	cch	mw	spa	epa	sos	sgci	sci	csi	st	osc	pm	apc
0xa	nbsp	Ё	Ђ	Ґ	Є	S	І	Ї	J	Љ	Њ	Ћ	Ќ	shy	Ў	Џ
0xb	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
0xc	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0xd	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0xe	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
0xf	№	ё	ђ	ґ	є	s	і	ї	j	љ	њ	ћ	ќ	§	ў	џ

Arabic (ISO-8859-6)

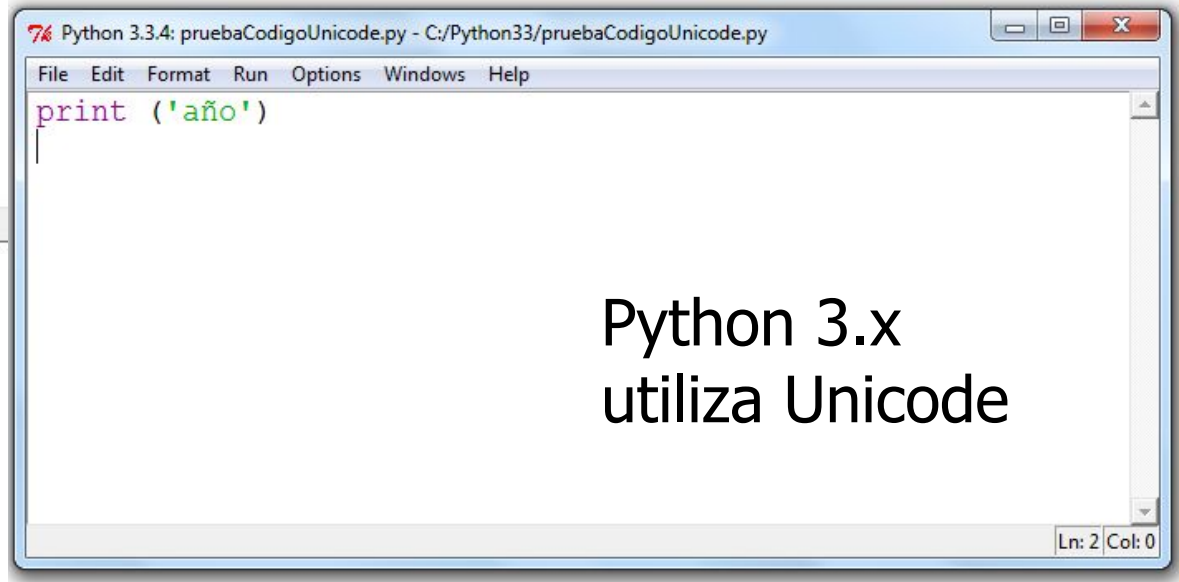
A	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	nul	stx	sot	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
0x1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
0x2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x3	٠	١	٢	٣	٤	٥	٦	٧	٨	٩	:	;	<	=	>	?
0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
0x8	pad	hop	bph	nbh	ind	nel	ssa	esa	hts	htj	vts	pld	plu	ri	ss2	ss3
0x9	dcs	pu1	pu2	sts	cch	mw	spa	epa	sos	sgci	sci	csi	st	osc	pm	apc
0xa	nbsp	xxx	xxx	xxx	¤	xxx	xxx	xxx	xxx	xxx	xxx	،	xxx	xxx	xxx	xxx
0xb	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	؛	xxx	xxx	xxx	؟
0xc	xxx	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د
0xd	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	xxx	xxx	xxx	xxx	xxx
0xe	ـ	ف	ق	ك	ل	م	ن	ه	و	ى	ي	ً	ٌ	ٍ	َ	ُ
0xf	ِ	ّ	ْ	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx

SINTAXIS

Se puede cambiar con #
-*- coding: utf-8 -*-



Python 2.x
utiliza ASCII



Python 3.x
utiliza Unicode

SINTAXIS

- **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 (creador de Python).

SINTAXIS

- **Palabra clave y palabra reservada**

  Array do else if

  - Palabra **clave** o **keywords**, son palabras 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

Ejemplos de lenguajes con uso de palabras reservadas:

- **C** ej.:auto,break ,case ,char ,const ,continue ,default ,do ,double ,else ,enum ,extern ,float ,for ,goto ,if ,int , etc
- **Pascal** ej.: absolute, and, array, begin, const, div, do, downto, else, if, in,label,mod,not,of, packed, procedure, record, set, shr, then,to, unit, uses, var, while, xor, etc

SINTAXIS

- **Palabra clave y palabra reservada**

  Array do else if

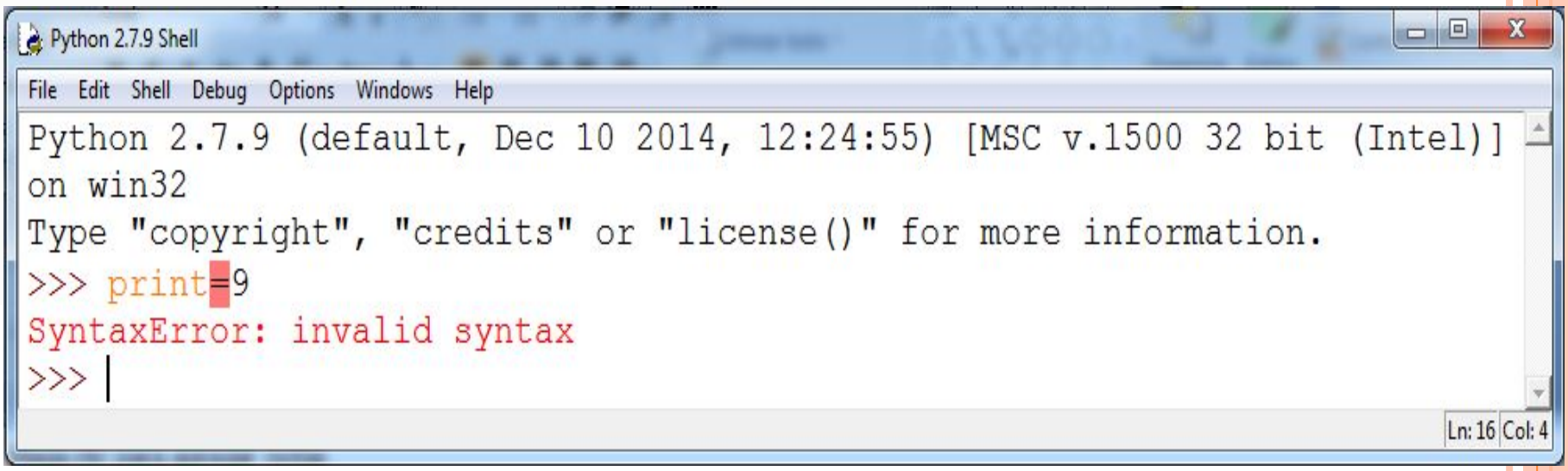
  Algunas alternativas:

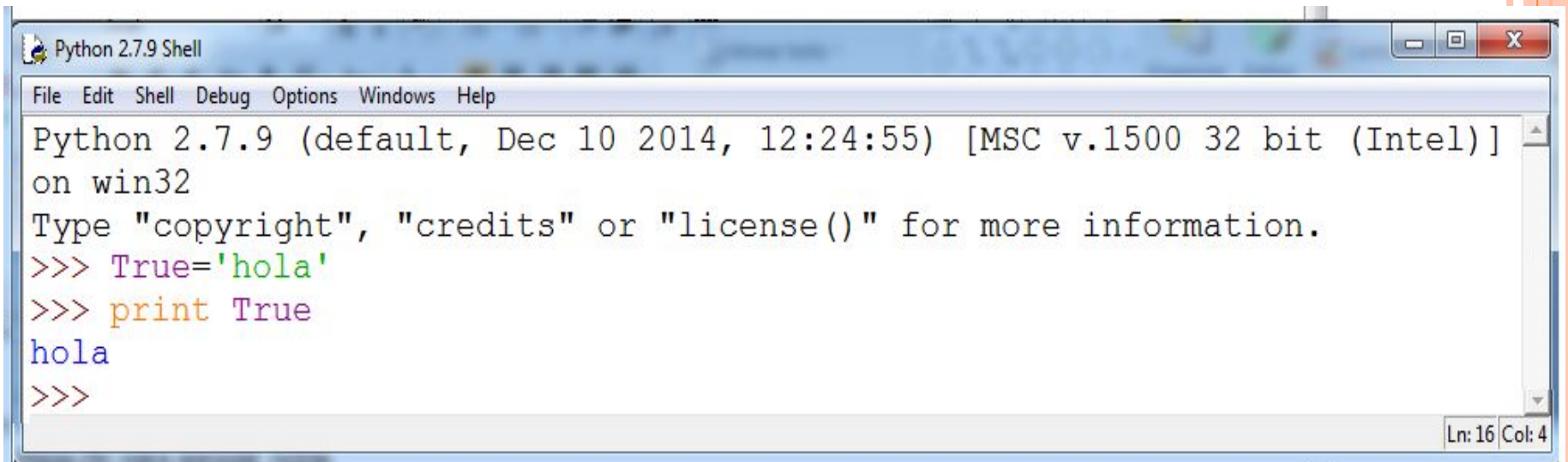
  - Usar palabras reservadas.
    - si se usan todas palabras reservadas es más fácil para el traductor, pero difícil para el programador, ya que debe conocerlas a todas. Además si se incorporan palabras reservadas nuevas no serán válidos los programas anteriores porque pueden haber utilizado esas palabras.
  - Identificarlas de alguna manera (Ej. Algol) usa ‘PROGRAM ‘END
  - Libre uso y determinar de acuerdo al contexto.

    Ej: if if=1 then if=0;

SINTAXIS

Python: las palabras reservadas y sus versiones…





SINTAXIS

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
```

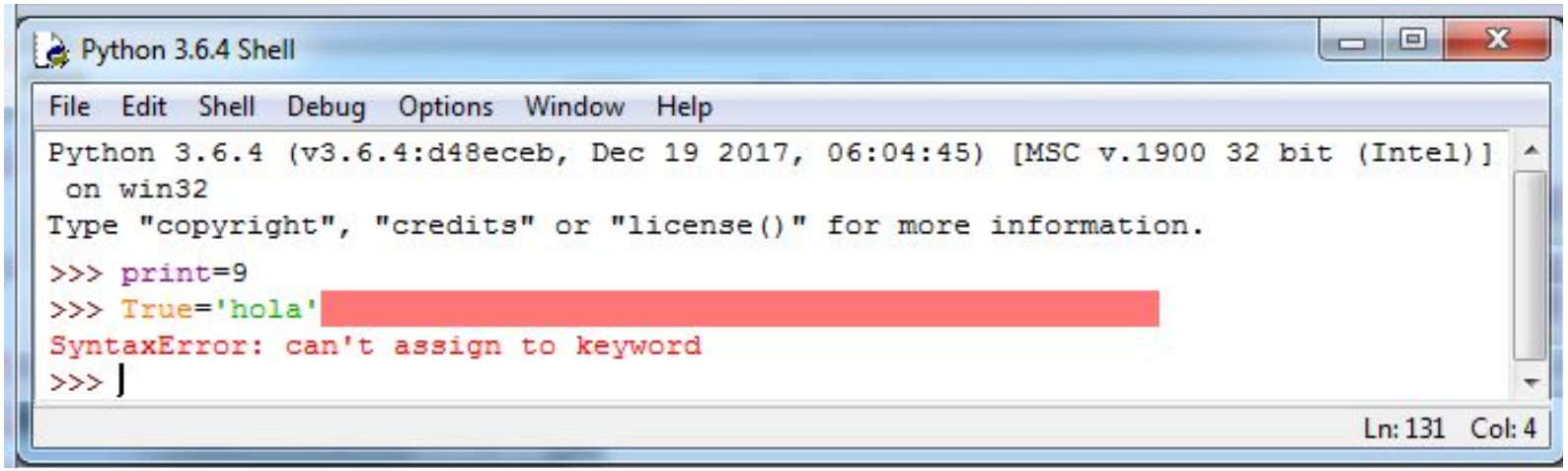
En la versión 3.x se quitaron de la lista **exec y print** han sido removidas, ya que ahora se presentan como funciones incorporadas por defecto.
Se han añadido: los términos nonlocal, True, False y None

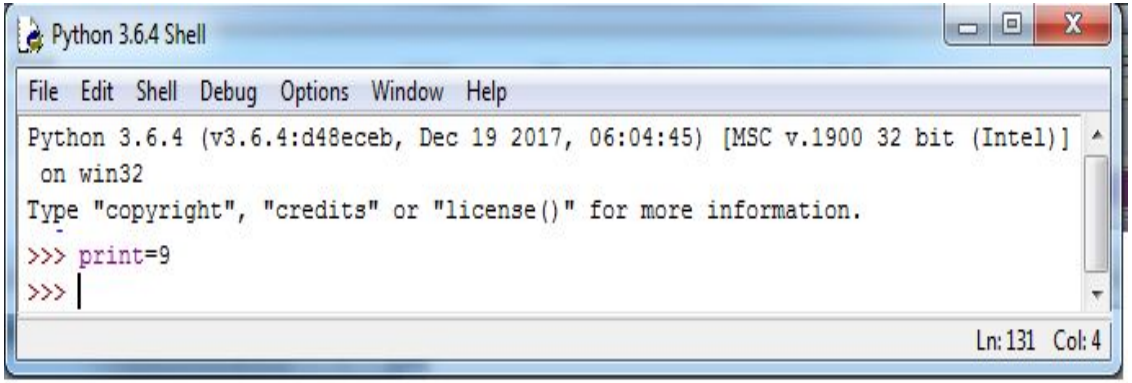
Por lo tanto, la lista de *keywords* en Python 3 resulta ser la siguiente.

```
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
```

SINTAXIS

Python: las palabras reservadas y sus versiones…





SINTAXIS

- **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 construcciones sintácticas compuestas de operadores y operandos, de cuya evaluación se obtiene un valor.
    - 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.

SINTAXIS

- **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 cómo formar a partir de esas palabras, las **“expresiones”** y **“sentencias”**

    - El If en C no lleva “”then””, en Pascal si

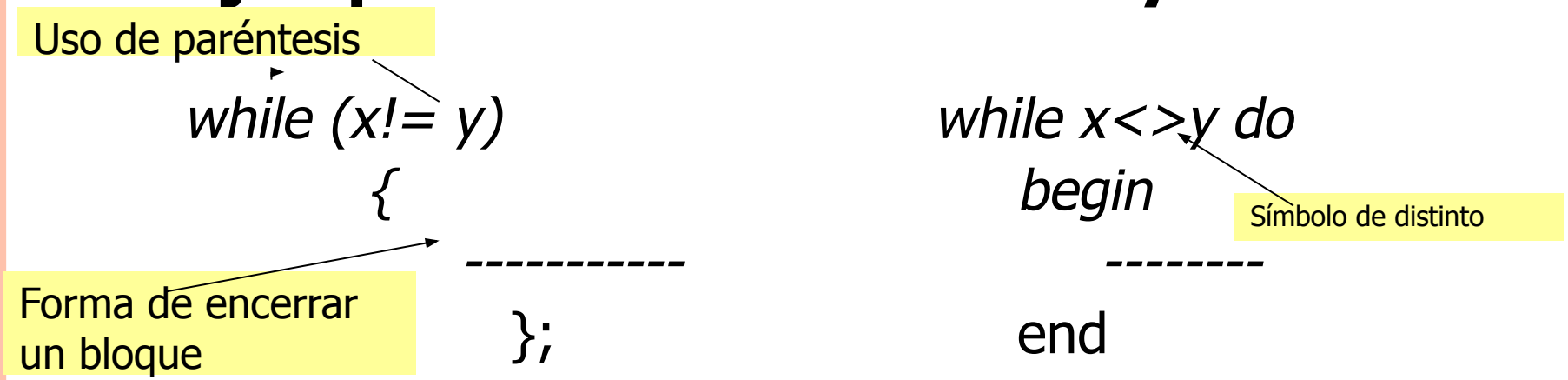
SINTAXIS

- **Tipos de Sintaxis**

  - **ABSTRACTA**
    - Se refiere básicamente a la estructura
  - **CONCRETA**
    - Se refiere básicamente a la parte léxica
  - **PRAGMÁTICA**
    - Se refiere básicamente al uso práctico

SINTAXIS

Ejemplo de sintaxis concreta y abstracta:



(En C) **(En Pascal)**

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

while condición
bloque

SINTAXIS

Ejemplo de sintaxis pragmática:

Ej1.

<> es más legible que **!=**

Ej2.

En C y Pascal {} o begin-end pueden omitirse si el bloque está 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 agregar el begin end o las {} (es decir que no define una única forma de escribirse).

SINTAXIS

- **Cómo definir la sintaxis**
  - Se necesitan métodos formales, una notación formal para describir la sintaxis de los lenguajes de programación, una descripción finita para definir un conjunto infinito de posibles programas bien escritos.
  - Ventajas:
    - Referencia para programadores: brinda un documento concreto y fiable.
    - Referencia para implementadores: minimiza distintas interpretaciones que generan distintas implementaciones.
    - Herramienta para prueba de programas: para testear y verificar programas.
    - Prueba de implementaciones: para analizar la correctitud de un programa.
    - Herramienta para implementaciones automáticas: estas herramientas permiten automatizar parte de la construcción de los traductores.
    - Diseño de un LP: para describir y visualizar más claramente un LDP.

SINTAXIS

- **Cómo definir la sintaxis**

  - Formas para definir la sintaxis:
    - Lenguaje natural. Ej.: Fortran
    - **Gramáticas libres de contexto**, BNF y EBNF. BNF fue definida por Backus y Naun. Ej: Algol
    - **Diagramas sintácticos** son equivalentes a BNF pero mucho más intuitivos.
    - **Árboles** de análisis sintáctico.

SINTAXIS

- **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:

< digito > ::= 0|1|2|3|4|5|6|7|8|9

No terminal

Metasímblo

Se define como **Terminales**

SINTAXIS

- **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

G = (N, T, S, P)

Conjunto de símbolos no terminales

Conjunto de símbolos terminales

Símbolo distinguido de la gramática que pertenece a N

Conjunto de producciones

SINTAXIS

- **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).

SINTAXIS

- **Árboles sintácticos**

Ej: Si tenemos una gramática G para "oración"

N={<oración>,<sujeto>,<predicado>,<artIndet>,<sustPropio>,<sustComun>,<verbo>}
T= {un, una, Juan, Maria, perro, manta, canción, tiene, compra, canta}
S= oración
P= {<oración> ::= <sujeto><predicado>,
<predicado>::= <verbo><objeto>| <verbo>,
<sujeto>::= <sustPropio> | <artIndet> <sustComun>
<sustPropio>::= Juan | Maria
<artIndet>::= un | una
<sustComun>::= manta | perro | canción
<verbo>::= tiene | compra | canta
<objeto>::= <artiIndet><sustComun> | <sustComun>}

El lenguaje generado por esta gramática contendrá todas las oraciones posibles: Juan tiene un perro, Maria compra una manta, Un perro tiene una manta, Juan canta una canción, etc.

SINTAXIS

- **Árboles sintácticos**

"Juan un canta manta"

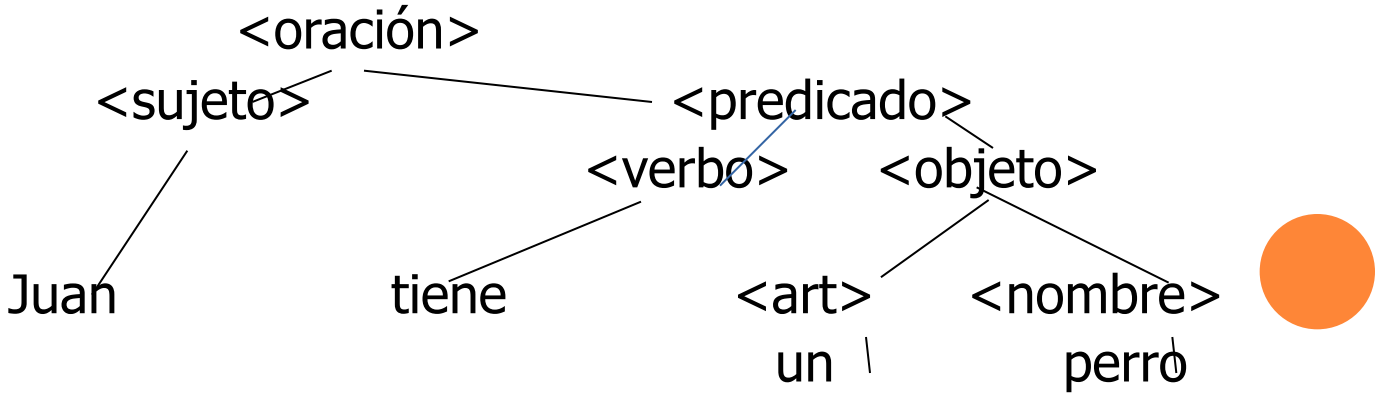
  - Es una oración sintácticamente incorrecta
  - No todas las oraciones que se pueden armar con los terminales son válidas
  - Se necesita de un **Método de análisis (reconocimiento)** que permita determinar si un string dado es válido o no en el lenguaje: **Parsing.**

El parse, para cada sentencia construye un "árbol sintáctico o árbol de derivación"

SINTAXIS

- **Árboles sintácticos**
  - Dos maneras de construirlo:
    - **Método botton-up**
      - De izquierda a derecha
      - De derecha a izquierda
    - **Método top-dow**
      - De izquierda a derecha
      - De derecha a izquierda

Ejemplo: árbol sintáctico de "oración". Top-down de izquierda a derecha sobre el ejemplo "Juan tiene un perro"



SINTAXIS

- **Árbol de derivación:**
  - Ejemplo top-down de izquierda a derecha

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

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

SINTAXIS

- **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

- **Ejemplo:** Expresiones simples de uno y dos términos

¿Cómo sería la gramática?

SINTAXIS

- **Ejemplo:** Expresiones simples de uno y dos términos

¿Cómo sería la gramática?

G=(N,T,S,P)
T={A,B,C,+,-,*,/}
N={<exp-simple>, }
S={<exp-simple>}
P={
<exp-simple>::= <id>|<id><operando><id>
<id>::=A|B|C
<operador>::=+|-|*|/
}

SINTAXIS

- **Producciones recursivas:**
  - Son las que hacen que el conjunto de sentencias descriptas sea infinito
  - Ejemplo de producciones recursivas:

<natural> ::= <digito> | <digito><digito> | ..... | <digito>.....<digito>

  - Si lo planteamos recursivamente

GN = (N, T, S, P)

N = { <natural>, <digito> } T = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

S = <natural>

P = {<natural> ::= <digito> | <digito> <natural>,

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

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

SINTAXIS

- **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 más de una forma

G= (N, T, S, P)

N = { <id>, <exp>, <asig>}

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

S = <asig>

P1 = {

<asig> ::= <id> := <exp>

<exp> ::= <exp>+<exp>|<exp>*<exp>|<exp>-<exp>|<exp>/ <exp>|<id>

<id> ::= A | B | C

}

Recursión a ambos lados. No recomendable.

SINTAXIS

- **Subgramáticas:**
  - Sea la gramática para identificadores GI = (N, T, S, P)

N = { <id>, <letra>, <digito>, <otro>}

T = { A, ……., Z, 0, ……., 1}

S = <id>

P = { <id> ::= <letra> | <letra><otro>,

<otro> ::= <letra> | <digito> | <letra><otro> | <digito><otro>,

<letra> ::= A | B | C | ……| Z

<digito> ::= 0 | 1 | 2 | | 9 }

  - 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"

SINTAXIS

- **EBNF: otra forma de describir la sintaxis libres de contexto**
  - Esta gramática es la BNF extendida
  - Los meta símbolos que incorporados son:

[ ] elemento optativo puede o no estar

(|) selección de una alternativa

{} repetición

*** 0 o más veces + 1 o más veces**

SINTAXIS

- **Ejemplos:**

  Definición números enteros en BNF y en EBNF

BNF

<enterosig> ::= + <entero> | - <entero> | <entero>

<entero> ::= <digito>| <digito><entero>

Recursión

EBNF

<enterosig>::= **[**(+|-)**]** <digito>{<digito>}*

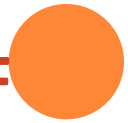
Repetición

Eliminó la recursión y es más fácil de entender

SINTAXIS

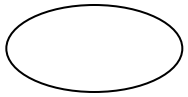
- **Diagramas sintácticos (CONWAY):**
  - Es un grafo sintáctico o carta sintáctica
  - Cada diagrama tiene una entrada y una salida, y el camino determina el análisis.
  - Cada diagrama representa una regla o producción
  - Para que una sentencia sea válida, debe haber una camino desde la entrada hasta la salida que la describa.

Se visualiza y entiende mejor que BNF o EBNF



SINTAXIS

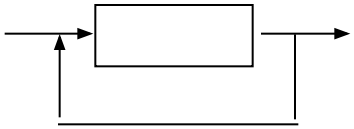
- **Diagramas sintácticos (CONWAY):**



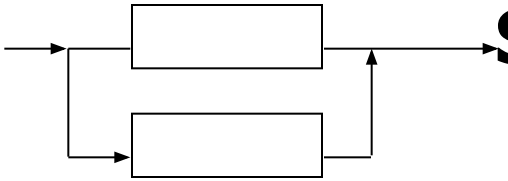
Terminales

Flujo

No terminales



Repetición



Selección

Ej:

SINTAXIS

- **Pensar:**

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