

CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

2024





**Antes de
empezar**

2024



Para tener en
cuenta

Atención a Estudiantes



Ante cualquier situación con estudiantes que observen o presencien y que requiera ser atendida, se disponen de los siguientes espacios a los que puede recurrir para consulta, asesoramiento u orientación:

● ACCESIBILIDAD

A.Paola Amadeo

accesibilidad@info.unlp.edu.ar
pamadeo@info.unlp.edu.ar

● ORIENTACIÓN AL ESTUDIANTE

Ana Ungaro

orientación.estudiantil@info.unlp.edu.ar
anaungaro@info.unlp.edu.ar

● GÉNERO

Sofía Martín

ddhhygenero@info.unlp.edu.ar
smartin@info.unlp.edu.ar

● ASESORAMIENTO PEDAGÓGICO

Anahí Almán

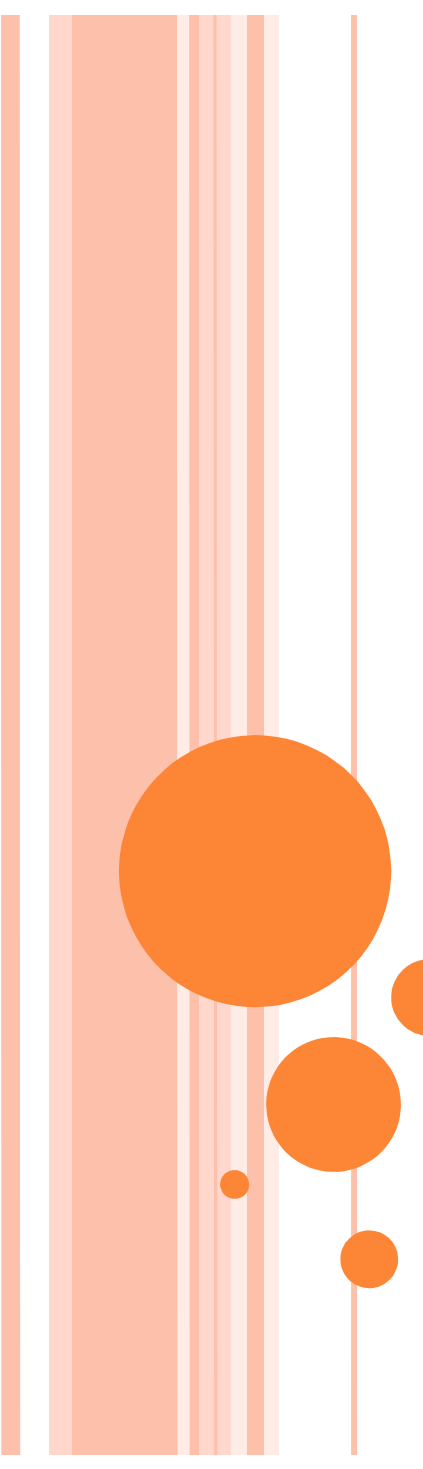
direccion.pedagogica@info.unlp.edu.ar



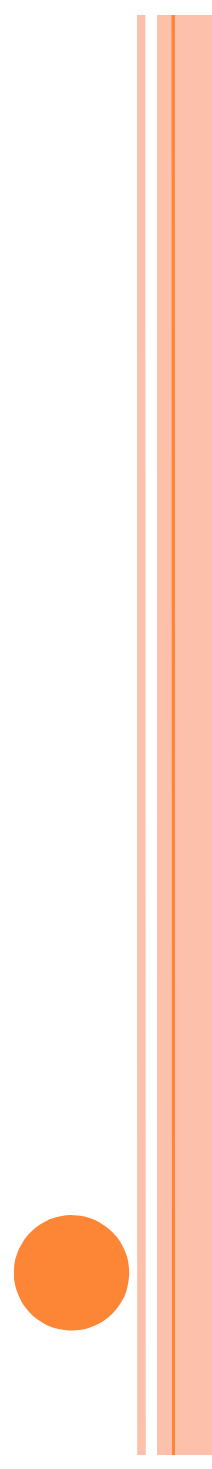
CONVOCATORIA COLABORADORES PARA PROYECTO “**EL BARRIO VA A LA UNIVERSIDAD**”

- *16 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.
- Como se puede colaborar:
 - Colaborando en las capacitaciones.
 - Colaborando con la producción del material





INTRODUCCION Y EVALUACION DE LENGUAJES



INTRODUCCIÓN

**Los lenguajes de Programación son
el corazón de la Ciencia de la
Infomática.**

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



CUAL 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”

- **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 PROGRAMACION

Para poder evaluar los lenguajes
necesitamos establecer criterios de
evaluación.



OBJETIVOS DE DISEÑO

- 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 excepciones,

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
- Ejemplo de cuestiones que atentan contra esto:
 - Muchas componentes elementales
 - Conocer subconjuntos de componentes
 - El mismo concepto semántico – distinta sintaxis
 - Distintos conceptos semánticos - la misma notación sintáctica
 - 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.
 - Abstracción de procesos y de datos



ORTOGONALIDAD

- 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 usuario comprende mejor si tiene un pequeño número de primitivas y un conjunto consistente de reglas de combinación



ORTOGONALIDAD



3. Funciones en C

3.5 Acceso a una función

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()
```

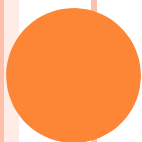
EFICIENCIA

- Tiempo y Espacio
- Esfuerzo humano
- Optimizable





SINTÁXIS



Sintáxis y Semántica

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



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

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**

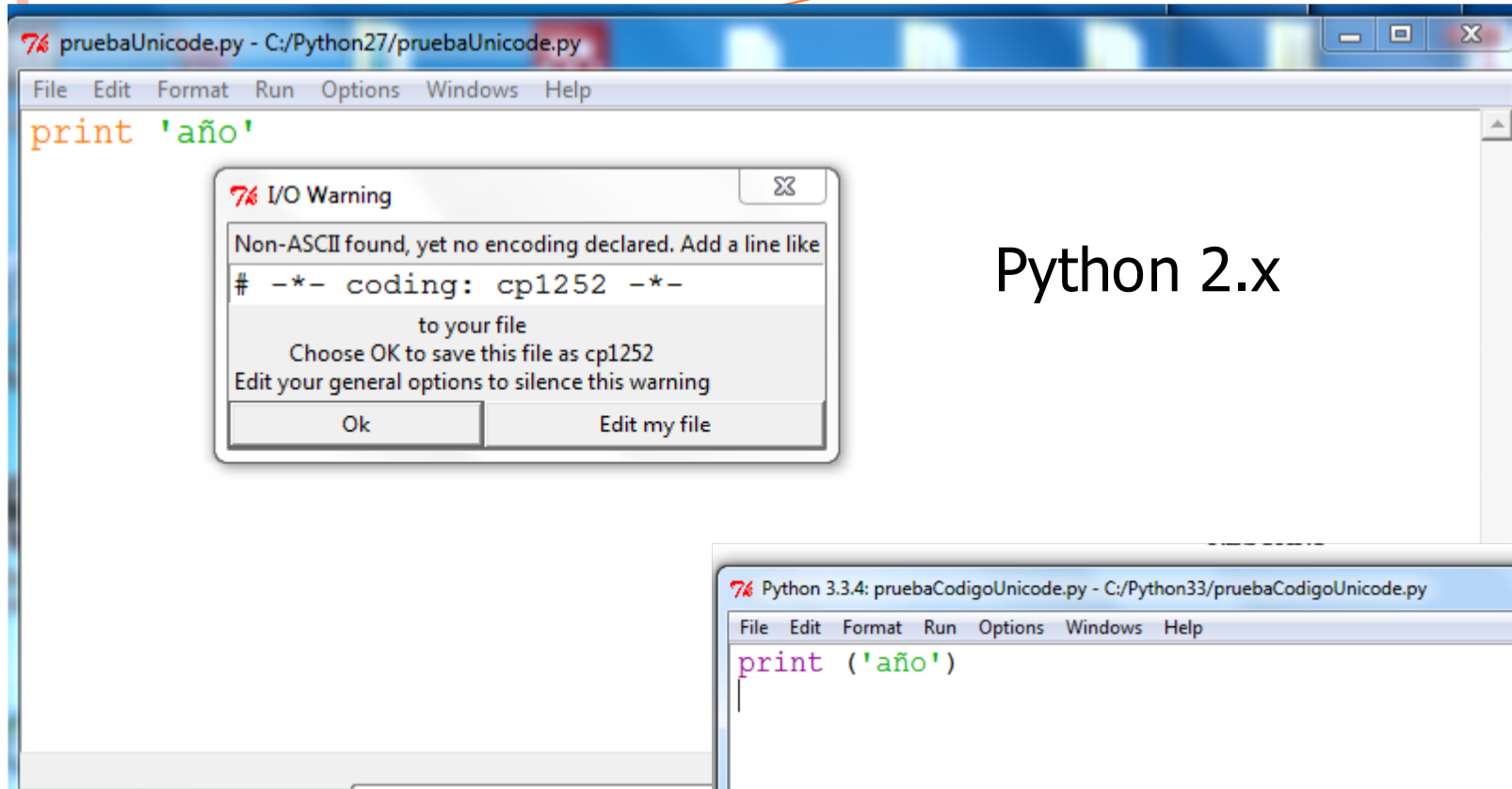
Importante: Tener en cuenta con qué conjunto de caracteres se trabaja sobre todo por **el orden** a la hora de comparaciones.

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

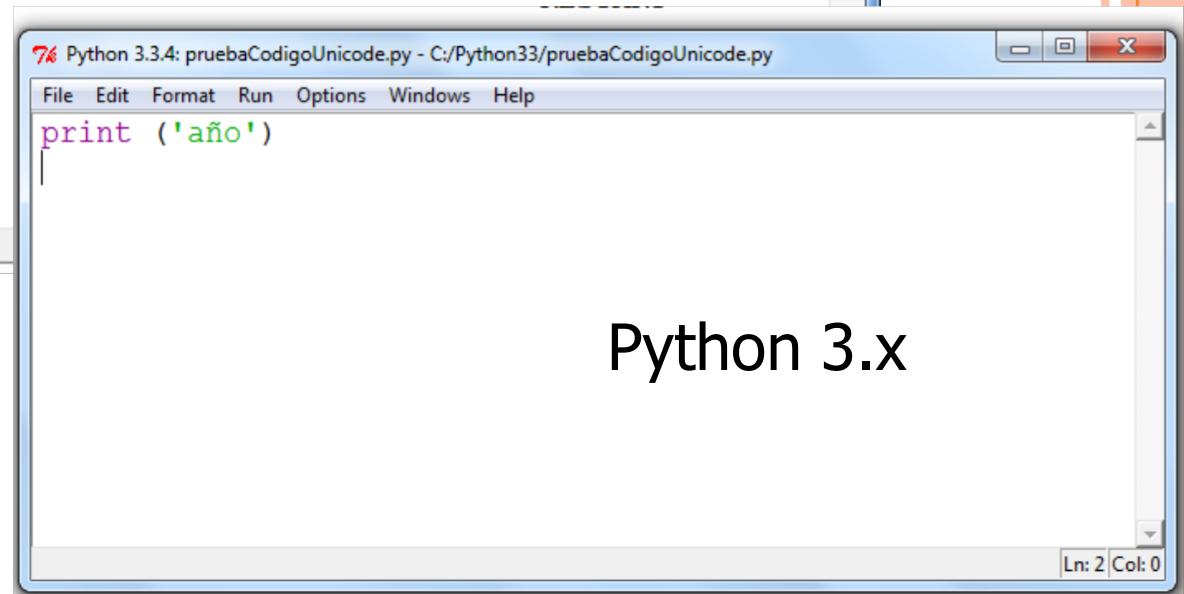


Sintáxis

-*- 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”.



Sintaxis

■ 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 f=1 then f=0;

Sintaxis

- **Palabra clave y palabra reservada**

Array do else if

Ejemplos de lenguajes con uso de palabras reservadas:

- **C**

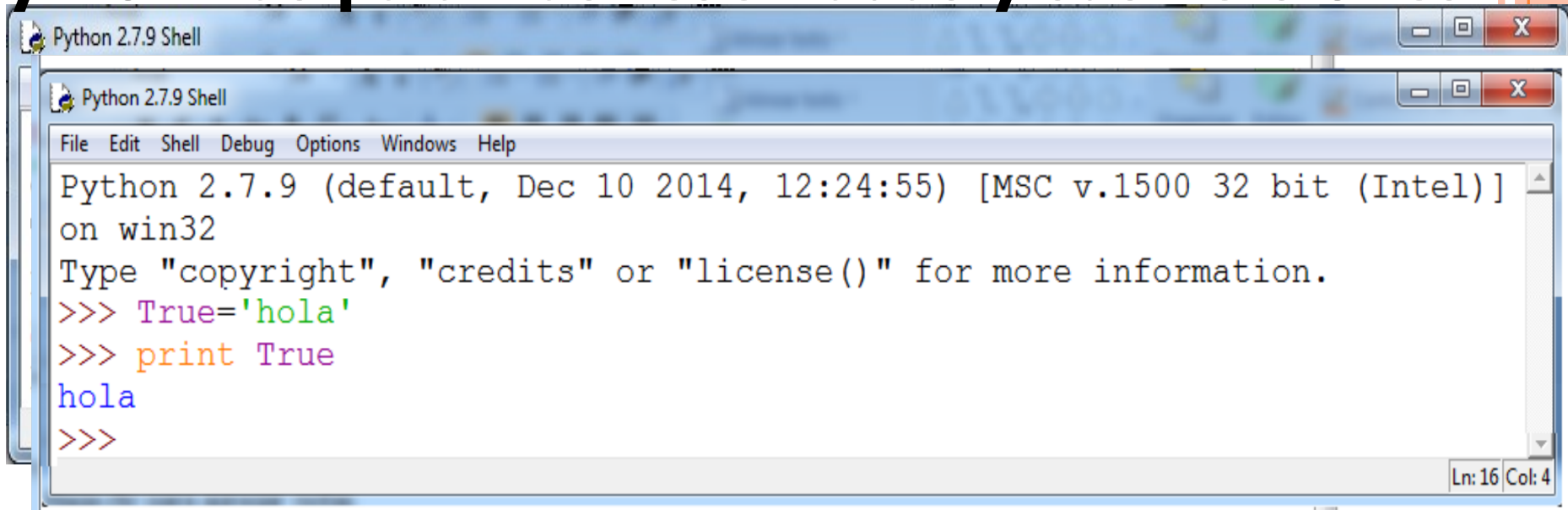
ej.: auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, etc.

- **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

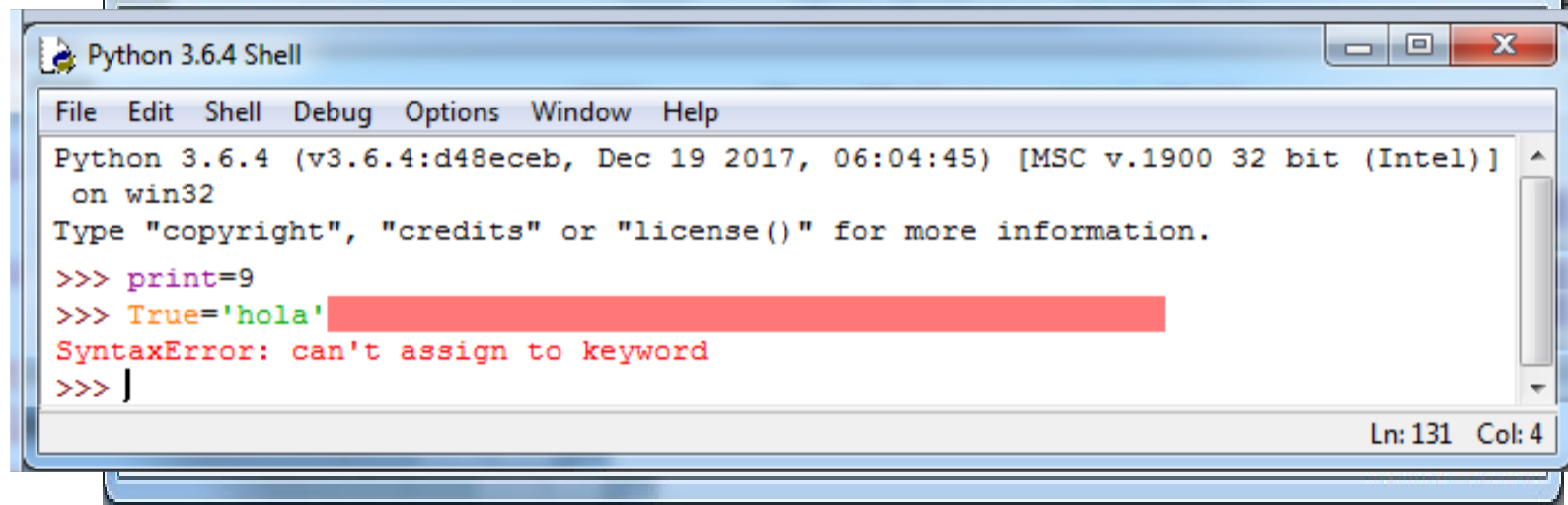
Python: las palabras reservadas y sus versiones...



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 information.
>>> True='hola'
>>> print True
hola
>>>
```

Ln: 16 Col: 4



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

Símbolo de distinto

end

(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 {}

(es decir que no define una única forma de escribirse)

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

↘
Metasímblo

Se define como **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”

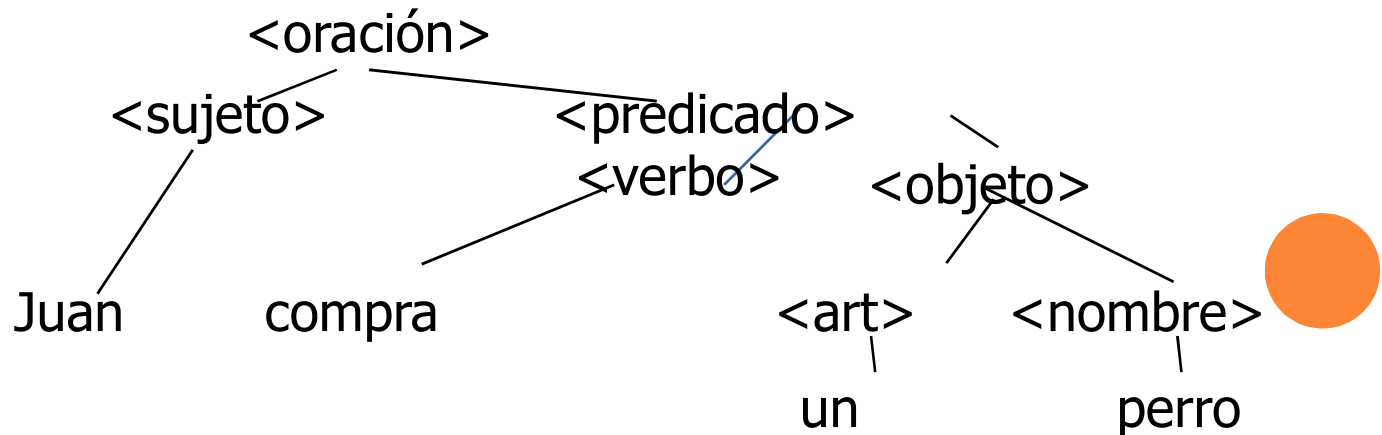
- Es una oración sintácticamente incorrecta
- No todas las oraciones que se pueden armar con los terminales son válidas
- Se necesita de un **Método de análisis (reconocimiento)** que permita determinar si un string dado es valido o no en el lenguaje: **Parsing**.
- El **parse**, para cada sentencia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

de **“Juan adopta un perro”**

<oración> => <sujeito><predicado>
=> Juan <predicado>
=> Juan <verbo><objeto>
=> Juan adopta <objeto>
=> Juan adopta <art><sustan>
=> Juan adopta un <sustan>
=> Juan adopta un perro

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



- Otro ejemplo:
 - Expresiones simples de uno y dos términos
 - Posibles operaciones: $+$ $/$ $*$ $-$
 - 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:

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

- 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

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

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

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

$S = \langle asig \rangle$

$P1 = \{$

$\langle asig \rangle ::= \langle id \rangle := \langle exp \rangle$

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

$\langle id \rangle ::= A \mid B \mid C$

$\}$

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 metasímbolos 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

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

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

Recursión

EBNF

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

Eliminó la recursión y es mas fácil de entender



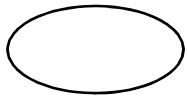
Sintáxis

- **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

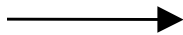


Sintaxis

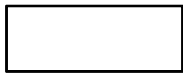
■ Diagramas sintácticos (CONWAY):



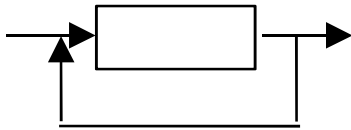
Terminales



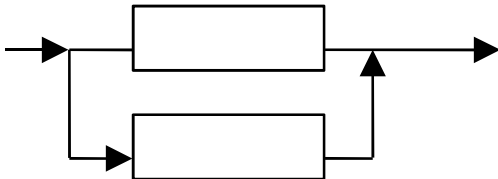
Flujo



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



Entonces...

Sintaxis	Es el conjunto de reglas que gobiernan la construcción o formación de sentencias válidas en un lenguaje	Forma: if <condición> then <sentencia>
Semántica	Es el conjunto de reglas que proporcionan el significado de una sentencia del lenguaje	Significado: Si el valor de <condición> es verdadero, se ejecutará <sentencia>
Gramática	Es la definición formal de la sintaxis de un lenguaje de programación. Cada lenguaje posee su propia gramática con su correspondiente sintaxis y semántica.	

