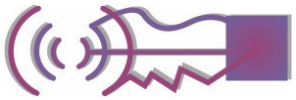
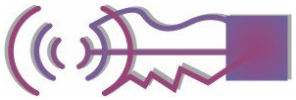


Sintáxis y Semántica

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

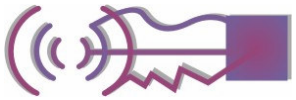
- Lenguaje de programación → Sintáxis
- Lenguaje de programación → Semántica



Sintáxis y Semántica

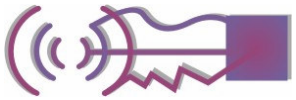
- **Definiciones.**

  - 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.



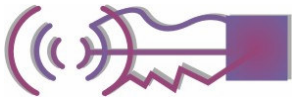
Sintáxis y Semántica

- ¿Cuál es la utilidad de definir la sintáxis y la semántica de un lenguaje? ¿Quiénes se benefician?
  - Un programa es válido?
  - Si lo es, qué significa?



Sintáxis

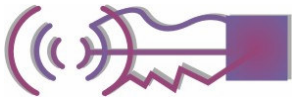
- **Características de la sintáxis**
  - ayudar al programador a escribir programas correctos sintácticamente
  - establece reglas que sirven para que el programador se comunique con el procesador f
  - Favorece:
    - Legibilidad
    - Verificabilidad
    - Traducción
    - Falta de ambigüedad



Sintáxis

La sintáxis establece reglas que definen cómo deben combinarse las componentes básicas, 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

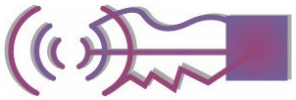
- **Alfabeto o conjunto de caracteres**
  - Fortran A-Z 0-9 b = + - / ( ) , . $; “ : 36 caracteres ANSI y 13 especiales (este es el mas usado)
  - Algol-60 A-Z a-z 0-9 < < > > = ¬..... 28 caracteres especiales
  - Antes: Tomar el alfabeto EBCDEIC 8 bits

    Corrientes

    Tomar el alfabeto ASCII 7 bits 93 caracteres imprimibles
  - Hoy: Se tiende a caracteres de 16 bits, ya que no alcanzan con 256 configuraciones, por la aparición de caracteres especiales, ej.: ~

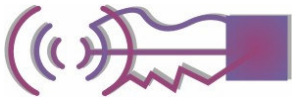
Importante: Tener en cuenta **el orden** de los caracteres que es lo que se utiliza en las comparaciones.

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



Sintáxis

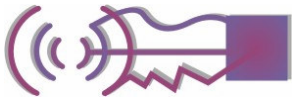
- **Identificadores**
  - Cadena de letras y dígitos, que deben comenzar con una letra
  - Si se restringe la longuitud se pierde legibilidad
- **Operadores**
  - Uniformidad de los aritmenticos
- **Comentarios**
  - Hacen los programas más legibles



Sintáxis

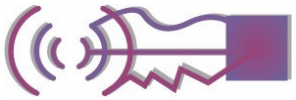
– Palabra clave y palabra reservada

- Palabra **clave** o **keywords:** tienen un significado dentro de un contexto.
- Palabra **reservada:** palabras claves que no pueden ser usadas por el programador como identificador de otra entidad.
- Ventajas de su uso:
  - expresividad
  - Legibibilidad
- desambiguar
  - Usar palabras reservadas
  - diferenciarlas
  - Libre uso y determinar de acuerdo al contexto.



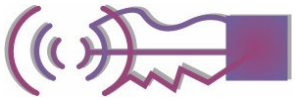
Sintáxis

- **Estructura sintáctica**
  - **Vocabulario o words**
    - Conjunto de caracteres y palabras para construir expresiones, sentencias y programas.
    - *Las words no son elementales se construyen a partir del alfabeto*
  - **Expresiones**
    - Son funciones que devuelven un resultado.
    - Son bloques sintácticos básicos
  - **Sentencias**
    - Componente sintáctico más importante.
    - Tiene un fuerte impacto en la facilidad de escritura y legibilidad
    - Hay sentencias simples, estructuradas y anidadas.



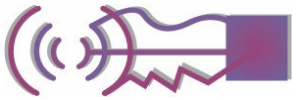
Sintáxis

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



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

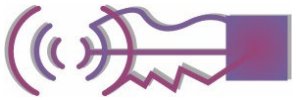
Ejemplo de sintáxis concreta y abstracta:.

While (x!=)	*While x<>y do*
{	*begin*
----------	*--------*
};	end
(En C)	**(En Pascal)**

- diferente **sintáxis concreta**
- igual **sintáxis abstracta**

w*hile condición*

bloque



Sintáxis

Ejemplo de sintáxis pragmática:.

Ej1.

<> es mas legible que !=

Ej2.

- **En C y Pascal {} o begin-end**
- ***while (x!=y) x=y+1***

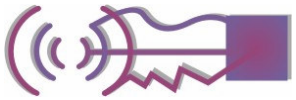
si se necesitara agregar una sentencia
begin end o las {}.

En Modula:

If x=y then

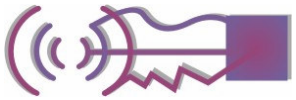
----

end



Sintáxis

- **Cómo definir la sintáxis**
  - Se necesita una descripción finita para definir un conjunto infinito 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

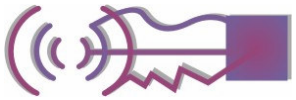
- **BNF (Backus Naun Form)**
  - Es una notación formal para describir la sintaxis
  - Es un metalenguaje
  - Utiliza metasímbolos
    - **< > ::= |**
  - Define las reglas por medio de “producciones”

Ejemplo:

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

No terminal Se define como **Terminales**

Metasímbolo



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

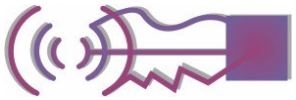
G = (N, T, S, P)

Conjunto de símbolos no terminales

Conjunto de símbolos terminales

Símbolo distinguido de la gramática que pertenece a N

Conjunto de producciones

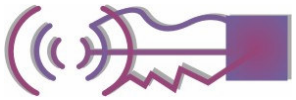


Sintáxis

- **Árboles sintácticos**

"Juan el un manta"

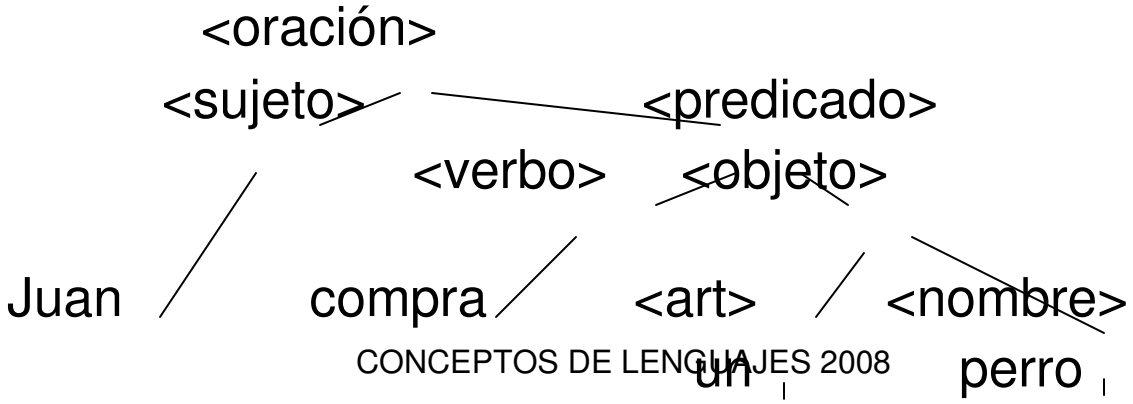
  - No todas las oraciones que se pueden armar con los terminales son válidas
  - Se necesita de un **Método de análisis (reconocimiento)**: **Parsing.**
  - El **parse,** para cada setencia construye un **"árbol sintáctico o árbol de derivación"**

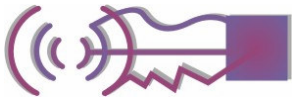


Sintáxis

- **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



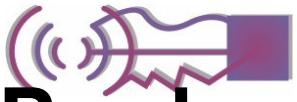


Sintáxis

- **Árbol de derivación:**
  - Ejemplotop-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



Sintáxis

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

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

  - Si lo planteamos recursivamente

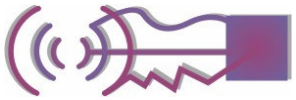
GN = (N, T, S, P)

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

S = <natural>

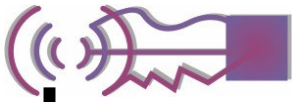
P = {<natural> ::= <digito> | <digito> <natural>,
<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

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



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



Sintáxis

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>,

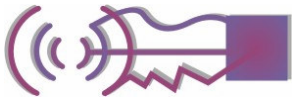
<otro> ::= <letra> | <digito> | <letra><otro> | <digito><otro>,

<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 utiliza las **subgramáticas** GN y GI



Sintáxis

- **Gramáticas ambiguas:**
  - una sentencia puede derivarse de mas de una forma

G= (N, T, S, P)

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

T = { A, ..., Z, +,*,:=}

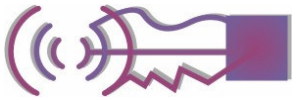
S = <asig>

P1 = {

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

<exp> ::= <exp>+ <exp>|<exp> * <exp>|<id> } **recursión**

<id> ::= A | B | C

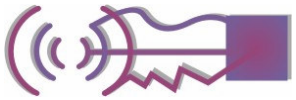


Sintáxis

- **Gramáticas libres de contexto y sensibles al contexto :**

int e; a := b + c;

  - son sentencias sintácticamente válidas, 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.



Sintáxis

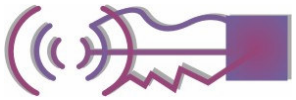
- **Otras formas de describir la sintaxis libres de contexto:**
  - **EBNF.** Esta gramática es la **BNF extendida**
  - Los metasimbolos que incorporados son:

[ ] elemento optativo puede o no estar

(|) selección de una alternativa

{} repetición

*** 0 o mas veces + una o mas veces**



Sintáxis

- **Ejemplo con EBNF:**

  Definición números enteros en BNF y en EBNF

  BNF

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

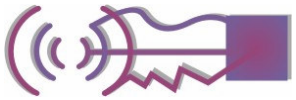
  <entero> ::= <digito>| <entero><digito>

  Recursión

  EBNF

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

Eliminó la recursión y es mas fácil de entender



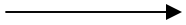
Sintáxis

- **Diagramas sintácticos (CONWAY):**
  - Es un grafo sintáctico o carta sintáctica
  - Cada diagrama tiene una entrada y una salida,
  - 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

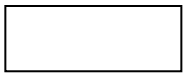
Sintáxis

- **Diagramas sintácticos (CONWAY):**

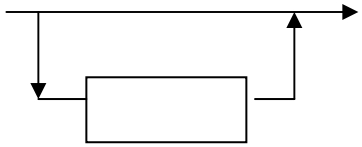
Terminales



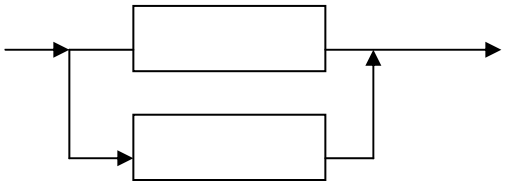
Flujo



No terminales



Repetición



Selección

Ej: