



Practica 2

[Siguiente](#)

[Anterior](#)

Objetivo: conocer como se define léxicamente un lenguaje de programación y cuales son las herramientas necesarias para hacerlo

- [Ejercicio 1 Complete el siguiente cuadro](#)
- [Ejercicio 2 ¿Cuál es la importancia de la sintaxis para un lenguaje?](#)
- [Ejercicio 3 ¿Explique a qué se denomina regla lexicográfica y regla sintáctica?](#)
- [Ejercicio 4 ¿En la definición de un lenguaje, a qué se llama palabra reservadas?](#)
- [Ejercicio 5 Dada la siguiente gramática escrita en BNF](#)
- [Ejercicio 6 Defina en BNF](#)
- [Ejercicio 7 Defina en EBNF la gramática para la definición de números reales](#)
- [Ejercicio 8 Utilizando la gramática que desarrolló en los puntos 6 y 7](#)
- [Ejercicio 9 Defina utilizando diagramas sintácticos la gramática para la definición](#)
- [Ejercicio 10](#)
- [Ejercicio 11 : La siguiente gramática intenta describir sintácticamente la sentencia for de ADA](#)
- [Ejercicio 12 Realice en EBNF la gramática para la definición un tag div en html 5](#)
- [Ejercicio 13 Defina en EBNF una gramática para la construcción de números primos](#)
- [Ejercicio 14 Sobre un lenguaje de su preferencia escriba en EBNF la gramática para la definición](#)

BNF Para la Practica



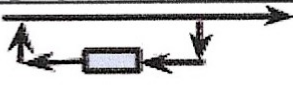
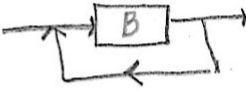
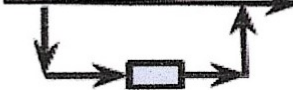
- [Pagina para testear](#)
- [Pagina para EBNF](#)

En la gramática de Backus-Naur Form (BNF) se utilizan diferentes símbolos para representar elementos gramaticales y construcciones sintácticas. Aquí te presento los símbolos más comunes utilizados en la notación BNF:

- `::=` se utiliza para indicar la definición de una regla de producción.
 - `|` se utiliza para separar opciones dentro de una misma regla de producción.
 - `< >` se utilizan para indicar no terminales (símbolos no léxicos o variables).
 - `" " o ' '` se utilizan para indicar literales o símbolos terminales (símbolos léxicos o constantes).
 - `[...]` se utiliza para indicar que un elemento es opcional.
 - `{...}` se utiliza para indicar que un elemento se puede repetir cero o más veces.
 - `(...)` se utiliza para agrupar elementos.
-

Ejercicio 1

Ejercicio 1: Complete el siguiente cuadro:

Meta símbolos utilizados por		Símbolo utilizado en Diagramas sintácticos	Significado
BNF	EBNF		
palabra terminal	palabra terminal		Definición de un elemento terminal
		rectángulo 	Definición de un elemento no terminal
::=	::=	diagrama con rectángulos, óvalos y flechas	<i>Definición de una Producción</i>
	()	flecha que se divide en dos o más caminos	<i>Selección de una alternativa</i>
< p > < p1 >	{ }	<i>Elemento que vuelve a la transición</i>	Repetición
	*		Repetición de 0 o más veces
	+		Repetición de 1 o más veces
	[]		<i>aproximadamente que esto presente o no esto</i>

Ejercicio 2

¿Cuál es la importancia de la sintaxis para un lenguaje? ¿Cuáles son sus elementos?

Conjunto de reglas que definen como componer letras, dígitos y otros caracteres para formar los programas

Por ejemplo

Ejemplo en pascal	<code>v: array [1..10] of integer;</code>
Ejemplo en C	<code>int v[10];</code>

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

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
- Palabra clave y palabra reservada
- Comentarios y uso de blancos

Detallados

- **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
- **Palabra clave y palabra reservada** 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

Ejercicio 3

¿Explique a qué se denomina regla lexicográfica y regla sintáctica?

Las reglas léxicas y sintácticas le dan la apariencia externa al lenguaje, establecen reglas para la estructura de la sintaxis. La estructura de la sintaxis se compone de:

- **Words** Vocabulario o words es el conjunto de palabras y caracteres necesarios para construir expresiones, sentencias y programas.
 - Por ejemplo: identificadores, operadores, palabras clave, etc. Las words no son elementales, se construyen a partir del alfabeto.
- **Expresiones** No son más que funciones a partir de un conjunto de datos que devuelven un valor, son bloques sintácticos.
- **Sentencias** Es el componente más importante. Tiene un fuerte impacto en la escritura y legibilidad. Hay sentencias simples, estructuradas y anidadas.
- Las reglas de la estructura son:
 - **Reglas léxicas** : Conjunto de reglas para formar las word, a partir de caracteres del alfabeto. Por ejemplo: diferencias entre mayúsculas y minúsculas, símbolos distintos, etc.
 - **Reglas sintácticas** : Conjunto de reglas que definen cómo formar las expresiones y sentencias. Por ejemplo: cuando se escriba la sentencia if se debe escribir la palabra if, luego la condición, luego el then, el begin, la sentencia, etc.

Ejercicio 4

¿En la definición de un lenguaje, a qué se llama palabra reservadas? ¿A qué son

equivalentes en la definición de una gramática? De un ejemplo de palabra reservada en el lenguaje que más conoce. (Ada,C,Ruby,Python,...)

En la definición de un lenguaje, se llama palabras reservadas a aquellas palabras que tienen un significado especial y están reservadas para su uso por parte del lenguaje en sí mismo. Estas palabras no pueden ser utilizadas como identificadores o nombres de variables en el código del programa.

En la definición de una gramática, las palabras reservadas son equivalentes a los símbolos no terminales o las producciones que representan las estructuras sintácticas especiales del lenguaje.

En el lenguaje de programación Python, algunas palabras reservadas incluyen:

- **if:** se utiliza para declarar una condición que debe cumplirse para que se ejecute un bloque de código.
- **else:** se utiliza para declarar un bloque de código que se ejecutará si la condición del if no se cumple.
- **while:** se utiliza para declarar un bucle que se ejecutará mientras se cumpla una condición.
- **for:** se utiliza para declarar un bucle que se ejecutará un número determinado de veces.
- **def:** se utiliza para declarar una función en Python.
- **class:** se utiliza para declarar una clase en Python.

Estas palabras reservadas tienen un significado especial en Python y no pueden ser utilizadas como nombres de variables o funciones en el código.

Ejercicio 5

Dada la siguiente gramática escrita en BNF:

```
G = ( N, T, S, P )
N = { <numero_entero>, <digito> }
T = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
S = <numero_entero>
P = {
    <numero_entero> ::= <digito> <numero_entero> | <numero_entero> <digito> | <digito>
    <digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
}
```

Compilado

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

- a) Identifique las componentes de la misma
 - **G** Gramática, son reglas finitas que definen un conjunto infinito de posibles sentencias válidas en el lenguaje. Está formada por una 4-tupla.
 - **N** conjunto de símbolos no terminales, que en este caso son <numero_entero> y <digito> .
 - **T** conjunto de símbolos terminales, que son los dígitos del 0 al 9.
 - **S** símbolo inicial, que es <numero_entero> .
 - **P** conjunto de producciones, que definen cómo se generan las cadenas válidas en el lenguaje. En este caso, hay dos producciones para <numero_entero> y una para <digito> . Las producciones indican que un <numero_entero> puede ser generado concatenando un <digito> y otro <numero_entero> , o bien concatenando un <numero_entero> y un <digito> , o bien siendo simplemente un <digito> . La producción para <digito> indica que un <digito> puede ser cualquiera de los dígitos del 0 al 9.
- b) Indique porqué es ambigua y corríjala

La gramática es ambigua porque una misma cadena puede ser generada por más de un árbol de derivación. Por ejemplo, la cadena "123" puede ser generada de dos formas diferentes:

- **<numero_entero>** →
 - <digito> <numero_entero> →
 - 1 <numero_entero> →
 - 1 <digito> <numero_entero> →
 - 1 2 <numero_entero> →
 - 1 2 3
- **<numero_entero>** →
 - <numero_entero> <digito>
 - → 12 <digito>
 - → 1 2 3

Para corregir la ambigüedad, se puede modificar la gramática de varias formas posibles. Aquí se presenta una opción:

```
G = (N, T, S, P)
N = {<numero_entero>, <digito>, <resto_numero>}
T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
S = <numero_entero>
P = {
    <numero_entero> ::= <digito> <resto_numero>
    <resto_numero> ::= <digito> <resto_numero> | " "
    <digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
}
```

Compilado

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

La principal diferencia es que se ha introducido un nuevo símbolo no terminal <resto_numero>, que se encargará de generar los dígitos restantes después del primer dígito. Además, se ha modificado la producción de <numero_entero> para que sea la concatenación de un <digito> y un <resto_numero>, y se ha añadido una nueva producción para <resto_numero> que indica que puede ser la concatenación de un <digito> y otro <resto_numero>, o bien ser la cadena vacía ε.

Con esta modificación, la gramática ya no es ambigua, ya que cada cadena generada solo tiene un árbol de derivación posible.

Ejercicio 6

Defina en BNF (Gramática de contexto libre desarrollada por Backus- Naur) la gramática para la definición de una palabra cualquiera.

```
G = (N, T, S, P)
N = {
    <palabra>,
    <caracter>
}
T = {0-9, a-Z}
S = <palabra>
P = {
    <palabra> ::= <letra> | <letra><palabra>
    <letra> ::= a | ... | z | A | ... | Z | " "
}
```

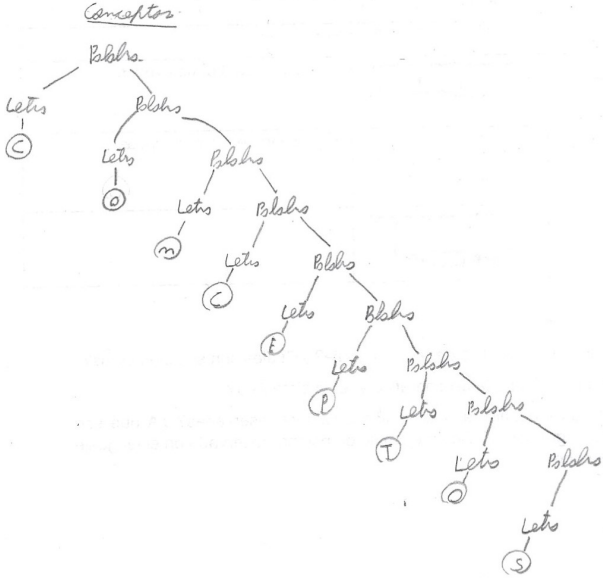
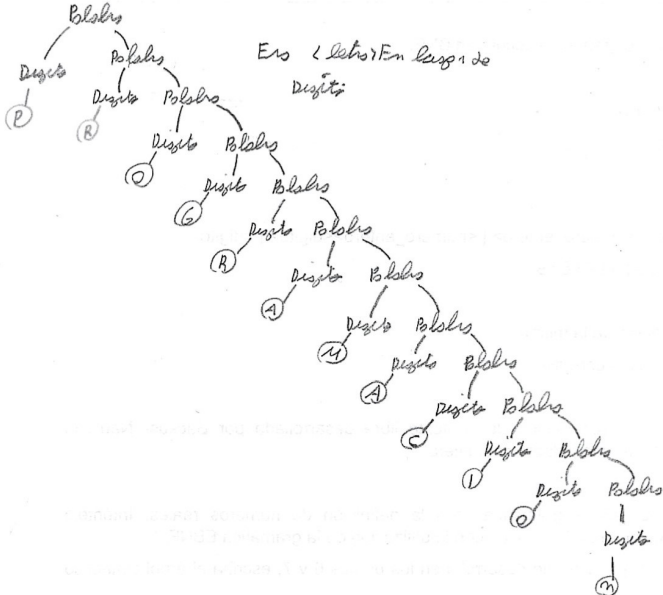
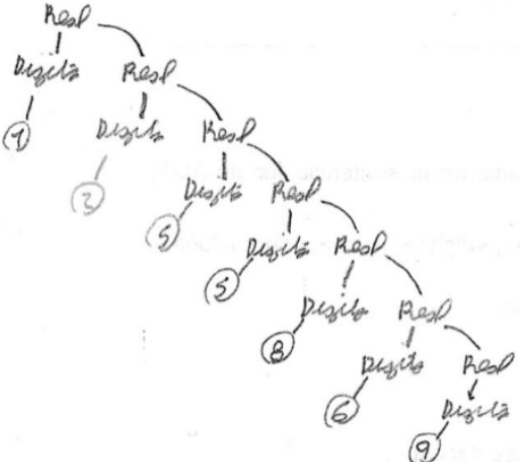
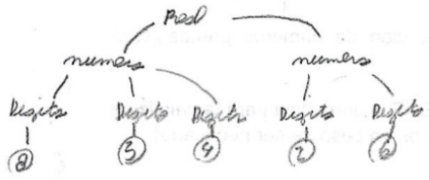
Ejercicio 7

Defina en EBNF la gramática para la definición de números reales. Inténtelo desarrollar para BNF y explique las diferencias con la utilización de la gramática EBNF.

EBNF	BNF
<pre>G = (N, T, S, P) N = {<real>, <numero>, <digito>} T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -} S = <real> P = { <real> ::= ["-"] <numero> ["." <numero>] <numero> ::= {<digito>}+ <digito> ::= 0 ... 9 }</pre>	<pre>G = (N, T, S, P) N = {<real>, <numero>, <digito>} T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., -} S = <real> P = { <real> ::= <numero> "." <numero> - <numero> "." <numero> <numero> ::= <digito> <digito><numero> <digito> ::= 0 ... 9 }</pre>

Ejercicio 8

Utilizando la gramática que desarrolló en los puntos 6 y 7, escriba el árbol sintáctico de:

Conceptos	Programación
<p><u>Conceptos</u></p> 	<p><u>Programación</u></p> <p>Eso < Letra > En lugar de Dígito</p> 
1255869	854,26
<p>!</p> <p><u>1255869</u></p> 	<p><u>854,26</u></p> <p>Una que es Ana</p> 

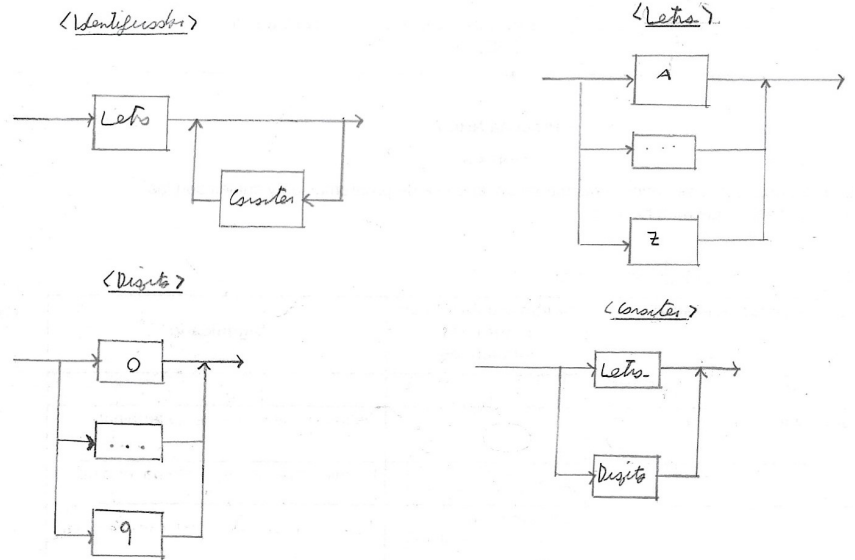
Ejercicio 9

Defina utilizando diagramas sintácticos la gramática para la definición de un identificador de un lenguaje de programación. Tenga presente como regla que un identificador no puede comenzar con números.

```

G = ( N, T, S, P)
N = { <real>, <letra> }
T = { 0-9, A-Z, a-z }
S = <real>
P = {
  <identificador> ::= <letra> { <caracter> } *
  <caracter> ::= ( <letra> | <digito> )
  <letra> ::= ( a | ... | z | A | ... | Z )
  <digito> ::= ( 0 | ... | 9 )
}

```



Ejercicio 10

a) Defina con EBNF la gramática para una expresión numérica, dónde intervienen variables y números. Considerar los operadores +, -, * y / sin orden de prioridad. No considerar el uso de paréntesis.

b) A la gramática definida en el ejercicio anterior agregarle prioridad de operadores.

```

G = ( N, T, S, P)
N = {
  <elemento>,
  <identificador>,
  <caracter>,
  <numero>,
  <digito>,
  <letra>,
}
T = { 0-9, a-Z, +, -, *, / }
S = <operacion>
P = {
  <expresion> ::= <elemento> { <termino> } +
  <termino> ::= { ( * | / | + | - ) <elemento> }

  <elemento> ::= ( <identificador> | <numero> )
  <identificador> ::= <letra> { <caracter> } *
  <caracter> ::= <digito> | <letra>
  <numero> ::= { <digito> } +
  <digito> ::= 0 | ... | 9
  <letra> ::= a | ... | z | A | ... | Z
}

```

```

G = ( N, T, S, P)
N = {
  <expresion>,
  <operacion>,
  <sumaResta>,
  <multiplicacionDivision>,
  <elemento>,
  <identificador>,
  <caracter>,
  <numero>,
  <digito>,
  <letra>
}
T = { 0-9, a-Z, "+", "-", "*", "/" }
S = <operacion>
P = {
  <expresion> ::= <operacion> { <sumaResta> } *
  <operacion> ::= <elemento> <multiplicacionDivision> | <elemento>
  <sumaResta> ::= ( "+" | "-" ) <operacion>
  <multiplicacionDivision> ::= ( "*" | "/" ) <elemento>

  <elemento> ::= ( <identificador> | <numero> )
  <identificador> ::= <letra> { <caracter> } *
  <caracter> ::= <digito> | <letra>
  <numero> ::= { <digito> } +
  <digito> ::= 0 | ... | 9
  <letra> ::= a | ... | z | A | ... | Z
}

```

c) Describa con sus palabras los pasos y decisiones que tomó para agregarle prioridad de operadores al ejercicio anterior. Lo primero que hacemos es determinar las multiplicaciones y divisiones del programa y de las sumas para el final.

Ejercicio 11

La siguiente gramática intenta describir sintácticamente la sentencia for de ADA, indique cuál/cuáles son los errores justificando la respuesta

```

N= {
  <sentencia_for>,
  <bloque>, <variable>,
  <letra>,
  <cadena>,
  <digito>,
  <otro>,
  <operacion>,
  <llamada_a_funcion>,
  <numero>,
  <sentencia>
}

P= {
  <sentencia_for> ::= for (i= IN 1..10) loop <bloque> end loop;
  <variable> ::= <letra> | <cadena>
  <cadena> ::= { ( <letra> | <digito> | <otro> ) }+
  <letra> ::= ( a | .. | z | A | .. | Z )
  <digito> ::= ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 )
  <bloque> ::= <sentencia> | <sentencia> <bloque> | <bloque> <sentencia> ;
  <sentencia> ::= <sentencia_asignacion> | <llamada_a_funcion> | <sentencia_if> |
  <sentencia_for> | <sentencia_while> | <sentencia_switch>
}

```

- <llamada_a_funcion> <sentencia_asignacion><llamada_a_funcion> no está definido.
- Mezcla recursión de EBNF y de BNF.
- <sentencia_if>, <sentencia_while>, <sentencia_switch> no está declarado.
- es ambiguo.
- Para preguntar en clase pero yo hice los siguiente y mas o menos funciona. Lo unico que no mire bien bien es la sentencia bloque

```

<sentencia_for> ::=
  "for " <identificador> " = " " IN " " 1.." <rango>
  " loop "
    <bloque>
  " end loop;"
<identificador> ::= [a-z] | [A-Z]
<rango> ::= [1-9]+
<bloque> ::= ([a-z] | [A-Z])*

```

- Machea for n = IN 1..6 loop as end loop;

Ejercicio 12

Realice en EBNF la gramática para la definición un tag div en html 5. (Puede ayudarse con el siguiente enlace (<https://developer.mozilla.org/es/docs/Web/HTML/Elemento/div>))

```

<div> ::=
  "<div>"
  <bloque>
  "</div>"
<bloque> ::= [0-9]*

```

Ejercicio 13

Defina en EBNF una gramática para la construcción de números primos. ¿Qué debería agregar a la gramática para completar el ejercicio?

No se puede porque tiene nros infinitos

Ejercicio 14

Sobre un lenguaje de su preferencia escriba en EBNF la gramática para la definición de funciones o métodos o procedimientos (considere los parámetros en caso de ser necesario)

```
G = (N, T, S, P)
N = {<funcion>}
T = {a-z, A-Z, 0-9}
S = <funcion>
P = {
    <div> ::= "funcion " <espacio> <identificador> <espacio> "(" <espacio> <parametros>? <espacio> ")" <espacio>
    <espacio> ::= "\s"*
    <letras> ::= ([a-z] | [A-Z])

    <tipoDato> ::= "int" | "real" | "string"
    <parametros> ::= <parametro> <parametros> | <parametro>
    <parametro> ::= <tipoDato> <espacio> <identificador> ","
    <identificador> ::= <letras> ( ([0-9] | <letras> )*)
}
```
