• Review Regular Expressions
• Overview Of Compiler Construction
• Dive Into Go Spec
  • EBNF
  • UTF8, LETTERS, DIGITS
• Lexical Elements
  • Comments
  • Semicolons
  • Identifiers
• Keywords, Operators,!
  • Punctuation
• Literals
<table>
<thead>
<tr>
<th>REG. EXP.</th>
<th>DENOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>${\epsilon}$</td>
</tr>
<tr>
<td>$a$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$ab$</td>
<td>${ab</td>
</tr>
<tr>
<td>$a\beta$</td>
<td>${s</td>
</tr>
<tr>
<td>$a^*$</td>
<td>${Euvu\alpha\nuuvu\ldots}$</td>
</tr>
<tr>
<td>$a^+$</td>
<td>$a^*\epsilon$</td>
</tr>
<tr>
<td>$a?$</td>
<td>$a</td>
</tr>
<tr>
<td>$[a\ldots bc]$</td>
<td>$a</td>
</tr>
<tr>
<td>$[^a\ldots bc]$</td>
<td>any $d \in \Sigma$ where $d$ is not in $[a\ldots bc]$</td>
</tr>
<tr>
<td>()</td>
<td>grouping</td>
</tr>
</tbody>
</table>
SEMANTIC ANALYSIS

CODE GENERATION

OPTIMIZATION

DECORATED PARSE TREE
But you can think in and use ASCII.

Source code is Unicode text encoded in UTF-8.
The syntax is specified using Extended Backus-Naur Form (EBNF):
Some identifiers are predeclared.

Follow the handy link. You’ll see that things like `int` and `printf` are predeclared.

A character in an identifier must be a letter. The first identifier is a sequence of one or more letters and digits. The first character in an identifier name program entities such as variables and types. An identifier name program entities such as variables and types.
In the Unicode Standard 8.0, Section 4.5, "General Category" defines a set of character categories. Co-reatras all characters in any of the letter categories Ll, Ll, Lm, Lr, L, R, or to an Unicode letter.

"Number, decimal digit" \* = / \* a Unicode code point classifed as
uncoode_digit

"Letter" \* = / \* a Unicode code point classified as
uncoode_letter

"Except newline" except newline uncoode_char

"The Unicode code point U+000A" \* = / \* the Unicode code point U+000A

The following terms are used to denote specific Unicode character

Characters
The underscore character _ (U+005F) is considered a letter.

**Letter**

<table>
<thead>
<tr>
<th>Hex Digit</th>
<th>Octal Digit</th>
<th>Decimal Digit</th>
<th>Unicode Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>&quot;u&quot;</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>7</td>
<td>&quot;f&quot;</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>15</td>
<td>&quot;f&quot;</td>
</tr>
</tbody>
</table>

...
A comment cannot start inside a rune or string literal, or inside a

```
/\[*\] *(\[*\] /\) ) \[*\] */
```

with the first subsequence character sequence */.

Regular expression metacharacters in green

```
\[\n/\n\]
```

alphabet in red

end of the line.

```
\n\n```

Line comments start with the character sequence /* and stop at the

Coments serve as program documentation. There are two forms:

Comments
(Important for their semicolon-elimination strategy.)

Any other comment acts like a newline.

... because C comments act like no characters.

which becomes

\[ z_{\text{real}} = a_{\text{real}} + b_{\text{real}} \]

then

\[ z_{\text{real}} = \text{real} \cdot \text{add}(a, b) \]

(there's an old C trick:

\[ \text{define } \text{real} \cdot \text{add}(x, y) \]

becomes a := foo + bar

\[ a := \text{foo} \cdot \text{hello} + \text{bar} \]

A general comment containing no newlines acts like a space.
1.3, and 4.2

To yield the three numbers

you don't want the program text

usual rule.

sequence of characters that form a valid token.

While breaking the input into tokens, the next token is the longest

also, a newline or end of the may trigger the insertion of a semicolon.

White space, isolated from spaces (U+0020), horizontal tabs (U+0009), carriage returns (U+000D), and newlines (U+000A), is ignored except

classes: identifiers, keywords, operators and punctuation, and literals.

Tokens form the vocabulary of the Go language. There are four
To refactor Grammatica use, code examples in this document will use semicolons using these rules.

1. When the input is broken into tokens, a semicolon is automatically inserted into the token stream immediately after a line's final token. When the input is broken into tokens, a semicolon is automatically inserted into the token stream immediately after a line's final token.

2. To allow complex statements to occupy a single line, a semicolon may be omitted before a closing "}" or ""]", or "}" or """

   - one of the operators and punctuation "+-`--', '{', '}', or ""

   - return

   - one of the keywords break, continue,=datetime, or

   - an integer, floating-point, imaginary, rune, or string literal

   - an identifier

The formal grammar uses semicolons as terminals in a number of productions. Go programs may omit most of these semicolons.
The following character sequences represent operators (including assignment operators) and punctuation:

Operators and punctuation:

- var
- return
- import
- for
- continue
- const
- fallthrough
- if
- else
- channel
- case
- break
- package
- switch
- struct
- map
- interface
- select

The following keywords are reserved and may not be used as identifiers.
Note that negative numbers are handled by the unary – operator.

\[ \text{\texttt{0x}} \quad \text{\texttt{0}} \quad 42 \]

- \{ \text{\texttt{0x}} \text{\texttt{digit}} \} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}}
- \{ \text{\texttt{0o}} \text{\texttt{digit}} \} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}}
- \{ \text{\texttt{0}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}}
- \text{\texttt{int}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}} \text{\texttt{digit}}

Represent values 10 through 15. 0x for hexadecimal. In hexadecimal literals, letters a–f and A–F constant. An optional prefix sets a non-decimal base: 0 for octal, 0x or Integer Literals
decimal point or the exponent may be elided. One of the integer part or the fractional part may be elided; one of the

The integer and fractional part comprise decimal digits; the exponent

an exponent part.

A floating-point literal is a decimal representation of a floating-point

Floating-point Literals
1.2345E+5  
1.25E  
1.6E1  
6.7428E-11T  
1.0E+0T  
2.718284  
0.5  
0.1  
01IIT == 11T  
01

imaginary 11T = (deciTmaT | float 11T) "f".  

integer followed by the lower-case letter 'I'. A complex constant. It consists of a floating-point literal or decimal literal followed by the lower-case letter 'I'. An imaginary literal is a decimal representation of the imaginary part of the imaginary literal.
A Rune Literal is expressed as one or more characters enclosed in single quotes, as in 'x' or 'n'.

A Rune Literal represents a Rune constant, an integer value identifying a Unicode code point.
This is a modern string constant definition, having both raw and interpreted styles.

In runtime literals, as in "bar" ... with backslash escapes, interpolated as they are.

Interpolated string literals are character sequences between double quotes, as in "foo" ... backslashes have no special meaning and the string may contain newlines. Can have new character (+) inside new string literals are discarded from the raw string value.

Raw string literals are character sequences between back quotes, as in 'foo' ... backslashes have no special meaning and the string may contain newlines. Can have new character (+) inside new string literals are discarded from the raw string value.

There are two forms: Raw string literals.

Interpolated string literals. A string literal represents a string constant obtained from concatenating a sequence of characters.

In runtime literals, as in "bar" ... with backslash escapes, interpolated as they are.

Interpolated string literals are character sequences between double quotes, as in "foo" ... backslashes have no special meaning and the string may contain newlines. Can have new character (+) inside new string literals are discarded from the raw string value.

Raw string literals are character sequences between back quotes, as in 'foo' ... backslashes have no special meaning and the string may contain newlines. Can have new character (+) inside new string literals are discarded from the raw string value.

There are two forms: Raw string literals.

Interpolated string literals. A string literal represents a string constant obtained from concatenating a sequence of characters.
Turing Machine

Linear Bounded ATM

Pushdown Automaton

Deterministic Finite Automaton

N -> N+1
N -> a
N -> b
M

DFA

N -> M
N -> b
M

PDA

N -> a
P
A

LBA

N -> P
A

TM

Decidable (R.E.)

Chomsky Hierarchy
For that nonterminal, with the R.H.S. of a production by replacing a nonterminal.

\[ S = E \]

\[ \{ \varepsilon, T, F \} \]

\[ N = \{ \varepsilon, T, F \} \]

\[ T = \{ \varepsilon, *, \varepsilon, \varepsilon \} \]

\[ d \]

\[ p \in D \]

\[ E \to T + T \]

\[ F \to \varepsilon \]

\[ F \to T \]

\[ F \to F * F \]

\[ F \to F + F \]

\[ F \to p \cdot \varepsilon \cdot d \]
CREATE a x = FORM a
starts with \( x_0 = \emptyset \)

**Derivation:**

\( x \) is a string on \( N^* \)

- \( N \)
- \( \Sigma \) - Productions
- \( S \) - Start symbol
- \( P \) - Set of terminals
- \( T \) - Set of non-terminals

\( \emptyset = N^T \)

- \( N^T \)
- \( \Sigma \) - Non-termininals
- \( \Sigma \) - Context-Free Grammar

- \( \Sigma \) - Lowercase terminals
- \( \Sigma \) - capital letters - non-terminals