I'm not robot

Backus naur form

Backus-Naur Form (BNF): A formal notation system used to describe programming languages. **Key Features**: BNF consists of a series of rules that define the structure and syntax of a language. Each rule is represented by a unique identifier, followed by an equals sign (=) and then an expression that defines the possible values for that identifier. **BNF Syntax**: The syntax of BNF is defined using a set of production rules, which are used to generate valid expressions in the language. These rules can be combined to form more complex expressions, allowing BNF to describe a wide range of programming languages. **Extended Backus-Naur Form (EBNF)**: A variation of BNF that adds additional features and syntax for describing languages. **Augmented Backus-Naur Form (ABNF)**: A superset of BNF that provides its own set of production rules and syntax. ABNF is widely used in the Internet community to define communication protocols, such as those used by IETF. **Purpose**: The primary purpose of BNF is to provide a formal way to describe the structure and syntax of programming languages, allowing for more precise and unambiguous communication among developers and users. I've tried to preserve the technical details while making the text more accessible to non-experts. The Backus-Naur Form (BNF) is a notation system used to describe the syntax of programming languages and other formal languages. Developed by John Backus and Peter Naur, BNF provides a precise way to define language structures, making it essential for official language specifications, manuals, and textbooks on programming language theory. Over time, various extensions and variants of BNF have emerged, including Extended Backus-Naur Form (EBNF) and Augmented Backus-Naur Form (ABNF). BNF consists of three components: non-terminal symbols, terminal symbols, and rules for replacing non-terminal symbols with sequences of symbols. These rules, known as derivation rules, are written in a specific format to indicate how symbols can be combined to form a syntactically correct sequence. By applying these rules iteratively, BNF definitions can generate longer and longer sequences of symbols. To illustrate this concept, consider a possible BNF definition for a U.S. postal address. This definition outlines the structure of an address, including name parts, street addresses, and zip code parts. Each part is further broken down into its constituent elements, demonstrating how BNF can be used to describe complex structures in a clear and concise manner. BNF notation describes opt-suffix-part as having suffix such as Sr. Jr. roman numeral or empty string. OPT-APT NUM consists of prefix Apt followed by apartment number or empty string. many things are left unspecified here and may be described using additional BNF rules. BNF idea can be traced back to Pāṇini, ancient Indian Sanskrit grammarian who lived between 6th and 4th century BC. Pāṇini notation is equivalent in power to Backus notation has many similar properties. Western society regarded grammar as teaching subject rather than scientific study. descriptions were informal targeted at practical usage. 20th century linguists Leonard Bloomfield Zellig Harris attempted formalize language description including phrase structure. string rewriting rules introduced and studied by mathematicians such as Axel Thue Emil Post Alan Turing. Noam Chomsky combined linguistics mathematics taking essentially Thue's formalism basis syntax natural language. Chomsky distinguished generative transformation rules context-free grammars 1956. John Backus proposed metalanguage metalinguistic formulas describe syntax programming language IAL known today ALGOL 58 1959. Backus notation first used in ALGOL 60 report. BNF notation described as Chomsky's context-free grammars. Backus may familiar with Chomsky work but doubts exist about this. Backus notation defines classes names enclosed angle brackets example denotes class basic symbols. Bnf developed further led to ALGOL 60 Peter Naur called Backus normal form. Donald Knuth argued BNF should read as Backus–Naur form conventional sense unlike Chomsky normal form The BNF metalanguage has distinct features that differentiate it from Chomsky context-free grammars. Unlike the latter, BNF does not require a rule to define its own formation; instead, this can be described naturally within brackets. The ALGOL 60 report showcases this difference with an example where variables are defined without a specific rule governing their structure. This example illustrates how natural language descriptions can suffice for specifying certain aspects of the language. BNF's use as a metalanguage is highlighted in the report, emphasizing its ability to define and describe other languages. Its simplicity and clarity make it a suitable choice for explaining the rules and syntax of programming languages like ALGOL. The concept of metalinguistic variables, also known as non-terminals, plays a crucial role in BNF's functionality. These variables do not need a specific rule to define them but can be described within the brackets. They serve as classes or categories for grouping symbols together, facilitating the definition and understanding of complex language constructs. The ALGOL 60 report and subsequent programming course materials demonstrate how BNF is used as a metalanguage to describe the syntax and semantics of ALGOL. This approach has been followed by other early ALGOL manuals, solidifying its position in the development and documentation of programming languages. BNF's recursive repeat construct was replaced by a sequence operator and target language symbols defined using quoted strings in META II, Schorre's metalanguage. The < and > brackets were removed, and parentheses () for mathematical grouping were added. This change enabled the definition and extension of metalanguages, but it came at the cost of natural language descriptions and metalinguistic variables. Many spin-off metalanguages, such as TREE-META and Metacompiler, were inspired by BNF. A BNF class describes a language construct formation as a pattern or action forming the pattern. The class name "expr" is described in natural language as a term followed by a sequence of addops and terms. A class is an abstraction that can be discussed independently of its formation. The natural-language supplement provided specific details on language semantics for compiler implementation and ALGOL program writing. It also supplemented syntax, such as the integer rule: ::= |. This rule allowed for spaces between digits, but in natural language, it was explained that digit sequences should have no whitespace. BNF's origin is less important than its impact on programming language development. During the ALGOL 60 report publication period, BNF served as the basis for many compiler-compiler systems. Some systems directly used BNF, while others, like Schorre Metacompilers, modified it into a programming language with minimal changes. In META II, classes became symbol identifiers, dropping the enclosing < and > brackets and using quoted strings for target language symbols. Arithmetic-like grouping was introduced to simplify rules, removing the need for classes where grouping was the only value. Output expressions were used in META II rules to output code and labels in assembly languages. The Unix utility yacc is based on BNF and uses similar code production mechanisms as META II. yacc is commonly used as a parser generator, and its roots are clearly rooted in BNF. Today, BNF remains one of the oldest computer-related languages still in use. The syntax of BNF (Backus-Naur Form) can be represented using itself, as shown in the example provided. This demonstrates how BNF can be used to describe its own structure. In this representation, a rule is defined by an optional whitespace followed by a literal string enclosed in double quotes or single quotes, preceded by an identifier and separated from other rules by whitespace. The expression part of the rule consists of two lists separated by a vertical bar, each containing one or more terms. Each term is a rule-name. The text also explains that BNF has several variants and extensions, such as regular expression repetition operators and square brackets for optional items. These extensions aim to simplify and adapt BNF to specific applications. Additionally, the extended Backus-Naur form (EBNF) is mentioned, which adds regular expressions to describe repeating patterns. The Augmented Backus-Naur form (ABNF) and Routing Backus-Naur form (RBNF) are also discussed as extensions commonly used in Internet Engineering Task Force protocols. Finally, the text highlights that parsing expression grammars build on BNF and regular expression notations, creating an alternative class of formal grammar that is analytical rather than generative. are intended to be human-readable and informal. This includes syntax rules and extensions like optional items in square brackets [], items that exist 0 or more times in curly brackets or suffixed with an asterisk (*), and items that exist 1 or more times suffixed with a plus symbol (+). Terminals appear in bold, non-terminals in plain text, and grouped items are enclosed in simple parentheses. ANTLR, Coco/R, DMS Software Reengineering Toolkit, GOLD, RPA BNF parser, XACT X4MR System, XPL Analyzer, bnfparser2, bnf2xml, JavaCC, tm, GNU bison, yacc, Racket's parser tools, and Qlik Sense use various forms of Backus-Naur Form (BNF) to define languages. These include ANTLR, Coco/R, DMS Software Reengineering Toolkit, GOLD, RPA BNF parser, XACT X4MR System, XPL Analyzer, bnfparser2, bnf2xml, JavaCC, tm, GNU bison, yacc, Racket's parser tools, and Qlik Sense. The Backus Normal Form (BNF) notation was first proposed by John W. Backus in 1959 as part of the ALGOL 60 programming language. However, Peter Zilahy Ingerman suggested renaming it to the Pāṇini-Backus Form in 1967 to give credit to the ancient Indian grammarian Pāṇini for his earlier work on syntax. Noam Chomsky's theories on language structure, published in the 1950s and 60s, had a significant influence on the development of BNF. Donald Knuth compared BNF with Backus-Naur Form (BNF) in 1964, but they are actually two distinct notations. The ALGOL 60 report, revised in 1963, adopted BNF as its notation for specifying the language's syntax. Ingerman suggested that the name be changed to Pāṇini-Backus Form, which was supported by some authors. Today, BNF is widely used as a metalanguage for defining the syntax of programming languages and other formal languages. It has been implemented in various tools and compilers, including JavaCC, RPatk, and BNF parser². The notation has also been used to define the syntax of natural language, such as SQL-92, SQL-99, and SQL:2003. Free access to BNF grammars for various programming languages like SQL, Ada, Java, C/C++, Pascal, COBOL, and PL/I. The "BNF Web Club" in Australia provides these resources. Additionally, the University of Geneva's DB research group offers freely available BNF/EBNF grammars for compiler construction. Another source, "Free Programming Language Grammars for Compiler Construction", provides grammars for various languages including C/C++, Pascal, COBOL, Ada 95, and PL/I.

Backus naur form tutorial. Backus naur form in ai. Backus naur form in hci. Backus naur form online. Backus naur form pronunciation. Backus naur form c. Backus naur form symbols. Backus naur form (bnf). Backus naur form grammar. Backus naur form a level. Backus naur form syntax. Backus naur form in toc. Backus naur form python. Backus naur form explained. Backus naur form syntax diagram.