Assignment 2

Lexical Conventions and Grammars of Languages: C vs. Python vs. Javascript

Sagar Jain
CS17BTECH11034

April 2, 2019

Contents

C: Lexical Conventions

C has just six types of tokens (smallest meaningful elements of a program) i.e. there exists a logical breakup of a correct C program into:

1. Identifiers
2. Keywords
3. Constants
4. String Literals
5. Operators
6. Seperators

Lexical conventions of the aforementioned tokens would essentially be the descrpition of the lexical conventions of the language.

Identifiers

An identifier is used to represent various programming elements such as variables, functions, arrays, structures, unions and so on.

1. The first character must be a letter(including underscore _).
2. Upper and lowercase letters are considered different.
3. Identifiers may have any length, and for internal identifiers, at least the first 31 characters are significant.

Keywords

Keywords are predefined, reserved words and may not be used as identifiers or otherwise. For example, auto, int, else, etc.

Constants

C has four types of constants:

1. **integer-constant**
   An integer constant is a sequence of digits, it is octal if it begins with a 0 and decimal otherwise, if it begins with an 0X or 0x it is hexadecimal with digits a to f or A to F. Integer constants can be suffixed with u or U to specify that is it unsigned and l or L to specify that it is long. The form, value and suffix of an integer constant decides its type.

2. **character-constant**
   A character constant is a sequence of one or more characters enclosed in single quotes as in 'x' . The value of a character constant with only one character is the numeric value of the character in the machine's character set at execution time. Escape sequences are used to represent characters like ' or new lines, they are of the form \ followed by a sequence on digits or letters. \ followed by 1 or 2 or 3 digits can also be used to represent the characters by their values in octal. While there is no limit on the number of digits after the \ , the behaviour is undefined if the value exceeds the largest character.

3. **floating-constant**
   A floating constant consists of an integer part(sequence of digits), a decimal part(sequence of digits), a fraction part, an e or E, an optionally signed integer exponent and an optional type suffix, one of f, F , l, or L. The integer part and the fractional part both cannot be missing. The decimal part or the e and exponent both cannot be missing. Suffix F or f makes it float, L or l makes it long double, otherwise it is double.

4. **enumeration-constant**
   Enumerations are unique types with values ranging over a set of named constants called enumerators. The form of an enumeration specifier borrows from that of structures and unions. Essentially, Identifiers declared as enumerators are constants of type int.

String Literals

A string literal is a sequence of characters surrounded by double quotes. String literals do not contain newline or double-quote characters in order to represent them, the same escape sequences as for character constants are used.As with character constants, string literals in an extended character set are written with a preceding L.

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C has the following types of operators.

1. Arithmetic Operators (+, - , etc)
2. Relational Operators (!=, ==, etc)
3. Logical Operators (! , &&, etc)
4. Bitwise Operators (&, <<, etc)
5. Assignment Operators (=, += , &=, etc)
6. Misc Operators (sizeof(), ?:, etc)

Seperators

Blanks, horizontal and vertical tabs, newlines, formfeeds and comments as described below (collectively, "white space") are ignored except as they separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

Comments

The characters /* introduce a comment, which terminates with the characters */. Comments do not nest, and they do not occur within a string or character literals.

Python: Lexical Conventions

The primary topics in the lexical conventions of python are:

1. Line Structure and Grouping
2. Reserved Words
3. Literals
4. Operators
5. Tokens
6. Source Code encoding

Line Structure and Indentation

1. Each statement in a program is terminated with a newline. Long statements can span multiple lines by using the line-continuation character (\). Any part of a program enclosed in parentheses (...), brackets [...], braces {...}, or triple quotes can span multiple lines without use of the line-continuation character because they denote the start and end of a definition.
2. Indentation is used to denote different blocks of code, such as the bodies of func- tions, conditionals, loops, and classes.The amount of indentation used for the first state- ment of a block is arbitrary, but the indentation of the entire block must be consistent.
   If the body of any of the aforementioned is short it can be expressed in the same line following the :, for an empty body one can use the **pass** statement.
3. Multiple statements on the same line can be seperated by using a semicolon(;).
4. The # character denotes a comment that extends to the end of the line. A # appearing inside a quoted string doesnt start a comment, however.
5. Blank line signals the end of input when typing a statement that spans multiple lines.

Identifiers and Reserved Words

The lexical conventions for identifiers and reserved words are as follows:

1. An identifier is a name used to identify variables, functions, classes, modules, and other objects. Identifiers can include letters, numbers, and the underscore character (_), but must always start with a nonnumeric character.
2. Letters include a-z and A-Z, special symbols like $, % are not allowed in identifiers.
3. Words such as if, else, for, etc. are reserved and cannot be used as identifier name, the entire list of the reserved words is provided in the manual.

Literals

Python has five built-in numeric types,

1. Booleans
   The identifiers True and False are interpreted as Boolean values with the integer values of 0 and 1, respectively.

2. Integers
   Numbers without decimal points are treated as decimal integers, they are considered as octal or hexadecimal if they are prefixed with 0 and 0x respectively.

3. Long integers
   Integers suffixed with a l or L are considered as long integers which can be arbitrarily long.

4. Floating-point numbers
   Numbers having a decimal point or an e followed by an exponential are interpreted as floating point numbers.

5. Complex numbers
   An integer or floating-point number with a trailing j or J, such as 12.34J, is interpreted as a complex number.

String Literals

Python currently supports two types of string literals:

1. 8-bit character data (ASCII)
   or ISO-Latin character set as well as representing raw binary data as a sequence of bytes. By default, 8-bit string literals are defined by enclosing text in single (’), double (”), or triple (’ ’ ’ or ”””) quotes.You must use the same type of quote to start and terminate a string.The backslash (\) character is used to escape special characters such as newlines.

2. Unicode (16-bit-wide character data)
   Unicode string literals are defined by preceding an ordinary string literal with a u or U. Arbitrary Unicode characters are defined using the \uXXXX escape sequence.This sequence can only appear inside a Unicode string literal and must always specify a four-digit hexadecimal value.

Optionally, you can precede a string with an r or R, such as in r ”...”. The u must come before r, if at all i.e. ur ”...”.
Values enclosed in square brackets [...], parentheses (...), and braces {...} denote lists, tuples, and dictionaries, respectively.

Operators, Delimiters, and Special Symbols

Python has a list of recognised operators, including +, -, *=, etc.
Certain tokens serve as delimiters for lisrts, tuples and dictionaries, example, (), [ ], etc.

Documentation Strings

If the first statement of a module, class, or function definition is a string, that string becomes a documentation string for the associated objec

Decorators

Any function or method may be preceded by a special symbol known as a decorator, the purpose of which is to modify the behavior of the definition that follows. Decorators are denoted with the @ symbol and must be placed on a separate line immediately before the corresponding function or method.

Source Code Encoding

-*- coding: UTF-8 -*-
When the special coding: comment is supplied, Unicode string literals may be specified directly in the specified encoding (using a Unicode-aware editor program for instance).

JavaScript: Lexical Conventions

We describe the lexical conventions of javascript piece by piece in the following subsections.

Whitespace

Whitespace can take the form of formatting characters or comments. It is used to seperate tokens which would be otherwise considered as a single token.JavaScript offers two forms of comments, block comments formed with /* */ and line-ending comments starting with //.

Names

Names are used for statements, variables, parameters, property names, operators, and labels. A name is a letter(A-Z, a-z) followed by an optional sequence of letters, numbers and underscores except some certain words called reserved words which have a special meaning.

Numbers

JavaScript has a single number type. Unlike most other programming languages, there is no separate integer type. A number in javascript can described as theh following:

- A number is an integer followed by an optional fraction and an optional exponent.
- An integer is defined as either the digit 0 or a non-zero digit followed by a sequence if digits which may include a 0.
- A fraction is defined as a decimal point followed a sequence of digits.
- An exponent is defined an e or E followed by an optional sign (+,-) and a sequence of digits.

The value **NaN** is a number value that is the result of an operation that cannot produce a normal result The value **Infinity** represents all values greater than the upper limit on numbers in javascript.

Strings

A string literal can be wrapped in single quotes or double quotes. It can contain zero or more characters. The \(backslash) is the escape character. JavaScript does not have a character type. To represent a character, make a string with just one character in it.

- Strings literals are bounded by double quotes or single quotes, inside the quotes the strings can contain any unicode character except a single quote or a double quote (for a string enclosed in a single quote or double quote respectively) and backslash, other that these the string can contain escapd characters as shown above.
- An escaped character is either a baskslash followed by a single character or a backslash followed by four hexadecimal numbers.

Statements & Expressions

The type of statements in javascript are *var statements, disruptive statements, while statements, for statement, if statement, etc.*

- A block of statements is a sequence of statements delimited by a semicolon, enclosed in a pair of curly braces.
- An expression can be the following:
  - A literal value example string, number, etc., a variable or a built-in value (true, false, null, undefined, NaN, or Infinity)
  - An invocation expression preceded by new.
  - A refinement expression preceded by delete.
  - An expression wrapped in parentheses.
  - An expression preceded by a prefix operator.
  - or an expression followed by:
    * An infix operator and another expression.
    * The ? ternary operator followed by another expression, then by :, and then by yet another expression.
    * An invocation.
    * A refinement.

Functions

A function is basically described as the string function followed by a an optional name, followed by the parameters and then the body. Where:

- Parameters is a list of comma seperated names (list length can be zero) bounded in parantheses.
- The function body is a sequence of statements bounded by curly braces.

Short Note on Transition Diagrams

The transition diagrams are a very intuitive way of defining the lexical conventions of a language, they are essentially regular expressions (context free grammars on some occasions) using which, we can parse the input and split it into tokens.

- The diagram is started from the left end and finishes at the right end.
- Transition diagrams which have a double / symbol at the beginning or end allow whitespace to be inserted at the beginning and end respectively.
- Literals are in ovals and rules or descriptions are in rectangles.

Transition Diagram for Whitespace

From the transition diagram if whitespace we can infer the following:

- Whitespace can be just a space.
- It can be just a tab.
- It can be just a line end.
- It can be a // followed by any sequence of characters and ending with a line end. (i.e. a comment)
- It can be a /* followed by any sequenece of characters and ending with a */(*/ can not occur anywhere else). (i.e. a multiline comment)

Transition Diagram for Names

From the transition diagram of names it is quite easy to see that a name is essentially a letter followed by an optional sequence of numbers and characters(including underscores).

Transition Diagram for Numbers

The transition diagram describes a number as the following:

- A number is an integer followed by an optional fraction and an optional exponent.
- An integer is defined as either the digit 0 or a non-zero digit followed by a sequence if digits which may include a 0.
- A fraction is defined as a decimal point followed a sequence of digits.
- An exponent is defined an e or E followed by an optional sign (+,-) and a sequence of digits.

Transition Diagram for Strings

The transition diagram describes a string as:

- A " or ' enclosing the following:
  - A sequence of any Unicode character except " and \ and control character and escaped characters.
  - Where escaped characters are described as a backslash followed by a single character from a given set of characters or a backslash followed by a four digit hexadecimal sequence.