Assignment 2 : Lexical Conventions and Grammars of Languages: C vs. Python vs. Javascript

Puneet Mangla (CS17BTECH11029)

1. **C :**
    a. *Tokens* : Identifiers, keywords, constants, string literals, operators, and other separators are 6 types of tokens. Blanks, horizontal and vertical tabs, newlines, formfeeds and comments are ignored except when they are used to separate tokens. If input stream is separated into tokens upto a character, the next token is the largest strings that constitutes a token.
    b. *Comments* : A comment starts with /* characters and end with */ characters. They do not nest and do not occur within a string or character literals.
    c. *Identifiers* : It is a sequence of letter and digits where first character must be a letter including underscore (_). Upper and lower case letters are different. Identifiers may have any length, and for internal identifiers, at least the first 31 characters are significant. Internal identifiers include preprocessor macro names and all other names that do not have external linkage. Identifiers with external linkage are more restricted: implementations may make as few as the first six characters significant, and may ignore case distinctions.
    d. *Keywords* : Many identifiers are reserved as keywords and may not be used otherwise. Eg : `auto, double, int, struct, break, else, long, switch, case` etc.
    e. *String Literal :* They are defined as a sequence of characters surrounded by double quotes. Whether identical string literals are distinct is implementation-defined, and the behavior of a program that attempts to alter a string literal is undefined. Adjacent string literals are concatenated and after it a null byte \0 is appended to the string so that programs that scan the string can find its end. In order to represent newline or double-quote, escape sequences as for character constants are available.
    f. *Constants* : Integer , character, floating, enumeration are four types of constants .
        - Integer constants : It consists of sequence of digits interpreted as octal if starts with 0 otherwise as decimal. Octal contains digits from 0-7. If `0x` or `0X` precedes, it is taken as a hexadecimal number which additionally include `a or A` through `f or F` with values 10

through 15. They may contains u or U for unsigned and l and L for long as suffixes. If an integer constant is suffixed by UL, it is unsigned long.

- Character constants : It is a sequence of more than one characters under single quotes. 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. Character constants do not contain the ' character or newlines; in order to represent them, and certain other characters, the escape sequences may be used . For example \n for newline, \t for horizontal tab, \\ for backslash, \ooo for octal numbers, \xhh for hexadecimal numbers etc. The escape \ooo consists of the backslash followed by 1, 2, or 3 octal digits, which are taken to specify the value of the desired character.
- Floating constants : consists of an integer part, a decimal part, 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 and fraction parts both consist of a sequence of digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing. Default is double.
- Enumeration constants : Identifiers declared as enumerators are constants of type int.

Python :

a. *Line Structure* : 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.

b. *Indentation* : classes.The amount of indentation used for the first statement of a block is arbitrary, but the indentation of the entire block must be consistent. If the body of a function, conditional, loop, or class is short and contains only a few statements, they can be placed on the same line. To denote an empty body or block, use the pass statement.To place more than one statement on a line, separate the statements with a semicolon (;). When tab characters are encountered, they’re converted into the number of spaces required to move to the next column that’s a multiple of 8 A line containing a single statement can also be terminated by a semicolon although this is unnecessary.

c. *Comments* : # character denotes a comment that extends to the end of the line. A # appearing inside a quoted string doesn't start a comment.
d. *Identifiers* : Same as C style but Identifiers starting or ending with underscores often have special meanings. For example, identifiers starting with a single underscore such as `_foo` are not imported by the `from module import *` statement. Identifiers with leading and trailing double underscores such as `__init__` are reserved for special methods, and identifiers with leading double underscores such as `__bar` are used to implement private class members.
e. *Keywords* : identifiers like and, elif, if, global, or etc. are reserved words.
f. *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 object.
g. *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. More than one decorator can be used, but each one must be on a separate line.
h. *String Literals* : Python currently supports two types of string literals: 8-bit character data (ASCII) and Unicode (16-bit-wide character data).
    - ASCII : 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, the backslash itself, quotes, and nonprinting characters. Unrecognized escape sequences are left in the string unmodified and include the leading backslash. Triple-quoted strings can span multiple lines and include unescape newlines and quotes.
    - Unicode : Unicode string literals are defined by preceding an ordinary string literal with a u or U, such as in u"hello". In Unicode, each character is internally represented by a 16-bit integer value.

i. *Constants :* Booleans, Integers, Long integers, Floating-point and Complex numbers are 5 type constants.
    - Booleans : The identifiers True and False are interpreted as Boolean values with the integer values of 0 and 1, respectively.
    - Integer : Same as C.
    - Long Integers : unlike integers, which are limited by machine precision, long integers can be of any length (up to the maximum

memory of the machine). Although the trailing L is used to denote long integers, it may be omitted. In this case, a large integer value will automatically be converted into a long integer if it exceeds the precision of the standard integer type. No U suffix in python.

- Floating : Same as C.
- Complex : numbers.An integer or floating-point number with a trailing j or J, such as 12.34J, is a complex number.You can create complex numbers with real and imaginary parts by adding a real number and an imaginary number, as in 1.2 + 12.34J.

JavaScript :

a. *Whitespaces and comments* : the space between two consecutive tokens cannot be removed, but the other spaces can be removed. Comments are same as C.
b. *Identifiers / Names* : A name is a letter optionally followed by one or more letters, digits, or underbars.
c. *Keywords* : Names like `abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double` etc. are reserved in language.
d. *Numbers* : has a single number type. Internally, it is represented as 64-bit floating point, the same as Java double. There is no separate integer type, so 1 and 1.0 are the same value. 100 and 1e2 are the same number. Negative numbers can be formed by using the – prefix operator. The value NaN is a number value that is the result of an operation that cannot produce a normal result. NaN is not equal to any value, including itself. The value Infinity represents all values greater than 1.79769313486231570e+308.
e. *Strings / Characters* : 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. All characters in JavaScript are 16 bits wide. JavaScript does not have a character type. To represent a character, make a string with just one character in it. Strings are immutable. Once it is made, a string can never be changed. But it is easy to make a new string by concatenating other strings together with the + operator.

2. *< > here represents tokens .*

a. Whitespaces includes : either tab, spaces, newlines or comments

- ***Regex :*** **(<space>|<tab>|<newline>|**

(<//>(<any character except newline>|<nothing>)*<newline>)|
(</*>((<nothing>|</>*|(</>* <*>*)|<*>*) <any char except * and
/>)*|(</>*<*/>))*

b. *Names :* should start with letter letter optionally followed by one or more letters, digits, or underbars.
   - ***Regex :*** **<letter>[<digit><letter><_>]***
c. *Number Literal :* integer followed by optional fractional part and followed by optional exponent part.
   - ***Regex :*** **<integer><fraction>?<exponent>?**
   - *<integer>* : <0>|(<any digit except 0> <digit>*)
   - *<fraction>* : <.> <digit>*
   - *<exponent>* : (<e>|<E>)(<+>|<->|<nothing>)(<digit><digit>*)
d. *String Literal* : sequence of valid characters surrounded by either both “ or both ‘. valid sequence contains any character other than corresponding quote and escape sequences.
   - ***Regex :*** **(<″><valid_char_1>*<″>)|(<’><valid_char_2>*<’>)**
   - *<valid_char_1>* : [<any unicode char except “ and \ and control sequence><escaped sequence>]
   - *<valid_char_2>* : [<any unicode char except ‘ and \ and control sequence><escaped sequence>]
   - *<escaped_sequence>* : <\>(<”> | <’> | <\> | </> | <backspace> | <form feed> | <new line> |<carriage return> | <tab> | (<u><4 hexadecimal digits>))