Literals

Rule Engine · https://2yetis.net/docs/literals/

Item	Description
Integer Literals	1 or more digits from 0 to 9, eg 42.
Float Literals	1 or more digits from 0 to 9, followed by a decimal point and then one or more digits from 0 to 9, optionally followed by f or F, eg 42.0 or 42.0f.
Long Literals	1 or more digits from 0 to 9 suffixed with l or L , eg 42l.
Double Literals	1 or more digits from 0 to 9, followed by a decimal point and then one or more digits from 0 to 9 suffixed with d or D , eg 42.0d. A special literal NaN can be used to denote Double.NaN constant
BigInteger Literals	1 or more digits from 0 to 9 suffixed with h or H (for Huge ala OGNL, "does not interfere with hexa-decimal digits"), eg 42h.
BigDecimal Literals	1 or more digits from 0 to 9, followed by a decimal point and then one or more digits from 0 to 9 suffixed with b or B) , eg 42.0b.
Natural literals - octal and hex support	Natural numbers (i.e. Integer, Long, BigInteger) can also be expressed as octal or hexadecimal using the same format as Java. i.e. prefix the number with 0 for octal, and prefix with 0x or 0X for hexadecimal. For example 010 or 0x10.
Real literals - exponent support	Real numbers (i.e. Float, Double, BigDecimal) can also be expressed using standard Java exponent notation. i.e. suffix the number with e or E followed by the sign + or - followed by one or more decimal digits. For example 42.0E-1D or 42.0E+3B.

Item	Description
String literals	Can start and end with either ' or " delimiters, e.g. "Hello world" and 'Hello world' are equivalent. The escape character is \ (backslash). Unicode characters can be used in string literals; Unicode escape sequences consist of: a backslash '\' a 'u' 4 hexadecimal digits ([0-9],[A-H],[a-h]). Such sequences represent the UTF-16 encoding of a Unicode character, for example, 'a' is equivalent to '\u0061'.
Multiline format literals	Start and end with ` delimiter - back-quote -, e.g. `Hello world` The escape character is \ (backslash); Unicode escape sequences can also be used. These format literals can span multiple lines and allow Unified JEXL expressions (JSTL like expressions) to be interpolated. If a variable user valued JEXLis present in the environment - whether as a local or global variable -, the format `Hello ${user}` will evaluate as Hello JEXL.
Regular expression (regex) literals	Start with ~/ and ends with / delimiters, e.g. ~/ABC.*/ The escape character is \ (backslash); it only escapes the string delimiter \ (slash)
Boolean literals	The literals true and false can be used, e.g. val1 == true
Null literal	The null value is represented as in java using the literal null, e.g. val1 == null
Array literal	A [followed by zero or more expressions separated by , and ending with], e.g. [1, 2, "three"] This syntax creates an Object[]. Empty array literal can be specified as [] with result of creating Object[] JEXL will attempt to strongly type the array; if all entries are of the same class or if all entries are Number instance, the array literal will be an MyClass[] in the former case, a Number[] in the latter case. Furthermore, if all entries in the array literal are of the same class and that class has an equivalent primitive type, the array returned will be a primitive array. e.g. [1, 2, 3] will be interpreted as int[].
List literal	A [followed by zero or more expressions separated by , and ending with ,...], e.g. [1, 2, "three",...] This syntax creates an ArrayList

Item	Description
Set literal	A { followed by zero or more expressions separated by , and ending with }, e.g. { "one" , 2, "more"} This syntax creates a HashSet
Map literal	A { followed by zero or more sets of key : value pairs separated by , and ending with }, e.g. { "one" : 1, "two" : 2, "three" : 3, "more": "many more" } This syntax creates a HashMap
Range literal	A value followed by .. and ending with other value, e.g. 1 .. 42 This syntax creates a 'range' object in the form of a java iterable which can be used in for statement, e.g. for (i : 1..42) a = a + b[i]