# Operators

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

| Operator | Description |
|---|---|
| Boolean and | The usual && operator can be used as well as the word and, e.g. cond1 and cond2 and cond1 && cond2 are equivalent. |
| Boolean or | The usual \|\| operator can be used as well as the word or, e.g. cond1 or cond2 and cond1 \|\| cond2 are equivalent. |
| Boolean not | The usual ! operator can be used as well as the word not, e.g. !cond1 and not cond1 are equivalent. |
| Bitwise and | The usual & operator is used, e.g. 33 & 4, 0010 0001 & 0000 0100 = 0 |
| Bitwise or | The usual \| operator is used, e.g. 33 \| 4, 0010 0001 \| 0000 0100 = 0010 0101 = 37. |
| Bitwise xor | The usual ^ operator is used, e.g. 33 ^ 4, 0010 0001 ^ 0000 0100 = 0010 0100 = 37. |
| Bitwise complement | The usual ~ operator is used, e.g. ~33, ~0010 0001 = 1101 1110 = -34. |
| Left-shift << | The left shift operator (<<) shifts the first operand the specified number of bits to the left. 1 << 2 = 4 |
| Right-shift >> | The right shift operator (>>) shifts the first operand the specified number of bits to the right. 4 >> 2 = 1 |
| Unsigned Right-shift >>> | zero-fill right shift) shifts the first operand the specified number of bits to the right. The sign bit becomes 0, so the result is always non-negative. |
| Ternary conditional ?: | The usual ternary conditional operator condition ? if_true : if_false operator can be used as well as the abbreviation value ?: if_false which returns the value if its evaluation is defined, non-null and non-false, e.g. val1 ? val1 : val2 and val1 ?: val2 are equivalent. NOTE: The condition will evaluate to false when it refers to an undefined variable or null for all JexlEngine flag combinations. This allows explicit syntactic leniency and treats the condition 'if undefined or null or false' the same way in all cases. |

| Operator | Description |
| --- | --- |
| Null coalescing operator ?? | The null coalescing operator returns the result of its first operand if it is defined and is not null.<br>When xandyare null or undefined, x ?? 'unknown or null x' evaluates as 'unknown or null x' y ?? "default" evaluates as "default".<br>When var x = 42 and var y = "forty-two",x??"other" evaluates as 42 and y??"other" evaluates as "forty-two".<br>NOTE: this operator does not behave like the ternary conditional since it does not coerce the first argument to a boolean to evaluate the condition. When var x = false and var y = 0,x??true evaluates as false and y??1 evaluates as 0. |
| Equality | The usual == operator can be used as well as the abbreviation eq. For example val1 == val2 and val1 eq val2 are equivalent.<br>null is only ever equal to null, that is if you compare null to any non-null value, the result is false.<br>Equality uses the java equals method |
| Inequality | The usual != operator can be used as well as the abbreviation ne. For example val1 != val2 and val1 ne val2 are equivalent. |
| Less Than | The usual < operator can be used as well as the abbreviation lt. For example val1 < val2 and val1 lt val2 are equivalent. |
| Less Than Or Equal To | The usual <= operator can be used as well as the abbreviation le. For example val1 <= val2 and val1 le val2 are equivalent. |
| Greater Than | The usual > operator can be used as well as the abbreviation gt. For example val1 > val2 and val1 gt val2 are equivalent. |
| Greater Than Or Equal To | The usual >= operator can be used as well as the abbreviation ge. For example val1 >= val2 and val1 ge val2 are equivalent. |
| In or Match=~ | The syntactically Perl inspired =~ operator can be used to check that a string matches a regular expression (expressed either a Java String or a java.util.regex.Pattern). For example "abcdef" =~ "abc.* returns true. It also checks whether any collection, set or map (on keys) contains a value or not; in that case, it behaves as an "in" operator. Note that arrays and user classes exposing a public 'contains' method will allow their instances to behave as right-hand side operands of this operator. "a" =~ ["a","b","c","d","e",f"] returns true. |

| Operator | Description |
| --- | --- |
| Not-In or Not-Match!~ | The syntactically Perl inspired !~ operator can be used to check that a string does not match a regular expression (expressed either a Java String or a java.util.regex.Pattern). For example "abcdef" !~ "abc.* returns false. It also checks whether any collection, set or map (on keys) does not contain a value; in that case, it behaves as "not in" operator. Note that arrays and user classes exposing a public 'contains' method will allow their instances to behave as right-hand side operands of this operator. "a" !~ ["a","b","c","d","e",f"] returns false. |
| Starts With=^ | The =^ operator is a short-hand for the 'startsWith' method. For example, "abcdef" =^ "abc" returns true. Note that through duck-typing, user classes exposing a public 'startsWith' method will allow their instances to behave as left-hand side operands of this operator. |
| Not Starts With!^ | This is the negation of the 'starts with' operator. a !^ "abc" is equivalent to !(a =^ "abc") |
| Ends With=$ | The =$ operator is a short-hand for the 'endsWith' method. For example, "abcdef" =$ "def" returns true. Note that through duck-typing, user classes exposing an 'endsWith' method will allow their instances to behave as left-hand side operands of this operator. |
| Not Ends With!$ | This is the negation of the 'ends with' operator. a !$ "abc" is equivalent to !(a =$ "abc") |
| Addition | The usual + operator is used. For example val1 + val2 |
| Subtraction | The usual - operator is used. For example val1 - val2 |
| Multiplication | The usual * operator is used. For example val1 * val2 |
| Division | The usual / operator is used, or one can use the div operator. For example val1 / val2 or val1 div val2 |
| Modulus (or remainder) | The % operator is used. An alternative is the mod operator. For example 5 mod 2 gives 1 and is equivalent to 5 % 2 |

| Operator | Description |
| --- | --- |
| Side-effect operators | Some operators exist in side-effect forms. Their default behavior is to execute the operator and assign the left-hand side with the result. For instance a += 2 is equivalent to a = a + 2 The list of operators is:<br>+=<br>-=<br>*=<br>/=<br>%=<br>&=<br>\|=<br>^=<br><<=<br>>>=<br>>>>= |
| Negate | The unary - operator is used. It changes the sign of its numeric argument. For example -12 or -(a * b) |
| Positivize | The unary + operator is used. It performs an integral promotion meaning that byte, short, char arguments will be promoted to integer as a result. For example +12 or +(a * b) |
| Empty | The unary empty operator behaves exactly as the corresponding function empty(). For example empty arg and empty(arg) are equivalent |
| Size | The unary size operator behaves exactly as the corresponding function size(). For example size [1,2,3] and size([1,2,3]) are equivalent |