PERL OPERATORS

What is an operator?

Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. Perl language supports many operator types but following is a list of important and most frequently used operators:

- Arithmetic Operators
- Equality Operators
- Logical Operators
- Assignment Operators
- Bitwise Operators
- Logical Operators
- Quote-like Operators
- Miscellaneous Operators

Lets have a look on all operators one by one.

Perl Arithmetic Operators:

Assume variable *aholds*10*andvariable*b holds 20 then:

[Show Example]

Operator	Description	Example
+	Addition - Adds values on either side of the operator	*a* +b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	*a* –b will give -10
*	Multiplication - Multiplies values on either side of the operator	*a* ∗b will give 200
/	Division - Divides left hand operand by right hand operand	*b*/a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	*b*a will give 0

**	Exponent - Performs exponential *power* calculation on operators	*a* ** b will give 10 to the power 20

Perl Equality Operators:

These are also called relational operators. Assume variable *aholds*10*andvariable*b holds 20 then, lets check following numeric equality operators:

[Show Example]

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	$*a* == $*b* is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	$*a*! = $*b* is true.
<=>	Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.	$*a* <=> $*b* returns -1.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	$*a* > $*b* is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	$*a* < $*b* is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	$*a* >= $*b* is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	$*a* <= $*b* is true.

Below is a list of equity operators. Assume variable *aholds* " *abc* " *andvariable* b holds "xyz" then, lets check following string equality operators:

[Show Example]

Operator	Description	Example
lt	Returns true if the left argument is stringwise less than	$*alt*$*b* is true.

	the right argument.	
gt	Returns true if the left argument is stringwise greater than the right argument.	$*agt*$*b* is false.
le	Returns true if the left argument is stringwise less than or equal to the right argument.	$*ale*$*b* is true.
ge	Returns true if the left argument is stringwise greater than or equal to the right argument.	$*age*$*b* is false.
eq	Returns true if the left argument is stringwise equal to the right argument.	$*aeq*$*b* is false.
ne	Returns true if the left argument is stringwise not equal to the right argument.	$*ane*$*b* is true.
cmp	Returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument.	$*acmp*$*b* is -1.

Perl Assignment Operators:

Assume variable *aholds*10*andvariable*b holds 20 then:

[Show Example]

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	*c* =a + *bwillassignevalueof* a + *bint*oc
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	*c*+ =a is equivalent to *c* =c + $a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	*c*− =a is equivalent to *c* =c - $a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	*c*∗ =a is equivalent to *c* =c * $a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	*c*/ =a is equivalent to *c* =c / $a

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	*c*a is equivalent to *c* =c % a
**=	Exponent AND assignment operator, Performs exponential *power* calculation on operators and assign value to the left operand	*c* * * =a is equivalent to *c* =c ** $a

Perl Bitwise Operators:

Bitwise operator works on bits and perform bit by bit operation. Assume if *a* = 60; *and* b = 13; Now in binary format they will be as follows:

$a = 0011 1100

$b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~$a = 1100 0011

There are following Bitwise operators supported by Perl language

[Show Example]

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	$a & $b will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	$*a*\|$*b* will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	$*a*$^$*b* will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	$*a* will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	$a << 2 will give 240 which is 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	$a >> 2 will give 15 which is 0000 1111

Perl Logical Operators:

There are following logical operators supported by Perl language. Assume variable *aholdstrueandvariable*b holds false then:

[Show Example]

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	$*a**and*$*b* is false.
&&	C-style Logical AND operator copies a bit to the result if it exists in both operands.	$a && $b is false.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	$*a**or*$*b* is true.
||	C-style Logical OR operator copies a bit if it exists in eather operand.	$*a*\|\|$*b* is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not$*a**and*$*b* is true.

Quote-like Operators:

There are following Quote-like operators supported by Perl language. In the following table, a {} represents any pair of delimiters you choose.

[Show Example]

Operator	Description	Example
q{ }	Encloses a string with-in single quotes	q{abcd} gives 'abcd'
qq{ }	Encloses a string with-in double quotes	qq{abcd} gives "abcd"
qx{ }	Encloses a string with-in invert quotes	qx{abcd} gives `abcd`

Miscellaneous Operators:

There are following miscellaneous operators supported by Perl language. Assume variable a holds 10 and variable b holds 20 then:

[Show Example]

Operator	Description	Example
.	Binary operator dot . concatenates two strings.	If a =" abc ", b="def" then a.b will give "abcdef"
x	The repetition operator x returns a string consisting of the left operand repeated the number of times specified by the right operand.	$'-'x3$ will give ---.
..	The range operator .. returns a list of values counting *upbyones* from the left value to the right value	2..5 will give 2, 3, 4, 5
++	Auto Increment operator increases integer value by one	$a++ will give 11
--	Auto Decrement operator decreases integer value by one	$a-- will give 9
->	The arrow operator is mostly used in dereferencing a method or variable from an object or a class name	$obj->a$ is an example to access variable a from object obj.

Perl Operators Precedence

The following table lists all operators from highest precedence to lowest.

[Show Example]

```
left    terms and list operators (leftward)
left    ->
nonassoc        ++ --
right   **
right   ! ~ \ and unary + and -
left    =~ !~
left    * / % x
left    + - .
left    << >>
nonassoc        named unary operators
nonassoc        < > <= >= lt gt le ge
nonassoc        == != <=> eq ne cmp ~~
left    &
left    | ^
left    &&
left    || //
nonassoc        ..  ...
```

```
right   ?:
right   = += -= *= etc.
left    , =>
nonassoc        list operators (rightward)
right   not
left    and
left    or xor
```