Perl Operators

Before trying to organize the Perl operators, let's speak a litle about what we mean by operator in a programming language.

The operators work with numbers and strings and manipulate data objects called operands. We found the operators in expressions which we need to evaluate.

On the other side, we use Perl operators in a certain context. We have two types of contexts: the scalar context which is invoked when Perl is expected to return a single value (like the addition of two numbers) and the list context when Perl is expected to return two or more values.

The great majority of the Perl operators work in scalar context, if not we will specify.

Also note that you must be aware of the Perl operators precedence, if you are not sure, it is better to use the brackets.

www.profmariamichael.com

1. NUMERIC PERL OPERATORS

1.1. ARITHMETIC PERL OPERATORS

In the table below you see the Perl operators used in arithmetic operations.

Symbol	Name	Definition

+	addition	It's a binary operator that returns the sum of two operands.
		
		Example:
		`$v1 = 12;`
		`$v2 = 5;`
		`$v = $v1+$v2;`
		`print "$v (expected 17)\n";`

-	subtraction	It's a binary operator that returns the difference of two operands.

		Example:
		
		`$v1 = 12;`
		`$v2 = 5;`
		`$v = $v1-$v2;`
		`print "$v (expected 7)\n";`

-	negation	I's a unary operator that performs arithmetic negation.
		
		Example:
		
		`$v1 = 12;`
		`$v2 = -$v1;`
		`print "$v2 (expected -12)\n";`

*	multiplication	It's a binary operator that multiplies two operands.
		
		Example:
		
		`$v1 = 12;`
		`$v2 = 5;`
		`$v = $v1 * $v2;`
		`print "$v (expected 60)\n";`

/	division	It's a binary operator that divides left value by right value.
		
		Example:
		
		`$v1 = 12;`
		`$v2 = 5;`
		`$v = $v1 / $v2;`
		`print "$v (expected 2.4)\n";`

**	exponentiation	It's a binary operator that raises the left value to the power of the right value.
		
		Example:
		
		`$v1 = 12;`
		`$v2 = 2;`
		`$v = $v1 ** $v2;`
		`print "$v (expected 144)\n";`

%	modulus	It's a binary operator that returns the remainder of dividing left value by right value. Example: `$v1 = 12;`

		```
		$v2 = 5;
		$v = $v1 % $v2;
		print "$v (expected 2)\n";
		```

++	autoincrement	If you place this unary operator before/after a variable, the variable will be incremented before/after returning the value.
		
		Example:
		```
		$v1 = 10;
		$v2 = ++$v1;
		print "$v1 $v2(expected 11 11)\n";
		$v1 = 10;
		$v2 = $v1++;
		print "$v1 $v2(expected 11 10)\n";
		```

--	autodecrement	If you place this unary operator before/after a variable, the variable will be decremented before/after returning the value.
		
		Example:
		```
		$v1 = 10;
		$v2 = --$v1;
		print "$v1 $v2(expected 9 9)\n";
		$v1 = 10;
		$v2 = $v1--;
		print "$v1 $v2(expected 9 10)\n";
		```

1.2. NUMERIC RELATIONAL PERL OPERATORS

The numeric relational Perl operators compare two numbers and determine the validity of a relationship.

Symbol	Name	Definition

<	less then	The "less then" operator indicates if the value of the left operand is less than the value of the right one.
		
		Example:
		```
		($v1, $v2) = (5, 7);
		if($v1 < $v2){
		  print "OK (expected)\n";
		}
		```

<=	less than or equal to	The "less than or equal to" operator indicates if the value of the left operand is less than or equal to the value of the right one. Example: `($v1, $v2) = (5, 5);`

		`if($v1 <= $v2){` `print "OK (expected)\n";` `}`
>	greater than	The "greater than" operator indicates if the value of the left operand is greater than the value of the right one. Example: `($v1, $v2) = (5, 7);` `if($v2 > $v1){` `print "OK (expected)\n";` `}`
>=	greater than or equal to	The "greater than or equal to" operator indicates if the value of the left operand is greater than or equal to the value of the right one. Example: `($v1, $v2) = (5, 5);` `if($v2 >= $v1){` `print "OK (expected)\n";` `}`
==	equal	The "equal" operator returns true if the left operand is equal to the right one. Example: `($v1, $v2) = (5, 5);` `if($v1 == $v2){` `print "OK (expected)\n";` `}`
!=	not equal to	The "not equal to" operator returns true if the left operand is not equal to the right one. Example: `($v1, $v2) = (5, 7);` `if($v1 != $v2){` `print "OK (expected)\n";` `}`
<=>	numeric comparison	This binary operator returns -1, 0, or 1 if the left operand is less than, equal to or greater than the right one. Example: `($v1, $v2) = (5, 7);` `$v = $v1 <=> $v2;` `print "$v\n"; # displays -1;`

1.3. NUMERIC LOGICAL PERL OPERATORS

The numeric logical Perl operators are generally derived from boolean algebra and they are mainly used to control program flow, finding them as part of an if, a while or some other control statement. See in the table below the logical numerical Perl operators.

Symbol	Name	Definition

!	negation	This unary operator evaluates an operand and return true if the operand has the false value (0) and false otherwise.
		
		Example:
		`$v1 = !25;`
		`$v2 = !0;`
		`print "v1=$v1,v2=$v2\n"; # displays v1=,v2=1`

not	not	It has the same meaning as the "!" operator, described above.

and, &&	and	The "and" operator returns the logical conjunction of two operands.
		
		Example:
		`($v1, $v2) = (5, 7);`
		`if($v1 == 5 && $v2 == 7) {`
		`  print "OK (expected)\n";`
		`}`

or, ||	or	The "or" operator returns the logical disjunction of two operands.
		
		Example:
		`($v1, $v2) = (5, 7);`
		`if($v1 == 5 || $v2 == 0) {`
		`  print "OK (expected)\n";`
		`}`

xor	exclusive or	The "exclusive or" operator returns the logical exclusive-or of two operands (the result is true if either but not both of the operands is true).
		
		Example:
		`($v1, $v2) = (5, 7);`
		`print ($v1==5 xor $v2==3); # displays 1`
		`print ($v1==5 xor $v2==7); # displays`

?	conditional operator	This ternary operator is like the symbolic if ... then ... else clause from the C language. It returns the second operand if the leftmost operand is true and the third operand otherwise.
		
		Example:
		`$v = (2 == 2) ? "Equal\n" : "Not equal\n";`
		`print $v; # displays Equal`

1.4. NUMERIC BITWISE PERL OPERATORS

Numeric bitwise Perl operators are similar to the logical operators, but they work on the binary representation of data. They are used to change individual bits in an operand. Please note that both operands associated with bitwise operators are integers.

Symbol	Name	Definition

<<	shift left	The "shift left" << operator is a binary operator that shifts the bits to the left. Its first operand specifies the integer value to be shifted meanwhile the second one specifies the number of position that the bits in the value will be shifted. The rightmost bits of the integer value will be assigned with 0 and the leftmost bits will be discarded.
		
		Example:
		`$v = 25 << 3;`
		`print "$v (expected 200)\n";`

>>	shift right	The "shift right" >> operator is a binary operator that shifts the bits to the right. Its first operand specifies the integer value to be shifted meanwhile the second one specifies the number of position that the bits in the value will be shifted. The leftmost bits of the integer value will be assigned with 0 and the rightmost bits will be discarded.
		
		Example:
		`$v = 32 >> 3;`
		`print "$v (expected 4)\n";`

&	and	The "and" operator sets a bit to 1 if both of the corresponding bits in its operands are 1, and 0 otherwise.
		
		Example:
		`$v = 32 & 16;`
		`print "$v (expected 0)\n";`

|	or	The "or" operator sets a bit to 0 if both of the corresponding bits in its operands are 0, and 1 otherwise.
		
		Example:
		`$v = 32 | 16;`
		`print "$v (expected 48)\n";`

^	exclusive or	The "exclusive or" operator sets a bit to 1 if the corresponding bits in its operands are different, and 0 otherwise.
		
		Example:
		`$v = 3 ^ 9;`
		`print "$v (expected 10)\n";`

~	not	The unary "not" operator inverts each bit in the operand, changing all the ones to zeros and zeros to ones.

		Example: `$v = ~1024;`

1.5. OTHER NUMERIC PERL OPERATORS

See in the table below other numeric Perl operators:

Symbol	Name	Definition

,	comma	In scalar context this binary operator evaluates its left argument, discards this value, then evaluates its right argument and returns that value. In a list context, it's just a separator and inserts both its arguments into the list.
		
		Example (in scalar context):
		`$v1 = 2; $v2 = 4; $v3 = $v1 == $v2;`
		`$v = ($v3, 5 == 5);`
		`print(" $v3(expected )\n");`
		`print(" $v(expected 1)\n");`

=>	comma	It has the same function like the comma operator described above.

..	Range operator	In scalar context, this operator returns false as long as its left operand is false. When the left operand becomes true, the range operator returns true until the right operator remains true, after which it becomes false again. In a list context, this operator will return an array with contiguous sequences of items, beginning with the left operand value and ending with the right operand value (the items can be characters or numbers).
		
		Example (in a list context):
		`print ('a'..'zz');`
		`print ('1'..'10');`

1.6. NUMERIC ASSIGNMENT PERL OPERATORS

Numeric assignment Perl operators perform some type of numeric operation and then assign the value to the existing variable.

Symbol	Name	Definition

=	assignment	This is the ordinary assignment operator. In a scalar context, it assigns the right operand's value to the left operand. In a list context, it assigns multiple values to the left array operand if the right operand is a list.
		
		Example:
		`$v = 15;`
		`print $v, "\n";   # displays 15`
		`@array = (10, 20, 30);`

		`print "@array\n"; # displays 10 20 30`

+=	addition	It adds the right operand's value to the left operand.
		
		Example:
		`$v = 10;`
		`$v += 15;`
		`print "$v (expected 25)\n";`

-=	subtraction	It subtracts the right operand from the left operand.
		`$v = 25;`
		`$v -= 15;`
		`print "$v (expected 10)\n";`

*=	multiplication	It multiplies the left operand's value by the right operand's value.
		
		Example:
		`$v = 10;`
		`$v *= 15;`
		`print "$v (expected 150)\n";`

/=	division	It divides the left operand's value by the right operand's value.
		
		Example:
		`$v = 150;`
		`$v /= 15;`
		`print "$v (expected 10)\n";`

**=	exponentiation	It raises the left operand's value to the power of the right operand's value.
		
		Example:
		`$v = 12;`
		`$v **= 2;`
		`print "$v (expected 144)\n";`

%=	modulus	It divides the left operand value by the right operand value and assigns the remainder to the left operand.
		
		Example:
		`$v = 12;`
		`$v %= 5;`
		`print "$v (expected 2)\n";`

&&=	logical and	It's a combination between the logical "&&" and the assignment operators.
		
		Example:
		`$v = 1;`
		`$v &&= 7 == 7;`
		`print "$v (expected 1)\n";`

||=	logical or	It's a combination between the logical "||" and the assignment operators.

		Example:
		`$v = 0;`
		`$v ||= 7 == 7;`
		`print "$v (expected 1)\n";`

<<=	bitwise shift left	It's a bitwise left shift assign.
		
		Example:
		`$v = 25;`
		`$v <<= 3;`
		`print "$v (expected 200)\n";`

>>=	bitwise shift right	It's a bitwise right shift assign.
		
		Example:
		`$v = 32;`
		`$v >>= 3;`
		`print "$v (expected 4)\n";`

&=	bitwise and	It's a bitwise AND assign.
		
		Example:
		`$v = 32;`
		`$v &= 16;`
		`print "$v (expected 0)\n";`

|=	bitwise or	It's a bitwise OR assign.
		
		Example:
		`$v = 32;`
		`$v |= 16;`
		`print "$v (expected 48)\n";`

^=	bitwise exclusive or	It's a bitwise XOR assign.
		
		Example:
		`$v = 3;`
		`$v ^= 9;`
		`print "$v (expected 10)\n";`

2. STRING PERL OPERATORS

2.1.STRING RELATIONAL PERL OPERATORS

The string relational Perl operators compare two strings and determine the validity of a relationship.

Symbol	Name	Definition
lt	less than	It returns true if the left operand is stringwise less

		then the right one.
		
		Example:
		`($v1, $v2) = ("abc", "abz");`
		`if($v1 lt $v2){`
		`  print ("$v1 is less than $v2\n");`
		`}`

le	less than or equal to	It returns true if the left operand is stringwise less then or equal to the right one.
		
		Example:
		`($v1, $v2) = ("abc", "abc");`
		`if($v1 le $v2){`
		`  print("$v1 is less than or equal to`
		`$v2");`
		`  print "\n";`
		`}`

gt	greater than	It returns true if the left operand is stringwise greater then the right one.
		
		Example:
		`($v1, $v2) = ("abc", "abz");`
		`if($v2 gt $v1){`
		`  print ("$v2 is greater than $v1\n");`
		`}`

ge	greater than or equal to	It returns true if the left operand is stringwise greater then or equal to the right one.
		
		Example:
		`($v1, $v2) = ("abc", "abc");`
		`if($v1 ge $v2){`
		`  print("$v1 is greater than or equal to`
		`$v2");`
		`  print "\n";`
		`}`

eq	equality	It returns true if the left operand is stringwise equal to the right one.
		
		Example:
		`($v1, $v2) = ("abc", "abc");`
		`if($v1 eq $v2){`
		`  print ("$v1 is equal to $v2\n");`
		`}`

ne	not equal to	It returns true if the left operand is stringwise not equal to the right one.
		
		Example:
		`($v1, $v2) = ("abc", "ab1");`
		`if($v1 ne $v2){`
		`  print ("$v1 is not equal to $v2\n");`
		`}`

cmp	comparison	It returns -1, 0, or 1 if the left operand is stringwise less than, equal to or greater than the right one.
		
		Example:
		`($v1, $v2) = ("am", "ak");`
		`$v = $v1 cmp $v2;`
		`print ("$v(expected 1)\n");`

2.2. STRING LOGICAL PERL OPERATORS

The string logical Perl operators are generally derived from boolean algebra and they are mainly used to control program flow, finding them as part of an if, a while or some other control statement. See in the table below the logical string Perl operators.

Symbol	Name	Definition

!	not	It returns true if the operand is a null string or an undefined value and false otherwise.
		
		Example:
		`$v1 = !'some string here';`
		`$v2 = !'';`
		The $v1 variable will return false and the $v2 variable will return true.

not	not	The same meaning as above, but it is a lower-precedence version.

&&	and	This operator is used to determine if both operands are true.
		
		Example:
		`($v1, $v2) = ('abc', 'abzu');`
		`$v = $v1 == 'abc' && $v2 == 'abzu';`
		`print "$v (expected 1)";`

and	and	Same as above.

||	or	This operator is used to determine if either of the operands is true.
		
		Example:
		`($v1, $v2) = ('hello', 'good');`
		`$v = $v1 == 'hello' || $v2 == 'hello';`
		`print "$v (expected 1)";`

or	or	Same as above.

xor	exclusive or	It returns true if either but not both of the operands is true.
		
		Example:
		`($v1, $v2) = ('hello', 'good');`
		`$v = $v1 == 'hello' xor $v2 == 'world';`

		`print "$v (expected 1)";`

?	conditional operator	This is a ternary operator and it works like an if ... then ... else clause from the C language. If the left operand is true, it will return the central operand, otherwise the right operand.
		
		Example:
		`$v = ("abc" eq "abd") ? "It's equal.\n" :`
		`      "It's not equal. (expected)\n";`
		`print $v;`

2.3. OTHER STRING PERL OPERATORS

See in the table below other string Perl operators:

Symbol	Name	Definition

,	comma	In a scalar context, the comma operator evaluates each element from left to right and returns the value of the rightmost element. In a list context, the comma operator separates the elements of a literal list.
		
		Example:
		`# scalar context`
		`$colors = ('blue', 'red', 'yellow');`
		`print "$colors (expected yellow)\n";`
		`# list context`
		`@colors = ('blue', 'red', 'yellow');`
		`print "@colors[1] (expected red)\n";`

=>	comma	This operator is a special type of comma, for example ('dog', 'cat') is similar to (dog => 'cat'). Or it can be used to separate key/value pairs in a hash structure: %petColors = (dog => 'brown', cat => 'white');

-	negation	If the string operand begins with a plus or minus sign, the string negation operator returns a string with the opposite sign.
		
		Example:
		`$v = "-abcd";`
		`print (-$v, "(expected +abcd)\n");`

.	concatenation	This operator joins two or more strings like in the example below.
		
		Example:
		`$v = "Hello" . " World" . "!";`
		`print $v, " (expected Hello World!)\n";`

..	range operator	The range operator, in a list context, produces the range of values from the left value through the right

		value in increments of 1.
		
		Example:
		`@v = ('ab' .. 'ae');`
		`print @v, " (expected abacadae)\n";`

x	repetition	The repetition operator returns the first operand (which is a string) repeated by the number of times specified by the second operand (which is an integer).
		
		Example:
		`@v = 'ab' x 3;`
		`print @v, " (expected ababab)\n";`

2.4. STRING ASSIGNMENT PERL OPERATORS

String assignment Perl operators perform some type of string operation and then assign the value to the existing variable.

Symbol	Name	Definition

=	assignment	This is the ordinary assignment operator.
		
		Example:
		`$v = 'Hello World!';`
		`print $v, " (expected Hello World!)\n";`

&&=	logical and	It's a combination between the logical "&&" and the assignment operators.
		
		Example:
		`$v = 1;`
		`$v &&= 'abc' eq 'abc';`
		`print $v, " (expected 1)\n";`

||=	logical or	It's a combination between the logical "||" and the assignment operators.
		
		Example:
		`$v = 0;`
		`$v ||= 'abc' eq 'abc';`
		`print $v, " (expected 1)\n";`

.=	concatenation	It's a combination between the concatenation "." and the assignment operators.
		
		Example:
		`$v = "Hello ";`
		`$v .= 'World!';`
		`print $v, " (expected Hello World!)\n";`

x=	repetition	It's a repetition assignment operator.
		
		Example:
		`$v = "true ";`
		`$v x= 2;`

		`print $v, " (expected true true )\n";`

3. SPECIAL PERL OPERATORS

See in the table below the special Perl operators:

Symbol	Name	Definition
\	reference	We call reference the scalar value that contains a memory address. In order to reference a variable, we use the operator. Look at the below snippet code to see how to use it. Example: `$v="Hello World!";` `$ref_v=\$v;` `print $$ref_v, " (expected Hello World!)\n";`
->	dereference	The dereference operator was used for the first time in the Perl 5 language. This operator let us to access and manipulate the elements of an array, hash, object of a class data structure. If you use the reference operator to reference a scalar variable, before you use it you must dereference the reference variable. Example: `@v = ('black', 'white', 'blue', 'orange');` `$vRef = \@v; # the reference variable` `print $v[1], " (expected white)\n";` `$vRef->[1] = 'red'; # dereference before use` `print $v[1], " (expected red)\n";`
=~	pattern binding	This binary operator binds a string expression to a pattern match. The string which is intend to bind is put on the left meanwhile the operator itself is put on the right. We use the pattern binding operator in the case we have a string which is not stored in the $_ variable and we need to perform some matches or substitutions of that string. Example: `$v = "black and white";` `if($v =~ m/white/) {` `  print "Yes (expected)\n";` `}` `$v =~ s/black and white/red/;` `print $v, " (expected red)\n";`
!~	pattern binding (not)	This operator is similar the operator above, but the return value is negated logically. Example: `$v = "black and white";` `if($v =! m/yellow/) {`

```
  print "Yes (expected)\n";
}
```