

Perl 5 Constants and Variables						
Perl Variables Names	Scalar Naming Conventions			Array Naming Conventions	All: underscore or letter of the first character.	
Case is significant in all names. ASCII by default, UTF-8 if the utf8 pragma is used.	<ul style="list-style-type: none">Local variables:Global variables:Constants:All variables:		\$lowercase \$Title_Case \$UPPER_CASE words separated by underscores.	Similar conventions, except that array names should be plural . <ul style="list-style-type: none">@locals@Global_Arrays@CONSTANT_ARRAYS	<ul style="list-style-type: none">Module names are MixedCaseNoUnderscoresConstants are UPPERCASE_WITH_UNDERSCORESPackage wide vars are Mixed_Case_With_UnderscoresFunctions/methods are lowercase_with_underscoresAvoid ALLUPPERCASE: used by Perl special variables.	
Perl types	Sigil	Examples	Meaning		Extra Info	
Scalar	\$	\$foo \$days[28] \$days{'Feb'} \${days} \$Dog::days \$Dog'days \$#days \$days->[28] \$days[0][2] \$d{99}{'Feb'} \$d{99, 'Feb'}	Simple scalar value 29 th element of array @days Value associated with the <i>Feb</i> key of hash %days Same as \$days, but unambiguous before alphanumerics. Useful inside strings <u>for interpolation of variables followed by other letters</u> . The \$days variable inside the Dog package. Same as above. However this is an archaic use of the single quote. Last index of array @days. 29 th element of array pointed to by reference \$days. Multi-dimensional array Multi-dimensional hash Multi-dimensional hash emulation			
Array	@	@days @days[3,4,5] @days[3..5]	Array containing (<i>\$days[0]</i> , <i>\$days[1]</i> , ... <i>#days[\$#days]</i>) . Array slice containing (<i>\$days[3]</i> , <i>\$days[4]</i> , <i>\$days[5]</i>) . Array slice containing (<i>\$days[3]</i> , <i>\$days[4]</i> , <i>\$days[5]</i>) .			
<ul style="list-style-type: none">slices	<ul style="list-style-type: none">Use a slice to select multiple elements from a list, array, or hash.Don't use a slice when you know you need exactly one element.An lvalue slice imposes list context on the righthand side.					
<ul style="list-style-type: none">Anonymous arrays	<ul style="list-style-type: none">What are the advantages of anonymous array? @ StackOverflowPerlref @ Perldoc, Perl reference tutorial @ Perldoc			<ul style="list-style-type: none">Anonymous array := a type of array reference.Array reference allows Perl to treat the array as a single item.<ul style="list-style-type: none">This can be used to build, nested data structures.		
Hash/associative array	%	%days @days{'J','F'}	Associative array (hash): keys-value pairs. Can be initialized as: <ul style="list-style-type: none"><i>%days = (Jan => 31, Feb => \$leap? 29 : 28, ...)</i><i>%days = ("Jan, 31, 'Feb', \$leap? 29 : 28, ...)</i> Hash slice containing (<i>\$days{'J'}</i> , <i>\$days{'F'}</i>) .		Initialize a hash slice with array context: <i>@char_to_num{'A' .. 'Z'} = 1 .. 26;</i>	
Subroutine	&	&foo	& is needed to create reference to subroutine.			
Typeglob	*	*foo	See: Advanced Perl Programming , 1st Edition Section 3.2			
7 kinds of package variables or variable-like elements in Perl:	1. scalar variables 2. array variables 3. hash variables		4. subroutine name 5. format names <ul style="list-style-type: none">how to format output in Perl?, Perl-FormatsSee write and select		6. file handles 7. directory handles	
Scalar values			Numeric literals examples. Note: leading 0 work only for literals, not for string-to-number conversions.		Useful related builtin functions	
<ul style="list-style-type: none">numeric:	<ul style="list-style-type: none">integer : using the system's native format.<ul style="list-style-type: none">bigint - transparent big integer support.bignum - transparent big number support.floating-point : using the system's native format.<ul style="list-style-type: none">bigrat - transparent big rational number support.		<pre>my \$x = 12345; # integer my \$x = 12345.67; # floating point my \$x = 6.02e23; # scientific notation my \$x = 0x1f.0p3; # power2 exponent: <i>Perl >= v5.22</i> my \$x = 4_294_967_296; # underline for legibility my \$x = 0x1234_5678; # underline in hex is also OK my \$x = 0377; # octal my \$x = 0o377; # octal also <i>Perl >= v5.34</i> my \$x = 0xffff; # hexadecimal my \$x = 0b1100_0010; # binary</pre>		<ul style="list-style-type: none">oct - supports binary, octal, hexhexPOSIX::ceilPOSIX::floorabs	
<ul style="list-style-type: none">string	<ul style="list-style-type: none">double-quoted strings: perform backslash and variable interpolation of expression that begin with \$ (a scalar) or @ (an array). Hashes cannot be interpolated.single-quote strings: only perform \' and \\ substitution (to ' and \ respectively), nothing else.Single quote and double quote strings can spread multiple lines: it embeds the newline character on each new line.But \n is only expanded in double quoted strings! In single quote string it is treated as two characters; no substitution is done (as explained above).					
<ul style="list-style-type: none">Unicode support	To use Unicode literally in a program, add the utf8 pragma : use utf8;			See: Perl Unicode Tutorial , Perl Unicode Introduction , Perl Unicode Support @ perldoc		
<ul style="list-style-type: none">Quote constructs	Customary	Generic	Meaning	Interpolates?	Notes	
See: <ul style="list-style-type: none">Strings in Perl: quoted, interpolated and escaped	'' "" `` () // s/// tr/// ""	q// qq// qx// qw// m// s/// y/// qr//	Literal string Literal string Command execution World list Pattern match Pattern substitution Character translation Regular expression	No Yes Yes No Yes Yes No Yes	<ul style="list-style-type: none">Not all characters can be used as the / separator. { }, () and < > can also be used.You can use whitespace between the quote specifier and its initial bracketing character:<pre>my \$chuck_of_code = q { if (\$condition) { print "Salut!"; } };</pre>It's also possible to write: s<foo>(bar) and tr(a-f)[A-F] as well as:<pre>tr (a-f) [A-F];</pre>Array variables are interpolated by joining all elements with the separator specified by the \$' special variable (\$LIST_SEPARATOR) .	
<ul style="list-style-type: none">Character escapes (only inside double quoted strings)	\a \b \e \f \n \r \t	Alert (bell) Backspace ESC character Form feed Newline (usually LF) Carriage return (Usually CR) Horizontal tab	\e \033 \o{3f} \x7f \x{263a} \cC	ESC character ESC in octal ESC in octal DEL in hexadecimal Character number 0x263A Control-C		Any Unicode code point, by name: \N{LATIN SMALL LETTER E WITH ACUTE} é \N{ U+E9 } é
<ul style="list-style-type: none">translation escapes (only inside double quoted strings)	\u \l	Force next character to titlecase Force next character to lowercase	\U \L \F \Q	Force all following characters to uppercase. Ends at \E Force all following characters to lowercase. Ends at \E Force all following characters to Unicode fold case. Ends at \E Backslash all following non alphanumeric characters. Ends at \E		\E Ends \U, \L, \F or \Q
<ul style="list-style-type: none">bareword	In Perl, a <i>bareword</i> refers to a sequence of characters suitable for an identifier. It's not quoted. By default Perl allows barewords to behave like strings. <ul style="list-style-type: none">This is not allowed when any of use strict; or use strict "subs"; or use v5.12; is specified.					
<ul style="list-style-type: none">Here documentsHere docs @ Perl mavenPerl here doc @Wikipedia	Perl here-documents are a form of line oriented quoting. There are several forms of here documents, where the identifier (like EOF used below, but can be any word) must be placed at the beginning of the terminating line: <ul style="list-style-type: none">Default : <<EOF; Supports variable interpolation.Double quotes: <<"EOF"; Supports variable interpolation. Can also be written with whitespace as in <<"EOF";Single quotes: <<'EOF'; Does not support interpolation. Can also be written with whitespace as in <<'EOF';backticks: <<`EOF`; Execute commands in a shell and return text printed on stdout. Can also be written with whitespace as in <<`EOF`;indented: <<~EOF; Allows indenting the here-doc string. Can also use the ~ with the other forms: <<~\EOF, <<~"EOF", <<~"EOF", <<~`EOF`They can also be stacked and text can be transformed. See the documentation.					
<ul style="list-style-type: none">Perl Regexp info, cheatsheets & regexp testers	<ul style="list-style-type: none">Regexp TutorialLearn PCRE in X minutes		<ul style="list-style-type: none">PCRE cheatsheet		<ul style="list-style-type: none">Debuggex regexp testerregex101RegEx Pal	
Perl Constants	<ul style="list-style-type: none">Perl pragma to declare constants. ⚠ But be aware that these are still not read-only, that they inject sub-routines and have several limitations. Read the doc!!CPAN modules for defining constants by Neil Bowers . Of particular interest: Const::Fast and Attribute::Constant for efficient read-only constants.					

<div><div>Perl Special Variables</div><div><div>• Perl Variables</div><div><div><div></div><div></div></div><div><div>To get information about a Perl special variable from the command line use the perldoc -v command.</div><div>To get information about \$< use: perldoc -v '\$<'</div></div></div></div></div>					
• General variables					
default input and pattern searching space	<div><ul style="list-style-type: none">\$ARG\$_</div>	subroutine parameters	<div><ul style="list-style-type: none">@ARG@_</div>		
list separator	<div><ul style="list-style-type: none">\$LIST_SEPARATOR\$"</div>	Subscript separator for multidimensional array emulation	<div><ul style="list-style-type: none">\$SUBSCRIPT_SEPARATOR\$SUBSEP\$;</div>		
Name of executed program	<div><ul style="list-style-type: none">\$PROGRAM_NAME\$0</div>	Name used to execute the current copy of Perl	<div><ul style="list-style-type: none">\$EXECUTABLE_NAME\$^X</div>		
Perl process ID	<div><ul style="list-style-type: none">\$PROCESS_ID\$PID\$\$</div>				
Process real GID	<div><ul style="list-style-type: none">\$REAL_GROUP_ID\$GID\$(</div>	Process effective GID	<div><ul style="list-style-type: none">\$EFFECTIVE_GROUP_ID\$EGID\$)</div>		
Process real UID	<div><ul style="list-style-type: none">\$REAL_USER_ID\$UIG\$<</div>	Process effective UID	<div><ul style="list-style-type: none">\$EFFECTIVE_USER_ID\$\$EUID\$></div>		
Special variables in sort	<div><ul style="list-style-type: none">\$a\$b</div>	Example: by default Perl sort function sorts strings. Pass a sorting function that uses the <=> equality operator to force numerical comparisons: <code>@sorted = sort { \$a <=> \$b } @unsorted;</code>			
Current environment	%ENV <div>Environment variable accessed as an associative array (a hash).</div> <div>• See: Perl: How to access shell environment variables through Perl associative arrays.</div>				
Perl interpreter revision, version and subversion	<div><ul style="list-style-type: none">\$OLD_PERL_VERSION\$]</div>	Perl interpreter revision, version and subversion	<div><ul style="list-style-type: none">\$PERL_VERSION\$^V</div>		
Maximum file descriptor	<div><ul style="list-style-type: none">\$SYSTEM_FD_MAX\$^F</div>				
Fields of each line when auto-split mode is on.	@F				
Include Directories	@INC	Included filenames	%INC	Hook localization (?)	\$INC
inplace-edit extension value	<div><ul style="list-style-type: none">\$INPLACE_EDIT\$^I</div>				
Package's class parent classes	@ISA				
Emergency memory pool	\$^M				
Maximum block nesting	\${^MAX_NESTED_EVAL_BEGIN_BLOCKS}				
Name of OS where this Perl was built	<div><ul style="list-style-type: none">\$OSNAME\$^O</div>				
Signal handlers	%SIG				
Coderefs for various perl keywords	%{^HOOK}				
Time when program began running	<div><ul style="list-style-type: none">\$BASETIME\$^T</div>				
• Variables related to regular expressions					
captured sub-patterns	\$<digit>(\$1, \$2, ...)				
Capture buffer content	@{^CAPTURE}				
String matched	<div><ul style="list-style-type: none">\$MATCH\$&</div>	String matched (compiled regexp)	\${^MATCH}		
String preceding match	<div><ul style="list-style-type: none">\$PREMATCH\$`</div>	String preceding match (compiled regexp)	\${^PREMATCH}		
String following match	<div><ul style="list-style-type: none">\$POSTMATCH\$'</div>	String following match (compiled regexp)	{^POSTMATCH}		
Last capture group	<div><ul style="list-style-type: none">\$LAST_PAREN_MATCH\$+</div>	Most recently closed capture group	<div><ul style="list-style-type: none">\$LAST_SUBMATCH_RESULT\$^N</div>		
Match capture key values	<div><ul style="list-style-type: none">%{^CAPTURE}%LAST_PAREN_MATCH%+</div>				
Match start offsets	<div><ul style="list-style-type: none">@LAST_MATCH_START@-</div>	Match ends offsets	<div><ul style="list-style-type: none">@LAST_MATCH_END@+</div>	Named captured groups	<div><ul style="list-style-type: none">%{^CAPTURE_ALL}%-</div>
Last successful pattern	\${^LAST_SUCESSFUL_PATTERN}				
Result of last successful regexp assertion	<div><ul style="list-style-type: none">\$LAST_REGEXP_CODE_RESULT\$^R</div>				
Maximum regexp nested group	\${^RE_COMPILE_RECURSION_LIMIT}				
regexp debug flag	\${^RE_DEBUG_FLAG}				
regexp internal optimization/memory	\${^RE_TRIE_MAXBUF}				
• Variables related to file handles					
Name of current file read from <> <div>• diamond operator <></div>	\$ARGV	Command line arguments of the script	@ARGV	Number of arguments minus one	\$#ARGV

Special file handle that iterates over command-line filenames in @ARGV	ARGV	Special file handle that points to currently open output file when doing edit-in-place processing	ARGVOUT	
Output field separator for the print operator	<ul style="list-style-type: none"> IO::Handle->output_field_separator(EXPR) \$OUTPUT_FIELD_SEPARATOR \$OFS \$, 		Current line number for the last file handled accessed	<ul style="list-style-type: none"> HANDLE->input_line_number(EXPR) \$INPUT_LINE_NUMBER \$NR \$.
Input record separator (newline by default)	<ul style="list-style-type: none"> IO::Handle->input_record_separator(EXPR) \$INPUT_RECORD_SEPARATOR \$RS \$/ 		Output record separator	<ul style="list-style-type: none"> IO::Handle->output_record_separator(EXPR) \$OUTPUT_RECORD_SEPARATOR \$ORS \$\
Auto-flush control <ul style="list-style-type: none"> See: order of output @ Perl Maven 	<ul style="list-style-type: none"> HANDLE->autoflush(EXPR) \$OUTPUT_AUTOFLUSH \$! 		Last read file handle	\$_{^LAST_FH}
<ul style="list-style-type: none"> Variables related to format 				
Current value of the write() accumulator for format() lines.	<ul style="list-style-type: none"> \$ACCUMULATOR \$_A 			
Form feed format. defaults to \f	<ul style="list-style-type: none"> IO::Handle->format_formfeed(EXPR) \$FORMAT_FORMFEED \$_L 		Set of characters after which a string may be broken to fill continuation fields	<ul style="list-style-type: none"> IO::Handle->format_line_break_characters EXPR \$FORMAT_LINE_BREAK_CHARACTERS \$:
Number of lines left on the page on currently selected output channel	<ul style="list-style-type: none"> HANDLE->format_lines_left(EXPR) \$FORMAT_LINES_LEFT \$- 		Current page length of current output channel	<ul style="list-style-type: none"> HANDLE->format_lines_per_page(EXPR) \$FORMAT_LINES_PER_PAGE \$=
Name of current top-page format of output channel	<ul style="list-style-type: none"> HANDLE->format_top_name(EXPR) \$FORMAT_TOP_NAME \$_^ 		Report format name of output channel	<ul style="list-style-type: none"> HANDLE->format_name(EXPR) \$FORMAT_NAME \$~
<ul style="list-style-type: none"> Error Variables 	The variables \$@, \$!, \$^E, and \$? contain information about different types of error conditions that may appear during execution of a Perl program. They correspond to errors detected by the Perl interpreter, C library, operating system, or an external program, respectively.			
Perl error from the last eval operator	<ul style="list-style-type: none"> \$EVAL_ERROR \$@ 		Current state of interpreter	<ul style="list-style-type: none"> \$EXCEPTIONS_BEING_CAUGHT \$_AS
Current value of C errno integer variable	<ul style="list-style-type: none"> \$OS_ERROR \$ERRNO \$! 	\$! returns the system variable errno when used in a numeric context, but returns the string from pererror() when used in string context.	Hash of error names to 0 or 1, set to 1 if current error is this error.	<ul style="list-style-type: none"> %OS_ERROR %ERRNO %!
OS detected error	<ul style="list-style-type: none"> \$EXTENDED_OS_ERROR \$_^E 			
Status returned by last pipe close, backtick command, wait, waited, or system() call.	<ul style="list-style-type: none"> \$CHILD_ERROR \$? 		native status returned by last pipe close , backtick command, wait() or waitpid() or system() call	\$_{^CHILD_ERROR_NATIVE}
Current value of warning switch	<ul style="list-style-type: none"> \$WARNING \$_^W 		Current set of warning checks enabled by the use warnings pragma	\$_{^WARNING_BITS}
<ul style="list-style-type: none"> Variables related to the interpreter state 	These variables provide information about the current interpreter state.			
Flag associated with the -c switch	<ul style="list-style-type: none"> \$COMPILING \$_^C 		The current value of the debugging flags	<ul style="list-style-type: none"> \$DEBUGGING \$_^D
Current phase of the perl interpreter	\$_{^GLOBAL_PHASE}			
Compile-time hints for the perl interpreter. Internal use only	\$_^H		Values of compiled statements	%^H
Input/Output Layers. Internal use by PerlIO only.	\$_{^OPEN}			
Debugging support. Internal variable.	<ul style="list-style-type: none"> \$PERLDB \$_^P 			
Taint mode	\$_{^TAINT}		Safe locale operations availability	\$_{^SAFE_LOCALES}
Unicode Settings of Perl	\$_{^UNICODE}			
Internal UTF-8 offset caching code state	\$_{^UTF8CACHE}		State of UTF-8 locale detected by perl at startup.	\$_{^UTF8LOCALE}
<ul style="list-style-type: none"> Deprecated and removed variables: 	\$# \$* \$[\$_{^ENCODING} \$_{^WIN32_SLOPPY_STAT}			


Perl 5 open file 🚧🚧🚧

References	<ul style="list-style-type: none"> open @ perldoc browser Writing to files with Perl @ Perl Maven open file in-memory @ stackOverflow 	<ul style="list-style-type: none"> Stupid open() tricks @Perl.com: <ul style="list-style-type: none"> No explicit filename create an anonymous temporary file print to a string read lines from a string

Perl 5 Statements ⚠️

Conditional statements			
Loop control	<ul style="list-style-type: none">• <code>while (condition) { ... }</code>• <code>until (condition) { ... }</code>	loop control keywords: <ul style="list-style-type: none">• <code>next</code> : starts the next iteration of the loop.• <code>last</code> : exits the loop.• <code>redo</code> : restarts the loop block without evaluating the condition again.	loop control keywords: <ul style="list-style-type: none">• <code>continue</code> block: executed before evaluating condition again.
Statement modifiers	<ul style="list-style-type: none">• <code>if EXPR</code>• <code>unless EXPR</code>• <code>while EXPR</code>• <code>until EXPR</code>• <code>for LIST</code>• <code>foreach LIST</code>• <code>when EXPR</code>	The for and foreach statements impose a list context ; the complete list is processed. Therefore a loop like the following trying to stop on a line that has " <code>__END__</code> " on it will not work since it reads all of STDIN: <pre>foreach (<STDIN>) { last if ?__END__/ ...; }</pre>	The while statement imposes a scalar context ; it takes one line at a time from <STDIN> and the following code works properly: <pre>while (<STDIN>) { last if /__END__/ ...; }</pre>

Perl 5 Functions ⚠️

Perl Functions Perl syntax	 To get information about a Perl function from the command line use the perldoc -f command. <ul style="list-style-type: none">• To get information about <code>print</code> use: perldoc -f print		
⚠️ Cautionary notes			
<ul style="list-style-type: none">• <code>each</code> keyword is broken• Use <code>Var::Pairs</code> instead.	Do NOT use the built-in <code>each</code> . It is broken, as described by Damian Conway in his Modern Perl Best Practice O'Reilly course , section control structure. <ul style="list-style-type: none">• <code>each</code> is not re-entrant:<ul style="list-style-type: none">• nested loops of each over the same hash does not work as expected and will create infinite loop since the nested loop each juts iterates from where the first loop each left it.• Exiting the loop leaves the state of the each internal pointer at the current location.<ul style="list-style-type: none">• If you use each on the same hash later it will resume from where it left, it will not start form the beginning.		
print functions	<ul style="list-style-type: none">• <code>print</code>• <code>say</code>	<code>use feature qw(say);</code> or <code>use v5.10;</code> (or higher). Like <code>print</code> , but implicitly appends a newline at the end of the list.	

PerlTidy formatting control ⚠️

perltidy option	Option	Impact
indentation style	<ul style="list-style-type: none">• <code>-bl</code>,• <code>--opening-brace-on-new-line</code>• <code>--brace-left</code>	<ul style="list-style-type: none">• Without this option (the default) the code indentation style selected is K&R style.• With this option, the indentation style is Allman/BSD style.