Perl Programming Cheatsheet

A concise reference for Perl syntax, built-in functions, regular expressions, and common programming idioms. This cheat sheet provides a quick guide to the essential elements of Perl for both beginners and experienced programmers.

Basic Syntax and Data Types

Variables

Scalars	Start with `$`.
	`$name = "John";`
	`$age = 30;`

Arrays	Start with `@`. `@names = ("John", "Jane", "Doe");`
Hashes	Start with `%`. `%ages = ( "John" => 30, "Jane" => 25, "Doe" => 40 );`
Variable Interpolation	Variables are interpolated in double-quoted strings. `print "Name: $name, Age: $age\n";`
Context	Perl uses scalar, list, and void context to determine how expressions are evaluated.

Operators

Arithmetic	`+, -, *, /, %, **` (exponentiation)
String	`.` (concatenation), `x` (repetition)
Comparison	Numeric: `==, !=, <, >, <=, >=` String: `eq, ne, lt, gt, le, ge`
Logical	`&&, ||, !, and, or, not`
Assignment	`=, +=, -=, *=, /=, .=`

Control Structures

Conditional Statements:

```
if ($age >= 18) {
  print "Adult\n";
} elsif ($age >= 13) {
  print "Teenager\n";
} else {
  print "Child\n";
}

unless ($name) {
  print "Name is undefined\n";
}
```

Loops:

```
for ($i = 0; $i < 10; $i++) {
  print "$i ";
}
print "\n";

while ($age < 60) {
  $age++;
  print "Age: $age\n";
}

foreach $name (@names) {
  print "Name: $name\n";
}
```

Loop Control:
`next` (continue), `last` (break), `redo`

Subroutines and Modules

Subroutines

Defining a Subroutine	`sub greet { my ($name) = @_; print "Hello, $name!\n"; }`
Calling a Subroutine	`greet("World");`
@_	Array containing arguments passed to the subroutine.
Return Values	Subroutines can return values using the `return` keyword. `sub add { my ($x, $y) = @_; return $x + $y; }`
Local Variables	Use `my` to declare local variables within a subroutine scope.

Modules

Using Modules	`use Module::Name;`

Creating Modules	Create a `.pm` file with the module name.
	`package My::Module;`
	`use Exporter qw(import);`
	`our @EXPORT = qw(my_function);`
	`sub my_function { print "Module function\n"; }`
	`1;`

`Exporter`	Used to export functions from the module's namespace.
`@EXPORT`	Array of function names to export.
Always return `1;`	Modules must return true to indicate successful loading.

Common Built-in Functions

`print`: Prints output.

```
print "Hello, world!\n";
```

`length`: Returns the length of a string.

```
$length = length("abc"); # $length is 3
```

`substr`: Extracts a substring.

```
$substring = substr("hello", 1, 3); # $substring is "ell"
```

`split`: Splits a string into an array.

```
@words = split(/ /, "hello world"); # @words is ("hello", "world")
```

`join`: Joins an array into a string.

```
$string = join(" ", @words); # $string is "hello world"
```

Regular Expressions

Basic Matching

Pattern Matching	`if ($string =~ /pattern/) { print "Match found\n"; }`
Substitution	`$string =~ s/old/new/;`
Translation	`$string =~ tr/a-z/A-Z/;`

Quantifiers

`*`	Zero or more times
`+`	One or more times
`?`	Zero or one time
`{n}`	Exactly n times
`{n,}`	n or more times
`{n,m}`	Between n and m times

Character Classes

`.`	Any character except newline
`\d`	Digit (0-9)
`\w`	Word character (a-z, A-Z, 0-9, _)
`\s`	Whitespace character (space, tab, newline)
`[abc]`	Any of the characters a, b, or c
`[^abc]`	Any character except a, b, or c

Anchors

`^`	Start of the string
`$`	End of the string
`\b`	Word boundary

Modifiers

`i`	Case-insensitive
`g`	Global (match all occurrences)
`m`	Multiline (treat string as multiple lines)
`s`	Treat string as single line
`x`	Ignore whitespace in pattern

File I/O and System Interaction

File Handling

Opening Files	`open(my $fh, "<", "input.txt") or die "Cannot open file: $!"; # Read`
	`open(my $fh, ">", "output.txt") or die "Cannot open file: $!"; # Write`
	`open(my $fh, ">>", "append.txt") or die "Cannot open file: $!"; # Append`

Reading from Files	`while (my $line = <$fh>) { chomp $line; print "Line: $line\n"; }`
Writing to Files	`print $fh "Hello, file!\n";`
Closing Files	`close($fh);`
$!	Contains the system error message.

Command Line Arguments

@ARGV	Array containing command-line arguments. `foreach $arg (@ARGV) { print "Argument: $arg\n"; }`

Getopt::Long	Module for parsing command-line options.
	`use Getopt::Long;`
	`my ($name, $age); GetOptions( "name=s" => \$name, "age=i" => \$age ) or die "Error in command line arguments\n";`
	`print "Name: $name, Age: $age\n";`

System Calls

`system`	Executes a system command. `system("ls -l");`
`qx//` or `` ` ``	Captures the output of a system command. `my $output = `date`; print "Date: $output\n";`
`exec`	Replaces the current process with a new one. Terminates the current script after successful execution. `exec("find . -name '*.txt'");`

Environment Variables

%ENV	Hash containing environment variables. `print "PATH: $ENV{PATH}\n";`