A newline character.

$ scalar
@ array
% hash
& subroutine
* typeglob

Double quotes perform variable interpolation and backslash interpretation. Single quotes do not.
Back ticks capture the output from executing a command.

`""` or `0`, as appropriate.

As the expected type, depending on context.

interpolative context

list context

```
@threeprimes = (2, 3, 5);
($a, $b, $c) = @threeprimes;
```

Either involves a scalar so use `$` not `@`.

```
$lst[n] = new_el
print $lst[n]
```

```
@birthmonths = (
    "John" => "February",
    "Mary" => "March",
);
```

Arrows are just a nicer way of writing more arrows.

```
$hash{"key"}
$hash{"key"} = val
```

As with arrays, notice the use of `$` when dealing with individual elements.

Nouns can be singular (scalars) or plural (arrays and hashes).
Verbs can be procedures or functions.

```
perl -e 'some perl'
perl file.pl
```

The `-w` option prints warnings.

A data type that can represent files, devices, sockets, and pipes.

`STDOUT` and `STDERR` are provided by default.

Use `open`, whose simplest form is:

```
open(HANDLENAME, "filename");
```

Readonly (default): `"<filename"`
Write (clobber): `">filename"`
Write (append): `">>filename"`

```
open(FILEHANDLE, "file") or die "Error opening file: $!\n";
```

`$!` is the OS's error message.

```
$str = <FILEHANDLE>
$str = <STDIN>
```

```
print FILEHANDLE 'str'
```

These two are the same:

```
print STDOUT 'str'
print 'str'
```

`chop` removes the last character of the string passed to it, and returns it.
`chomp` removes endl from the string passed to it, and returns the *number* of characters removed.

Use a period (`.`) for string concatenation.

Because of weak typing addition of scalars created as strings, but that can be interpreted as numbers, would result in a sum.

Use the repeat operator (`x`).

- Using the dot operator (like idiomatic Java using `+`)
- Using interpolation (like idiomatic bash using `$`)
- Using list literal syntax

**

... specially.

In general, the following two are equivalent:

```
lval op= expr
lval = lval op expr
```

The value assigned.

Java

`&&` - `and`

`||` - `or`

`!` - `not`

The English ones bind less tightly.

There are numeric and string versions.

`==` - `eq`

`!=` - `ne`

`<` - `lt`

`>` - `gt`

`<=` - `le`

The standard "greater or equal" is absent.

Nonstandard, but handy:

`<=>` - `cmp`

`-e` (exists)
`-r` (readable)
`-w` (writable)
`-d` (directory)
`-f` (regular file
`-T` (text file)

Any string except `""` and `"0"`.
Any number except 0.

References evaluate to non-0 addresses.

Undefined values evaluate to 0 or `""`.

... booleans operators.

```
if (cond1) {
    #block1
}
elsif (cond2) {
  #block2
}
...
else {
  #blocknN
}
```

Braces are *always* required.

```
unless (cond) {
  #block
}
```

- while
- until
- for
- foreach

... `unless` which executes the block as long as the condition is falsy.

```
for ($i = 0; $i < num; $i++) {
    #body
}
```

```
foreach $el (@arr) {
    #body
}
```

`$el` is by reference, not value, so you can in fact mutate `@arr` by re-assigning `$el`.

`next`, `last`, and `redo`.

`next` is just like `continue`.
`last` is just like `break`.
`redo` executes the same iteration again.

Block labels, identifying which enclosing looping construction is being referred to.

It depends on context. List context can be provided various ways, including by assigning to a list or by an operator that provides the `LIST` context to its arguments.