```
$scalarref = \$foo;
$arrayref  = \@ARGV;
$hashref   = \%ENV;
$coderef   = \&handler;
$globref   = \*foo;
```

It isn't possible to create a true reference to an IO handle (filehandle or dirhandle) using the backslash operator. The most you can get is a reference to a typeglob, which is actually a complete symbol table entry. But see the explanation of the `*foo{THING}` syntax below. However, you can still use type globs and globrefs as though they were IO handles.

2. A reference to an anonymous array can be created using square brackets:

```
$arrayref = [1, 2, ['a', 'b', 'c']];
```

Here we've created a reference to an anonymous array of three elements whose final element is itself a reference to another anonymous array of three elements. (The multidimensional syntax described later can be used to access this. For example, after the above, `$arrayref->[2][1]` would have the value "b".)

Taking a reference to an enumerated list is not the same as using square brackets–instead it's the same as creating a list of references!

```
@list = (\$a, \@b, \%c);
@list = \($a, @b, %c);      # same thing!
```

As a special case, `\(@foo)` returns a list of references to the contents of `@foo`, not a reference to `@foo` itself. Likewise for `%foo`, except that the key references are to copies (since the keys are just strings rather than full-fledged scalars).

3. A reference to an anonymous hash can be created using curly brackets:

```
$hashref = {
    'Adam'  => 'Eve',
    'Clyde' => 'Bonnie',
};
```

Anonymous hash and array composers like these can be intermixed freely to produce as complicated a structure as you want. The multidimensional syntax described below works for these too. The values above are literals, but variables and expressions would work just as well, because assignment operators in Perl (even within local() or my()) are executable statements, not compile-time declarations.

Because curly brackets (braces) are used for several other things including BLOCKs, you may occasionally have to disambiguate braces at the beginning of a statement by putting a + or a `return` in front so that Perl realizes the opening brace isn't starting a BLOCK. The economy and mnemonic value of using curlies is deemed worth this occasional extra hassle.

For example, if you wanted a function to make a new hash and return a reference to it, you have these options:

```
sub hashem {        { @_ } }   # silently wrong
sub hashem {       +{ @_ } }   # ok
sub hashem { return { @_ } }   # ok
```

On the other hand, if you want the other meaning, you can do this:

```
sub showem {        { @_ } }   # ambiguous (currently ok, but may change)
sub showem {       {; @_ } }   # ok
sub showem { { return @_ } }   # ok
```