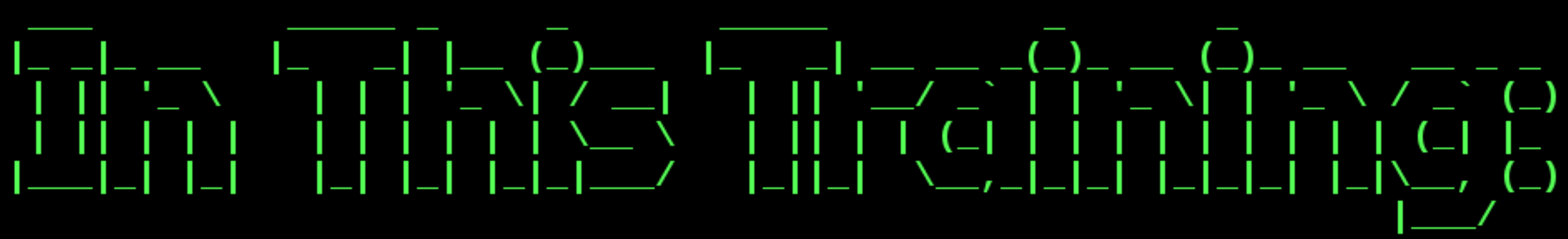


http://modernperlbooks.com/books/modern_perl_2014/

http://unnovative.net/perl.html

```
# Perl Quiz!!!!1

# Data::Dumper

# References

# Finish up scan.pl
```

Regexes

* Avoid list flattening
* Nested data
* Efficiency
* Out parameters
* Cyclic data!?!

Why wouldn't you want to use them?

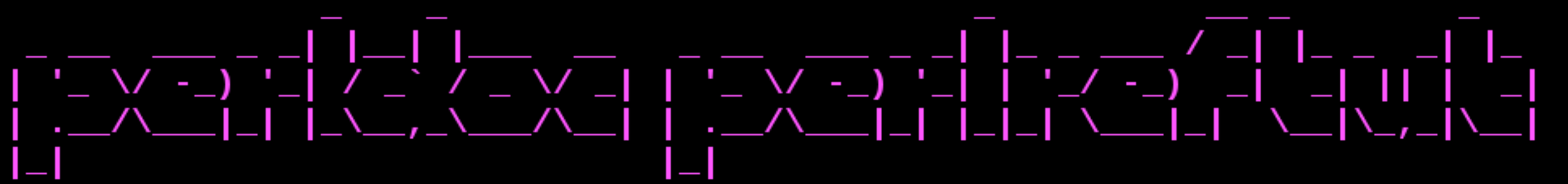
Whenever we pass values into a subroutine,
they arrive in a newly copied list

Now, that's one list, singular

* What do you do if you really wanted to pass in multiple distinct lists?

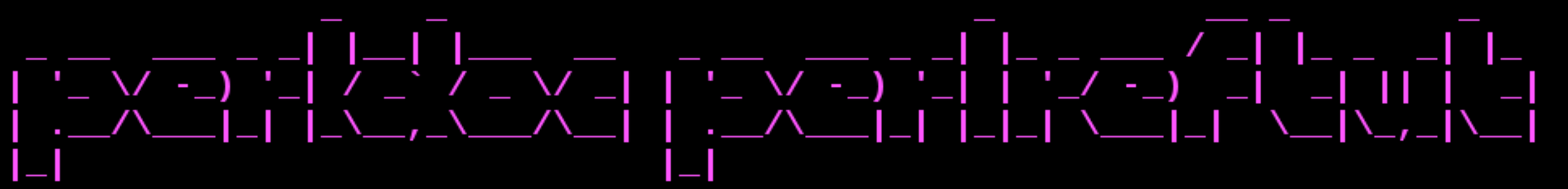
* What happens when you pass in a long list of arguments, or a few really big strings?

None of this sounds very efficient!



Paraphrasing 'perldoc perlreftut':

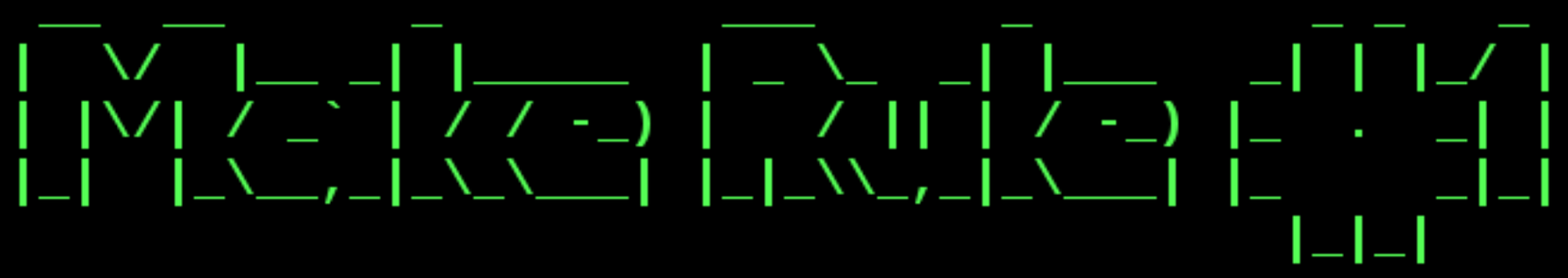
Fortunately, you only need to know 10% to get 90% of the benefit



This is that 10%

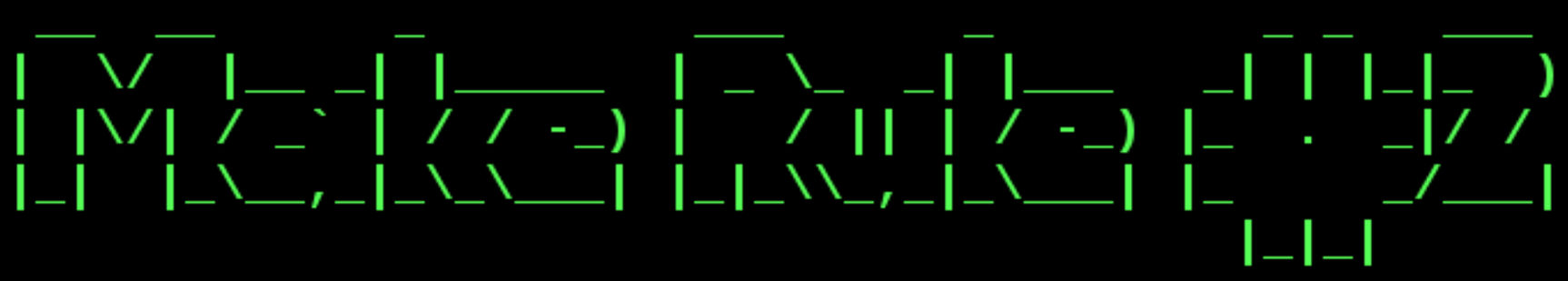
* There are two ways to make a reference

* There are three ways to use it once you have it



If you put a \ in front of a variable,
you get a reference to that variable.

```
my $ref = \$scalar;
```



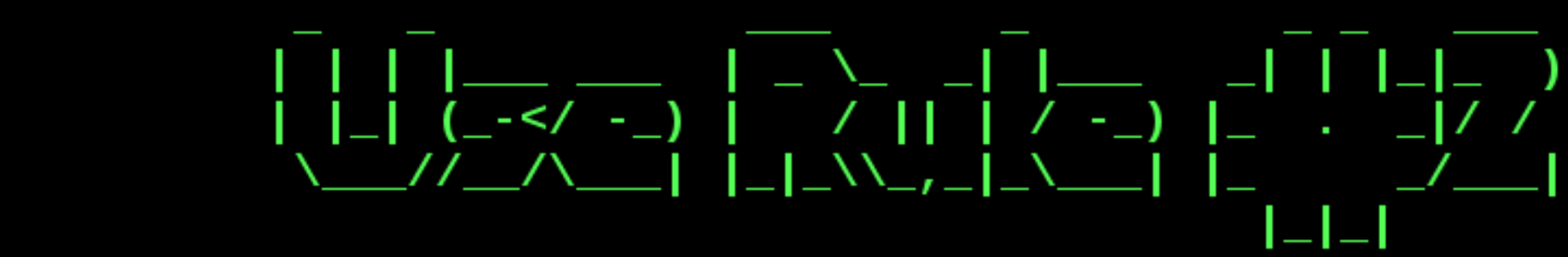
`[ ITEMS ]` makes a new, anonymous array,
and returns a reference to that array

```
my $array = [ 1, 'two', 3.0 ];
```

`{ ITEMS }` makes a new, anonymous hash,
and returns a reference to that hash

```
my $hash = {
   key => 'value',
   guanjian => 'shu',
   llave => 'valor',
};
```

Prefix the reference with an extra sigil for each level of dereference



You can always use an array reference in curly braces
in place of the name of an array

For example:

```
@{$aref}
```

instead of

```
@$aref
```

or

```
@a_real_array
```

Use Rule #2 is all you really need
it tells you how to do absolutely everything
you ever need to do with references

But the most common thing to do with an array or a hash
is to extract a single element,
and the **Use Rule 2** notation is cumbersome

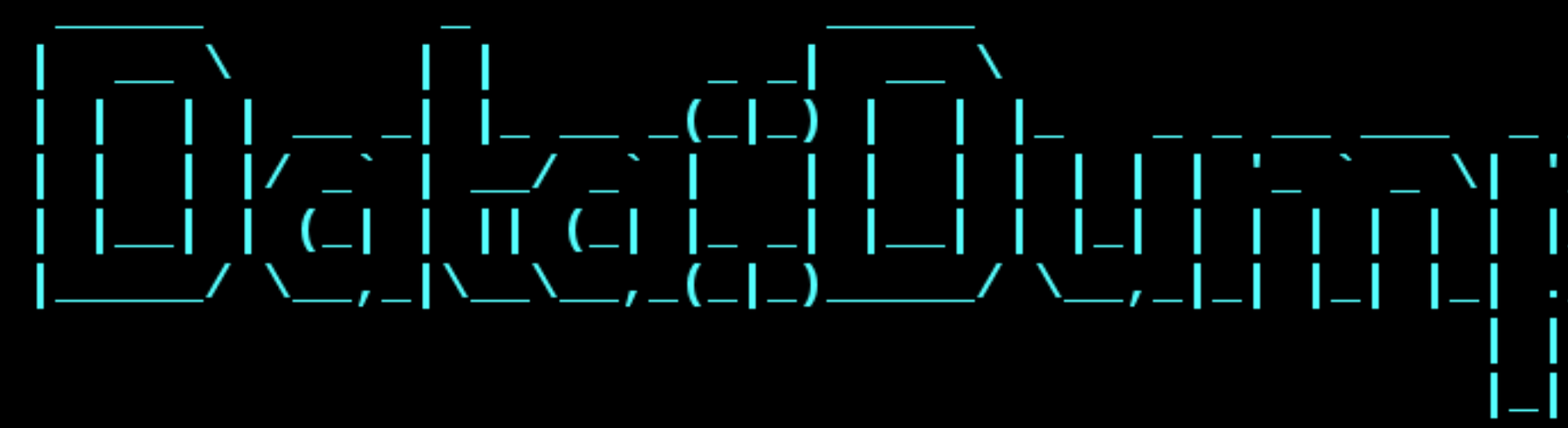
Use the dereferencing arrow ->

Hard to read

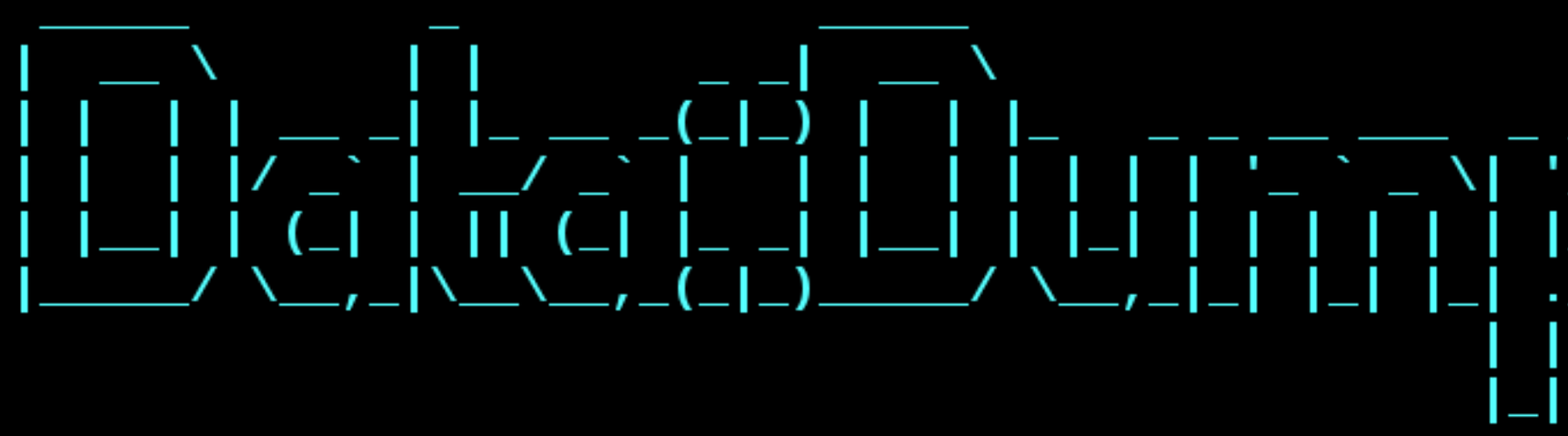
```
${$aref}[3]                    ${$href}{red}
```

Write these instead

```
$aref->[3]                     $href->{red}
```



This core module is invaluable as you work with complex, nested data structures based upon references



It works quite similarly to the debugger's `x` command

As it turns out, the output of Data::Dumper's methods
happens to be valid Perl code which will reconstitute the dumped data

You can (ab)use this feature to serialize data and impliment a rudimentary
configuration file system