```
open my $fh, "foo" or die $!;
local $/; # enable localized slurp mode
my $content = <$fh>;
close $fh;
```

But the following code is quite bad:

```
open my $fh, "foo" or die $!;
undef $/; # enable slurp mode
my $content = <$fh>;
close $fh;
```

since some other module, may want to read data from some file in the default "line mode", so if the code we have just presented has been executed, the global value of $/ is now changed for any other code running inside the same Perl interpreter.

Usually when a variable is localized you want to make sure that this change affects the shortest scope possible. So unless you are already inside some short {} block, you should create one yourself. For example:

```
my $content = '';
open my $fh, "foo" or die $!;
{
    local $/;
    $content = <$fh>;
}
close $fh;
```

Here is an example of how your own code can go broken:

```
for (1..5){
    nasty_break();
    print "$_ ";
}
sub nasty_break {
    $_ = 5;
    # do something with $_
}
```

You probably expect this code to print:

```
1 2 3 4 5
```

but instead you get:

```
5 5 5 5 5
```

Why? Because nasty_break() modifies $_ without localizing it first. The fix is to add local():

```
    local $_ = 5;
```

It's easy to notice the problem in such a short example, but in more complicated code you are looking for trouble if you don't localize changes to the special variables.

The following list is ordered by scalar variables first, then the arrays, then the hashes.

$ ARG