Отличия переменных с лексической областью видимости от глобальных переменных

Данную статью я писала больше для себя. Поэтому, если кто-то почерпнет для себя что-нибудь полезное, я буду только рада.
Конечно, вы скажите зачем вообще нужна эта статья , если без применения директивы **use strict** можно спокойно пользоваться переменными **$var** , не указывая точно к какому типу они относятся. Но в этом случае все переменные, определенные как **$var**, будут видны из любой точки Вашей программы т.е. будут являться глобальными. Вообщем статью писала для себя, так что начнем.

my - совершенно закрытые переменные

Объявления с лексической областью видимости действуют только от точки, где находится объявление, и до конца самой внутренней из охватывающих областей видимости (блока , файла или eval). С помощью **my** создаются именно такие переменные - переменные с лексической областью видимости.
Пример 1:

```
#!/usr/bin/perl -w
use strict;
my $private=5;
{
    #Здесь видим внутреннюю переменную
    my $private = 10;
    print "Private = $private \n";
}
#Здесь видим наружную переменную
print "Private = $private \n";
```

Результат:

```
Private = 10
Private = 5
```

Пример 2:

```
{
    my $hello = "hello";
}
print $hello;
```

Результат:
При компиляции будет выдана ошибка
Global symbol "$hello" requires explicit package name
Это означает, что в пакете **main::** данная переменная не объявлена, а другого пакета с этой переменной Вы не указали.

our - глобальные переменные

Повторные объявления **our** не имеют смысла вложенности. Каждое вложенное **my** создает новую переменную, как в примере 1. Каждое вложенное **local** создает новое значение, как в примере 5. Всякий раз, когда используется **our**, речь идет о той же самой глобальной переменной без учета вложенности.
Пример 3:

```
#Здесь значение глобальной переменной равно 5
our $global_var=5;
{
    #Здесь изменяем значение ТОЙ ЖЕ САМОЙ переменной на 10
    our $global_var = 10;
}
#Теперь значение глобальной переменной равно 10
print "Global var = $global_var \n";
```

Пример 4:

```
{
    package Utils;
    our $sum;
    #С глобальными переменными можно работать и внутри методов
    sub create_sum{
        $sum = int(rand(5)) + 1;
    }
    sub printing_sum{
        print $sum,"\n";
    }
}
#Создаем random значение переменной sum
Utils::create_sum;
#Выводим это значение
Utils::printing_sum;
```

local - позволяют давать временные значения для глобальных переменных

Вот пример для использования **local** для "подмены" значения глобальной переменной.
Пример 5:

```
{
    package Man;
    our $name = "Greg";
    our $years = 25;
    my $weight = 80;
}
#Выдаст 25
print $Man::years,"\n";
#Выдаст ошибку "Use of uninitialized value $Man::weight"
print $Man::weight,"\n";
#Выдаст исходное имя Greg
print $Man::name,"\n";
```

```
sub func{
    #Выдаст текущее число лет
    print "our var = ", $Man::years, "\n";
    local $Man::name = "James";
    #Выдаст временное значение имени James
    print "local var = ",$Man::name, "\n";
    #Увеличиваем значение глобальной переменной
    foreach (1, 2, 3, 4) {
        $Man::years+= 1;
    }
}
func;
func;
#Теперь число лет 33
print $Man::years,"\n";
#Значение глобальной переменной name Greg
print $Man::name,"\n";
```

Также определение переменной, как **local** позволяет использовать это значение не только в текущем блоке, но и во всех вызванных из данного блока подпрограммах.
Пример 6:

```
#!/usr/bin/perl -w
use strict;
#Даем значения глобальным переменным
our $x=10;
our $y=10;
sub Display{
        print "x = $x y=$y \n";
}
sub Create{
        my $x=15;
        local $y=20;
        print "x = $x y=$y \n";
        Display();
}
Create();
```

Результат :

```
x = 15 y=20
x = 10 y=20
```

no strict 'refs' - разрешить использование "нежестких" (символических) ссылок

Отключение директивы strcit 'refs' позволет получить значение переменной, не по ссылке на неё, а лишь по имени, хранящемуся в другой переменной.
Пример 7:

```
#!/usr/bin/perl -w
use strict;
#Даем значения глобальным переменным
our $x=10;
```

```
our $y="x";
{
    #Снимаем ограничение жестких ссылок
    no strict 'refs';
    #Выводим значение переменной x по её имени, записанному в переменную y
    print ${$y}, "\n";

}
#Здесь можно получить значение переменной x, только через "реальную" (жесткую) ссылку
$y=\$x;
print ${$y}, "\n";
```

Аналогичным образом можно подставлять не только имена переменных, но и имена классов , пакетов и т.д.