"my" variable masks earlier declaration in same scope

my (/search/my) scope (/search/scope)

Это предупреждение на этапе компиляции, которое вы увидите, если по ошибке попытаетесь объявить одну и ту же переменную дважды в одной области видимости.

```
"my" variable ... masks earlier declaration in same scope at ... lin
```

Как это происходит, и как работает переобъявление переменной на каждой итерации цикла?

Если нельзя написать `my $x` два раза в одной области видимости, то как нам обнулить эту переменную?

Рассмотрим различия между этими двумя случаями:

Простой скрипт

```
1. use strict;
2. use warnings;
3.
4. my $x = 'this';
5. my $z = rand();
6. my $x = 'that';
7. print "OK\n";
```

В этом случае мы получим следующее предупреждение на этапе компиляции:

```
"my" variable $x masks earlier declaration in same scope at ... line
```

Понятно, что это только предупреждение, и при запуске скрипт также выведет "OK".

Блок в условном выражении

```
1. use strict;
2. use warnings;
3.
4. my $z = 1;
5. if (1) {
6.     my $x = 'this';
7.     my $z = rand();
8.     my $x = 'that';
9. }
```

Здесь мы получим следующее предупреждение:

```
"my" variable $x masks earlier declaration in same scope at ... line
```

В обоих случаях мы объявили `$x` дважды в одной области видимости, и в результате будет сгенерировано предупреждение на этапе компиляции.

Во втором примере мы также дважды объявили `$z`, но от этого никакого предупреждения не появилось. Это потому, что `$z` в блоке находится в другой области видимости (/oblast-vidimosti-v-perl).

Область видимости функции

Тот же код, но в функции:

```
1. use strict;
2. use warnings;
3.
4. sub f {
5.     my $x = 'this';
6.     my $z = rand();
7.     my $x = 'that';
8. }
9. f(1);
10. f(2);
```

Здесь мы тоже получим предупреждение на этапе компиляции (один раз) для переменной `$x`. Несмотря на то, что переменная `$z` будет "пробуждаться к жизни" несколько раз, при каждом выхове этой функции. Это в порядке вещей. Переменная `$z` не вызывает предупреждения: Perl может создавать одну переменную дважды, только нам нельзя этого делать. Во всяком случае, не в пределах одной области видимости.

Область видимости цикла for

Тот же код, но в цикле:

```
1. use strict;
2. use warnings;
3.
4. for (1 .. 10) {
5.     my $x = 'this';
6.     my $z = rand();
7.     my $x = 'that';
8. }
```

Здесь мы тоже получим то же предупреждение один(!) раз для $x , но не для $z .

В этом коде одно и то же происходит **каждую** итерацию: Perl выделяет память для переменной $z .

Что же в действительности означает "my"?

Смысл my $x в том, что вы говорите perl'у, а именно strict 'у, что вы хотите использовать private переменную **$x** в текущей области видимости (/oblast-vidimosti-v-perl). Без этого perl будет искать объявление этой переменной в областях видимости высшего уровня и, если нигде не найдет, то выдаст ошибку этапа компиляции Global symbol requires explicit package name (/global-symbol-requires-explicit-package-name). Каждый блок, каждый вызов функции, каждая итерация цикла - это отдельный мир.

С другой стороны, написав my $x дважды в одной области видимости, вы по сути пытаетесь сказать perl'у два раза одно и то же. В этом нет необходимости, и обычно это значит, что где-то допущена ошибка.

Другими словами, предупреждение, которое мы получаем, относится к **компиляции** кода, а не к выполнению. Оно относится к объявлению переменной разработчиком, а не к выделению памяти perl'ом во время выполнения.

Как обнулить существующую переменную?

Так если мы не можем написать my $x; дважды в одной области видимости, как нам сделать эту переменную "пустой"?

Прежде всего, если переменная объвлена внутри области видимости, то есть между фигурными скобками, она автоматически исчезнет, когда по ходу выполнения программа покинет эту область видимости (/oblast-vidimosti-v-perl).

Если же вам нужно "обнулить" скалярную переменную в текущей области видимости, присвойте ей undef , а если это массив или хэш (https://perlmaven.com/undef-on-perl-arrays-and-hashes), чтобы опустошить их, нужно присвоить им пустой список:

```
1. $x = undef;
2. @a = ();
3. %h = ();
```

Итак, внесем ясность. "my" сообщает perl'у, что мы хотим использовать переменную. Когда Perl доходит до кода с таким объявлением переменной, он выделяет память под эту перемунную и ее содержимое. Когда Perl дойдет до кода `$x = undef;` , `@x = ();` или `undef @x;` , он удалит содержимое существующей переменной.

Переводчик
Vladimir Zatoloka
(https://www.linkedin.com/in/vzatoloka/)

Автор
Gabor Szabo
(https://www.linkedin.com/in/szabgab/)

Published on 2013-07-25

If you have any comments or questions, feel free to post them on the source of this page in GitHub. Source on GitHub. (https://github.com/szabgab/perlmaven.com/tree/main/sites/ru/pages//my-variable-masks-earlier-declaration-in-same-scope.txt) Comment on this post (https://github.com/szabgab/perlmaven.com/issues/new?title=&body=https://ru.perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope)

English (https://perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope)
Italiano (https://it.perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope)
Português (https://br.perlmaven.com/variavel-my-mascara-declaracao-anterior)
Русский (https://ru.perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope) ✔
 体中文 (https://cn.perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope)
    (https://ko.perlmaven.com/my-variable-masks-earlier-declaration-in-same-scope)

about the translations (https://perlmaven.com/about#translations)

Global symbol requires explicit package name (/global-symbol-requires-explicit-package-name)
Область видимости переменных в Perl (/oblast-vidimosti-v-perl)