Imagine code such as this:
```ruby
def test_smth
  result = nil
  assert_nothing_raised { result = some_slow_calculation }
  assert_equal expected_answer, result
end
```

Line `result = nil` means nothing more than our intention to simply bind variable `result` in scope of assertion block with outer local variable. In large methods it can be much heavier to find out this obscured intention.

It may be better to introduce a method (if it can be implemented) or a keyword which simply creates a local variable (probably with nil value just to mark its existence):

```ruby
define_variable 'x' or (define_variable x if it's a keyword)
```

This should definitely not be a method, however I would welcome a 'local' keyword for this purpose.

Assigning to matz, but don't hold your breath; matz has rejected such a explicit variable declaration syntax many times.

---

Yusuke Endoh mame@tsg.ne.jp

I have always been curious why there is no dynamic way to create local variables (other then eval).

Eg. `x = 10` might be dynamically written:

```ruby
local :x, 10
```

Since we can create just about anything else dynamically, it seems like stark omission.

Since I am sick of 'var' and 'local' in other languages, I don't want to add explicit local variable declaration, that requires a new keyword. Introducing a new keyword may break existing programs.

trans (Thomas Sawyer) Your idea would eliminate many chances to optimize.

I see why then. Thanks.