# Endless method definition

**Status:** Closed  
**Priority:** Normal  
**Assignee:** nobu (Nobuyoshi Nakada)  
**Target version:**

## Description

Ruby syntax is full of "end"s. I'm paranoid that the ends end Ruby. I hope Ruby is endless.

So, I'd like to propose a new method definition syntax.

```
def: value(args) = expression
```

As you see, there is no "end".

## Examples.

```
def: hello(name) =
  puts("Hello, #{name}")

hello("endless Ruby") #=> Hello, endless Ruby
```

```
def: inc(x) = x + 1
p inc(42) #=> 43
```

```
x = Object.new

def: x.foo = "FOO"
p x.foo #=> "FOO"
```

```
def: fib(x) =
  x < 2 ? x : fib(x-1) + fib(x-2)
p fib(10) #=> 55
```

## Limitations.

- def: foo x = x is invalid; the parentheses for formal arguments are mandatory.
- private def: foo = x is invalid; this method definition cannot be an method argument.

A patch is attached. No conflicts.

## Related issues:

- Related to Ruby master - Feature #5054: Compress a sequence of ends [Rejected]
- Related to Ruby master - Feature #5065: Allow "}" as an alternative to "end" [Rejected]
- Related to Ruby master - Feature #12241: super end [Rejected]

## Associated revisions

**Revision e853692 - 04/10/2020 09:02 AM - nobu (Nobuyoshi Nakada)**

Endless method definition [Feature #16746]

**Revision 302da060 - 04/10/2020 12:02 PM - nobu (Nobuyoshi Nakada)**

Reference to [Feature #16746] [ci skip]
I'm paranoid that the ends end Ruby. I hope Ruby is endless.

With so many cancellations of events this year, we may not want to want to jinx ruby too much for the 3.0 release this year because then 3.0 itself may become the endless version! (Or we could skip it, like PHP skipped a version ... I was briefly tempted to propose this on this special day today, but I really don't want to jinx it. I do, however had, remember matz joking last year about unforeseen events ... now I hope I just did not jinx it!)

#2 - 04/01/2020 02:33 AM - shyouhei (Shyouhei Urabe)
- Related to Feature #5054: Compress a sequence of ends added

#3 - 04/01/2020 02:33 AM - shyouhei (Shyouhei Urabe)
- Related to Feature #5065: Allow "}" as an alternative to "end" added

#4 - 04/01/2020 02:33 AM - shyouhei (Shyouhei Urabe)
- Related to Feature #12241: super end added

#5 - 04/01/2020 03:26 AM - mame (Yusuke Endoh)
- File deleted (endless-method-definition.patch)

#6 - 04/01/2020 03:26 AM - mame (Yusuke Endoh)
- File endless-method-definition.patch added

#7 - 04/01/2020 06:32 AM - matz (Yukihiro Matsumoto)
- Tags deleted (joke)

I totally agree with the idea seriously, (far better than #5054, #5065 and #12241) but don't like the syntax. First I thought

```
def foo(a) = expression
```

But it is a conflicting syntax.

Matz

#8 - 04/01/2020 04:03 PM - mame (Yusuke Endoh)
- Assignee set to nobu (Nobuyoshi Nakada)
- Status changed from Open to Assigned

Okay seriously. You should have written many "simple" method definitions that have only one expression:

```
# three-line style
def value
  @val
end

# one-line style
def value; @val; end
```

Surprisingly, according to the following rough estimate of ruby/ruby code base, this kind of simple method definitions account for 24% of the entire method definitions.

```
# three-line style: 8570
p Dir.glob("**/*.rb").map{|f| File.binread(f).scan(/\s*def.*\n.*\n\1end$/).size }.sum

# one-line style: 949
p Dir.glob("**/*.rb").map{|f| File.binread(f).scan(/\s*def.*;[^;]*;\s*end$/).size }.sum

# all method definitions: 39502
p Dir.glob("**/*.rb").map{|f| File.binread(f).scan(/\bdef\b/).size }.sum

# proportion = (8570 + 949) / 39502 = 24%
```
I agree that def foo(a) = expression is best, but I cannot implement it. Pass it to nobu (Nobuyoshi Nakada).

#9 - 04/02/2020 03:14 PM - ka8725 (Andrey Koleshko)
matz (Yukihiro Matsumoto) wrote in #note-7:

I totally agree with the idea seriously, (far better than #5054, #5065 and #12241) but don't like the syntax. First I thought

def foo(a) = expression

But it is a conflicting syntax.

Matz

How about this:

def foo(a):
    expression

#10 - 04/02/2020 03:28 PM - ruurd (Ruurd Pels)
shevegen (Robert A. Heiler) wrote in #note-1:

I'm paranoid that the ends end Ruby. I hope Ruby is endless.

Like - basically - a snake biting its own tail amirite ;)

#11 - 04/02/2020 04:36 PM - nobu (Nobuyoshi Nakada)
https://github.com/ruby/ruby/pull/2996

#12 - 04/02/2020 10:21 PM - ioquatix (Samuel Williams)
Have you considered adopting Python's whitespace sensitive indentation?

def hello(name):
    puts("Hello, #{name}")

hello("endless Ruby") #=> Hello, endless Ruby

#13 - 04/03/2020 03:28 AM - mame (Yusuke Endoh)
nobu (Nobuyoshi Nakada) implemented def foo(arg) = expression in one night. Also, it allows private def: foo = 42. Perfect.

https://github.com/ruby/ruby/pull/2996

matz (Yukihiro Matsumoto) We are ready to perform the experiment.

#14 - 04/05/2020 04:18 PM - retro (Josef Šimánek)
To be honest, I'm a little confused if this is serious proposal now, since it probably started as a joke and got some attention already at GitHub pull request (https://github.com/ruby/ruby/pull/2996) with mixed attitude.

Anyway for one-line method definitions I like part of https://bugs.ruby-lang.org/issues/5065 (originally at https://bugs.ruby-lang.org/issues/729, that was rejected because of invalid reason for the change request AFAIK). It looks "nature" to me since similar syntax is already available for blocks and often is used for one-liners. Ideally scope this syntax only for one-liners.

# original
def value; @val; end

# proposed - is that conflicting one?
def value { @val }

Alternatively, since the only use-case I have found in this issue is getter having different method and instance variable names, what about to support this out of attr_reader (and all attr_* friends)?

# few ideas
attr_reader :value, for: :val
# reads as -> attribute reader "value" for instance variable "val", probably impossible for multiple definitions on one line
attr_reader value: :val # this would support multiple definitions on one line

05/10/2020
Using the assignment operator like def value = expects the following behavior.

```ruby
value = 42
# Refer to variables outside scope
def hoge = value
hoge #=> 42
```

However, currently it is error: undefined local variable or method.
In the Endless method definition, the scope seems ambiguous, so I need to discuss whether to capture variables.

---

I'd like to experiment with this new syntax. We may find drawbacks in the future, but to find them, we need to experiment first.

Matz.

---

I'd like to experiment with this new syntax. We may find drawbacks in the future, but to find them, we need to experiment first.

If this was merged as an experiment, shouldn't it raise warning on usage (same as pattern matching does)?

---

It is not easy to control parsing time warnings, and bothers tests.

---

The exact same applies to pattern matching and yet we added a warning there. Testing/suppressing the warning is an eval away.

I think it's important for new experimental syntax to warn about it, especially when the right semantics are unclear.

My opinion is it should not capture, def/class/module never captured outside scopes and that shouldn't change.

---

Is it intended to allow multi-line definitions in this style?
I think we should not, and only single-line expressions should be allowed.

I think that's not the original purpose of this ticket and rather an incorrect usage of this new syntax, e.g., [https://twitter.com/devoncestes/status/1256222298431229133](https://twitter.com/devoncestes/status/1256222298431229133)

---

Files

| endless-method-definition.patch | 2.47 KB | 04/01/2020 | mame (Yusuke Endoh) |

---