Self - The current/default object

<Modules/Mixins | TOC | Constants >

At every point when your program is running, there is *one and only one* **self** - the current or default object accessible to you in your program. You can tell which object **self** represents by following a small set of rules.

Top level context

The top level context is before you have entered any other context, such as a class definition. Therefore the term top level refers to program code written outside of a class or module. If you open a new text file and type:

```
x = 1
```

you have created a top level local variable x. If you type:

```
def m
end
```

you have created a top level method - an instance method of **Object** (even though **self** is not **Object**). Top-level methods are always private. Ruby provides you with a start-up **self** at the top level. If you type:

```
puts self
```

it displays **main** - a special term the default **self** object uses to refer to itself. The class of the **main** object is **Object**.

Self inside class and module definitions

In a class or module definition, **self** is the class or module object.

```
# p063xself1.rb
class S
  puts 'Just started class S'
  puts self
  module M
    puts 'Nested module S::M'
    puts self
  end
  puts 'Back in the outer level of S'
  puts self
end
```

The output is:

```
>ruby p063xself1.rb
Just started class S
S
Nested module S::M
S::M
Back in the outer level of S
S
>Exit code: 0
```

Self in instance method definitions

At the time the method definition is executed, the most you can say is that **self** inside this method will be some future object that has access to this method.

```
# p063xself2.rb
class S
  def m
    puts 'Class S method m:'
    puts self
  end
end
s = S.new
s.m
```

The output is:

```
>ruby p063xself2.rb
Class S method m:
#<S:0x2835908>
>Exit code: 0
```

The output #<S:0x2835908> is Ruby's way of saying "an instance of S".

Self in singleton-method and class-method definitions

Singleton methods - those attached to a particular object can be called by only one object. When a singleton method is executed, **self** is the object that owns the method, as shown below:

```
# p063xself3.rb
obj = Object.new
def obj.show
  print 'I am an object: '
  puts "here's self inside a singleton method of mine:"
  puts self
end
obj.show
print 'And inspecting obj from outside, '
puts "to be sure it's the same object:"
puts obj
```

The output of the above example is:

```
>ruby p063xself3.rb
I am an object: here's self inside a singleton method of mine:
#<Object:0x2835688>
And inspecting obj from outside, to be sure it's the same object:
#<Object:0x2835688>
>Exit code: 0
```

Class methods are defined as singleton methods for class objects. Refer to the following program:

```
# p063xself4.rb
class S
  def S.x
    puts "Class method of class S"
    puts self
  end
end
S.x
```

The output is:

```
>ruby p063xself4.rb
Class method of class S
S
>Exit code: 0
```

self inside a singleton method (a class method, in this case) is the object whose singleton method it is.

<Modules/Mixins | TOC | Constants >