In this lecture, we will discuss…

- ✧ Blocks
- ✧ How they are used
- ✧ How to incorporate them into your own methods

Blocks

✧ **Chunks of code**

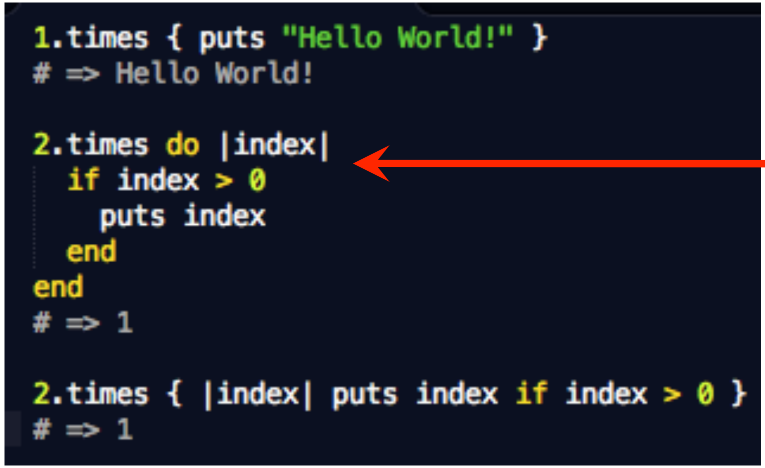
- Enclosed between either curly braces (`{ }`) or the keywords `do` and `end`
- Passed to methods as last “parameter”

Blocks

- **Convention:**
  - Use `{ }` when block content is a single line
  - Use `do` and `end` when block content **spans multiple lines**
- Often used as **iterators**

Blocks

- Can accept arguments



```ruby
1.times { puts "Hello World!" }
# => Hello World!

2.times do |index|
  if index > 0
    puts index
  end
end
# => 1

2.times { |index| puts index if index > 0 }
# => 1
```

Often accepts parameter(s) between ||

Coding with blocks

✧ Two ways to configure a block in your own method

Implicit

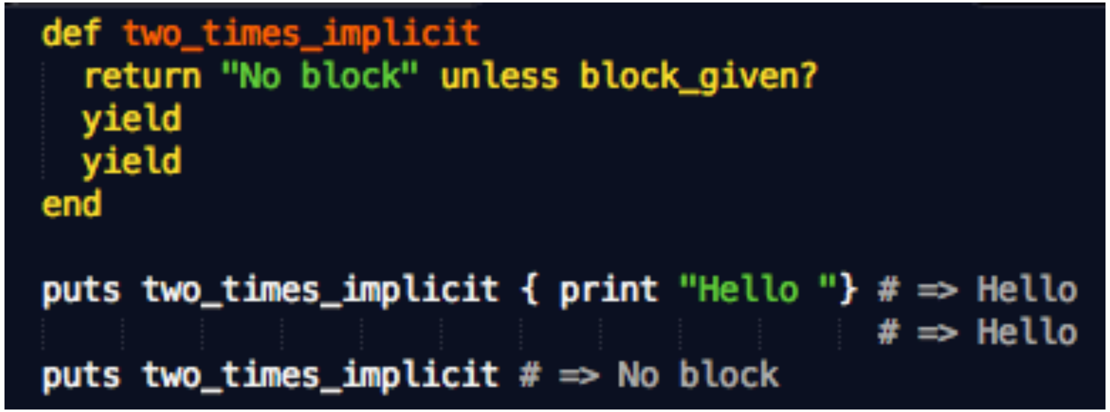
- **Use `block_given?` to see if block was passed in**
- **Use `yield` to "call" the block**

Explicit

- **Use `&` in front of the last "parameter"**
- **Use `call` method to call the block**

Implicit

- ✧ Need to check "`block_given?`"
  - Otherwise, an **exception is thrown**

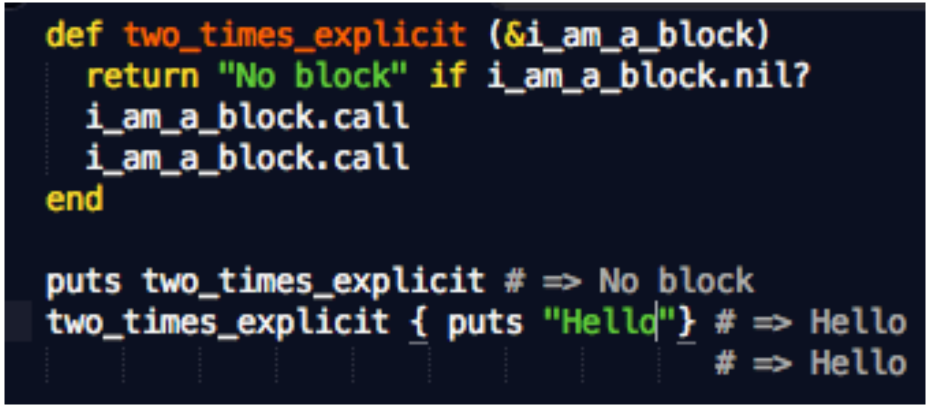


```
def two_times_implicit
  return "No block" unless block_given?
  yield
  yield
end

puts two_times_implicit { print "Hello "} # => Hello
                                          # => Hello
puts two_times_implicit # => No block
```

Explicit

✧ Should check if the block is `nil?`



```
def two_times_explicit (&i_am_a_block)
  return "No block" if i_am_a_block.nil?
  i_am_a_block.call
  i_am_a_block.call
end

puts two_times_explicit # => No block
two_times_explicit { puts "Hello"} # => Hello
                                   # => Hello
```

Summary

- ✧ Blocks are **just code** that you can pass into **methods**
- ✧ When incorporating into your own methods:
  - **Either** use blocks **implicitly**
  - **Or** call them **explicitly**

What's next?

- ✧ Files and Environment Variables