

FUN  
WITH  
BLOCKS



KIRU  
2011  
BLOCKS





Before we begin.

# Blocks

```
say_hello("Hi") do  
  "Welcome to Rorosyd"  
end
```

```
say_hello("Hi") do  
  "Welcome to Rorosyd"  
end
```

```
say_hello("Hi") do  
  place = "Rorosyd"  
  "Welcome to #{place}"  
end
```



```
say_hello("Hi") do  
  place = "Rorosyd"  
  "Welcome to #{place}"  
end
```

```
say_hello("Hi") {  
    place = "Rorosyd"  
    "Welcome to #{place}"  
}
```

```
def say_hello(greeting)
  message = greeting
  message << ", " + yield if block_given?
  message
end
```

```
say_hello("Hi") do
  place = "Rorosyd"
  "Welcome to #{place}"
end
```

```
# => "Hi, Welcome to Rorosyd"
```

```
def say_hello(greeting)
  message = greeting
  message << ", " + yield if block_given?
  message
end
```

```
say_hello("Hi") do
  place = "Rorosyd"
  "Welcome to #{place}"
end
```

```
# => "Hi, Welcome to Rorosyd"
```

```
def say_hello(greeting, &block)
  message = greeting
  message << ", " + block.call if block
  message
end
```

```
say_hello("Hi") do
  place = "Rorosyd"
  "Welcome to #{place}"
end
```

```
# => "Hi, Welcome to Rorosyd"
```

```
def say_hello(greeting, &block)
  message = greeting
  message << ", " + block.call if block
  message
end
```

```
say_hello("Hi") do
  place = "Rorosyd"
  "Welcome to #{place}"
end
```

```
# => "Hi, Welcome to Rorosyd"
```



```
def say_hello(greeting)
  message = greeting
  message << ". "
  message
end
```

```
say_hello("Hi") do
  place = "Rorosyd"
  "Welcome to #{place}"
end
# => "Hi."
```

```
place = "Rorosyd"
```

```
say_hello("Hi") do  
  "Welcome to #{place}"
```

```
end
```

```
# => "Hi, Welcome to Rorosyd"
```

```
def say_hello(greeting, visitors, &block)
  greetings = []
  visitors.each do |name|
    msg = greeting
    msg << ", #{block.call(name)}" if block
    messages << msg
  end
  messages
end
```

```
def say_hello(greeting, visitors, &block)
  ...
  msg << ", #{block.call(name)}" if block
  ...
end

place = "Rorosyd"
say_hello("Hi", ["Kate", "Sam"]) do |visitor|
  "Welcome to #{place}, #{visitor}."
end
# => [ "Welcome to Rorosyd, Kate.",
        "Welcome to Rorosyd, Sam." ]
```

Waitaminate

```
y = 1  
[1,2,3].each do |x|  
  puts (x + y)  
end
```

```
# 2
```

```
# 3
```

```
# 4
```



# Lambdas and Procs

# Lambdas

```
adder = lambda do |x,y|  
  x + y  
end
```

```
adder = lambda do |x,y|  
  x + y  
end
```

```
puts adder.call(1, 2)  
# => 3
```

```
adder = -> (x,y) {  
    x + y  
}
```

```
puts adder.call(1, 2)  
# => 3
```

```
adder = lambda do |x,y|  
  x + y  
end
```

```
puts adder.call(1, 2)  
# => 3
```



```
z = 0
```

```
adder = lambda do |x,y|
```

```
  x + y + z
```

```
end
```

```
puts adder.call(1, 2)
```

```
# => 3
```

```
adder = lambda do |x,y|  
  if x == 0 && y == 0  
    return 0  
  end  
  x + y  
end
```

```
puts adder.call(1, 2)  
# => 3
```

```
adder = -> (x,y) {  
  x + y  
}
```

```
puts adder.call(1,2)  
# => 3
```

```
class Adder  
  def call(x,y)  
    x + y  
  end  
end
```

```
adder = Adder.new  
puts adder.call(1,2)  
# => 3
```

```
works = -> (data) do  
  puts "Fetched #{data}"  
end
```

```
failed = -> (err) do  
  puts "Boom: #{err}"  
end
```

```
works    = -> (data) do  
  puts "Fetched #{data}"  
end
```

```
failed = -> (err) do  
  puts "Boom: #{err}"  
end
```

```
fetch(url, works, failed)  
# Boom: 404
```

```
make    = "Astra"  
year    = "2015"
```

```
car = -> (owner, plate) {  
  Car.new(  
    owner: owner,  
    plate: plate,  
    brand: brand,  
    year:  year,  
  )  
}
```



```
car = -> (owner, plate) {  
  Car.new(...)  
}
```

```
batch = [  
  car.call('Steve', 'NJ99AZ'),  
  car.call('Kate', 'LUCKY1'),  
  car.call('Sam', 'HTTP404'),  
]
```

```
car = -> (owner, plate) {  
    Car.new(...)  
}
```

```
owner = "Rob"
```

```
my_car = -> (plate) {  
    car.call(me, plate)  
}
```

```
batch = [  
    my_car.call('BUG'),  
    my_car.call('BANPROCS'),  
]
```

```
creator = -> (attrs) {  
  Post.with_db(db)  
    .create(attrs)  
}
```

```
post = creator.call(  
  title: "Title",  
)
```

```
class PostCreator  
  def new(db)  
    @db = db  
  end  
  def call(attrs)  
    Post.with_db(@db)  
      .create(attrs)  
  end  
end
```

```
creator = Creator.new(db)  
post = creator.call(  
  title: "Title",  
)
```

```
creator = -> (attrs) {  
  Post.with_db(db)  
    .create(attrs)  
}
```

```
post = creator.call(  
  title: "Title",  
)
```

```
class PostCreator  
  def self.call(db, attrs)  
    Post.with_db(db)  
      .create(attrs)  
  end  
end
```

```
creator = -> (attrs) {  
  Creator.call(db, attrs)  
}
```

```
post = creator.call(  
  title: "Title",  
)
```

# Procs

# Procs

:-)

```
adder = Proc.new do |x,y|  
  x + y  
end
```

```
puts adder.call(1, 2)  
# => 3
```

```
adder = proc do |x,y|  
  x + y  
end
```

```
puts adder.call(1, 2)  
# => 3
```





[www.bigstock.com](https://www.bigstock.com) · 5710741

```
adder = proc do |x,y|  
  x + y  
end
```

```
puts adder.call(1, 2, 7)
```

```
# => 3
```

```
# ...
```

```
# What? What about 7?
```

```
adder = proc do |x,y|  
  x + y  
end
```

```
puts adder.call(1)
```

```
# Kaboom!
```

```
# TypeError: nil can't
```

```
# be coerced into Fixnum
```

```
def write_file(file, operation)  
    f = File.open(file, 'w')  
    result = operation.call(f)  
    puts "Done."  
    f.close  
    result  
end
```

```
def write_file(file, operation)  
  f = File.open(file, 'w')  
  result = operation.call(f)  
  puts "Done."  
  f.close  
  result  
end
```

```
def save_to_file(file, contents)  
  writer = proc do |f|  
    return if contents.nil?  
    f.write(contents)  
  end  
  write_file(file, writer)  
end
```

```
def write_file(file, operation)  
  f = File.open(file, 'w')  
  result = operation.call(f)  
  puts "Done."  
  f.close  
  result  
end
```

```
def save_to_file(file, contents)  
  writer = proc do |f|  
    return if contents.nil?  
    f.write(contents)  
  end  
  write_file(file, writer)  
end
```

```
save_to_file('/tmp/foo', 'HI')  
# Done.  
# => 2
```

```
def write_file(file, operation)  
  f = File.open(file, 'w')  
  result = operation.call(f)  
  puts "Done."  
  f.close  
  result  
end
```

```
def save_to_file(file, contents)  
  writer = proc do |f|  
    return if contents.nil?  
    f.write(contents)  
  end  
  write_file(file, writer)  
end
```

```
save_to_file('/tmp/foo', nil)  
# => nil
```

```
def write_file(file, operation)  
  f = File.open(file, 'w')  
  result = operation.call(f)  
  puts "Done."  
  f.close  
  result  
end
```

```
def save_to_file(file, contents)  
  writer = proc do |f|  
    return if contents.nil?  
    f.write(contents)  
  end  
  write_file(file, writer)  
end
```

```
save_to_file('/tmp/foo', nil)  
# => nil
```



```
def write_file(file, &operation)
  f = File.open(file, 'w')
  result = operation.call(f)
  puts "Done."
  f.close
  result
end
```

```
def save_to_file(file, contents)
  write_file(file) do |f|
    return if contents.nil?
    f.write(contents)
  end
end
```

```
save_to_file('/tmp/foo', nil)
# => nil
```

```
def write_file(file, &operation)
  f = File.open(file, 'w')
  result = operation.call(f)
  puts "Done."
  f.close
  result
end
```

```
def save_to_file(file, contents)
  write_file(file) do |f|
    return if contents.nil?
    f.write(contents)
  end
end
```

```
save_to_file('/tmp/foo', nil)
# => nil
```

**Blocks given to methods act like Procs.**

&

```
[1,2,3].each do |x|  
  puts "Item: #{x}"  
end
```

```
# Item: 1  
# Item: 2  
# Item: 3
```

```
output = proc do |x|  
  puts "Item: #{x}"  
end
```

```
[1, 2, 3].each(&output)
```

```
# Item: 1  
# Item: 2  
# Item: 3
```

# Backflip

```
posts = Post.all
posts.map do |post|
  post.title
end
```

```
# => [
#   "Procs are Fun",
#   "Procs ain't Fun",
# ]
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are Fun",  
#   "Procs ain't Fun",  
# ]
```



```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are Fun",  
#   "Procs ain't Fun"  
# ]
```



&:title

`&:title`

`=> :title.to_proc`

&:title

=> :title.to\_proc

=> proc { |x| x.send(:title) }

```
    &:title  
=> :title.to_proc  
=> proc { |x| x.send(:title) }
```

Which makes our original call:

```
posts.map(&  
  proc { |x| x.send(:title) }  
)
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are Fun",  
#   "Procs ain't Fun",  
# ]
```

Summing up.

# Summing up.

- Blocks are bits of syntax.
- Lambdas and Procs are captured blocks.
- Lambdas can be good "shortcuts" or callbacks; "anonymous functions" in any other language.
- &s let you give blocks back to methods.
- Procs are nuts.
- &:method also nuts.



# Fin.

Rob Howard  
@damncabbage  
robhoward.id.au

