Harness the power of functions to build composable rack applications

Marc Busqué Pérez - @waiting-for-dev

@waiting-for-dev

Harness the power of functions to build composable rack applications

1. Functions 101
2. Rack: design model
3. Web Pipe: design model & main features
4. Web Pipe & Hanami: integrating Web Pipe into a framework

https://nebulab.com/careers

Marc Busqué Pérez

@waiting-for-dev

@waiting_for_dev

marc@lamarciana.com

hanami

solidus

dry-rb

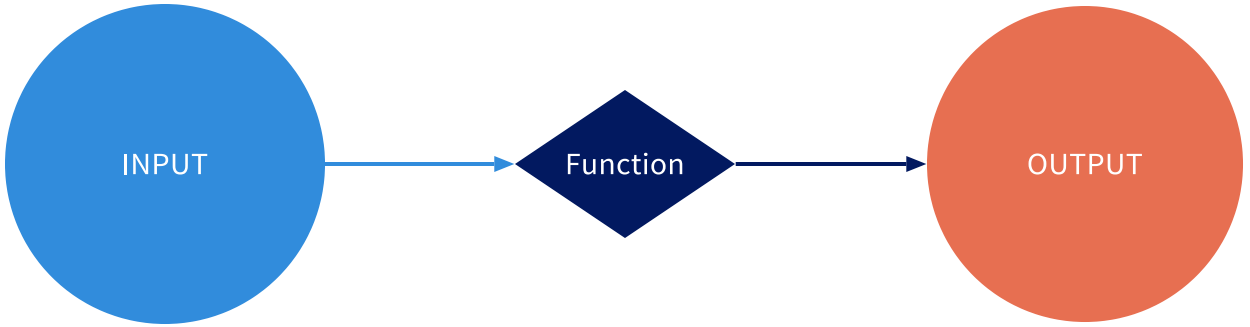
Marc Busqué Pérez - @waiting-for-dev. Harness the power of functions to build composable rack applications.

1

Functions 101

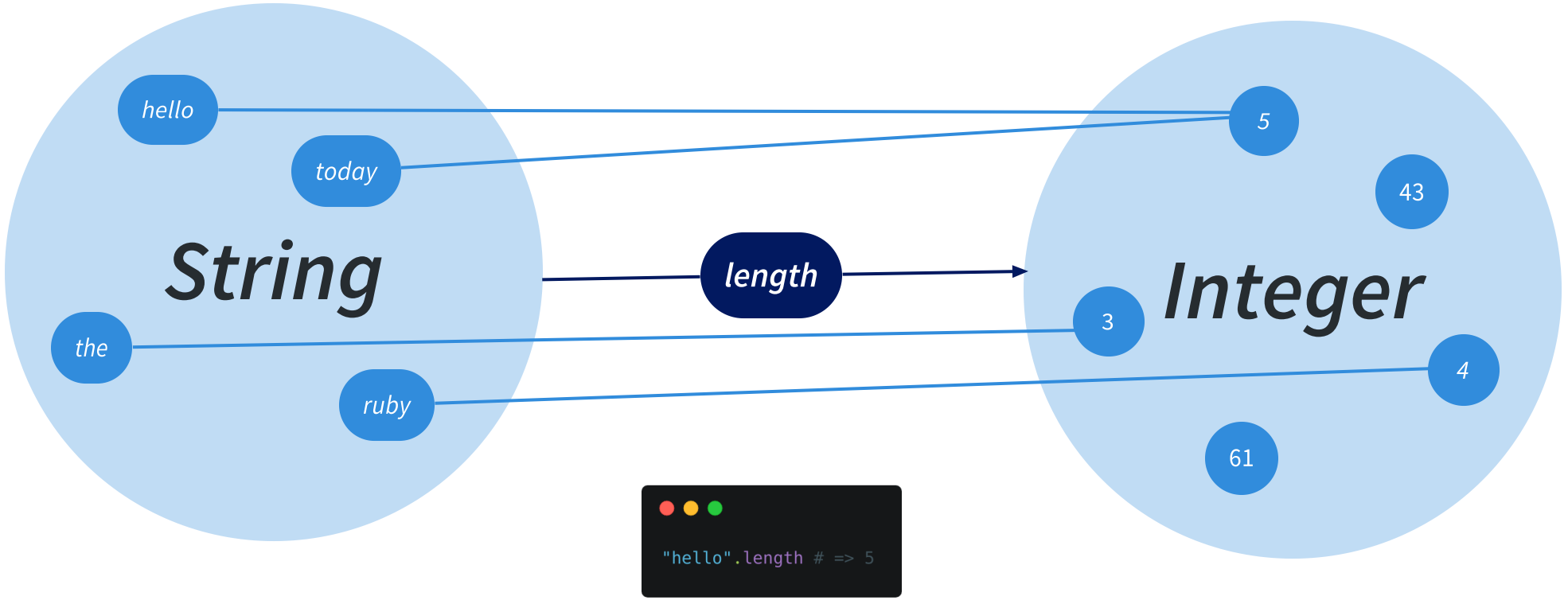
What is a function?

A function is an opaque box transforming input data into something else

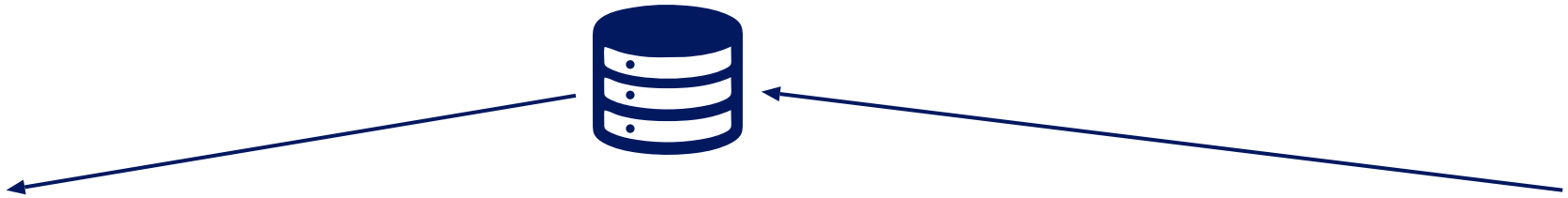


Ok, but, what is a function?

A function is a binary relationship between two sets, where each element of the input set is matched precisely to one element of the output set.



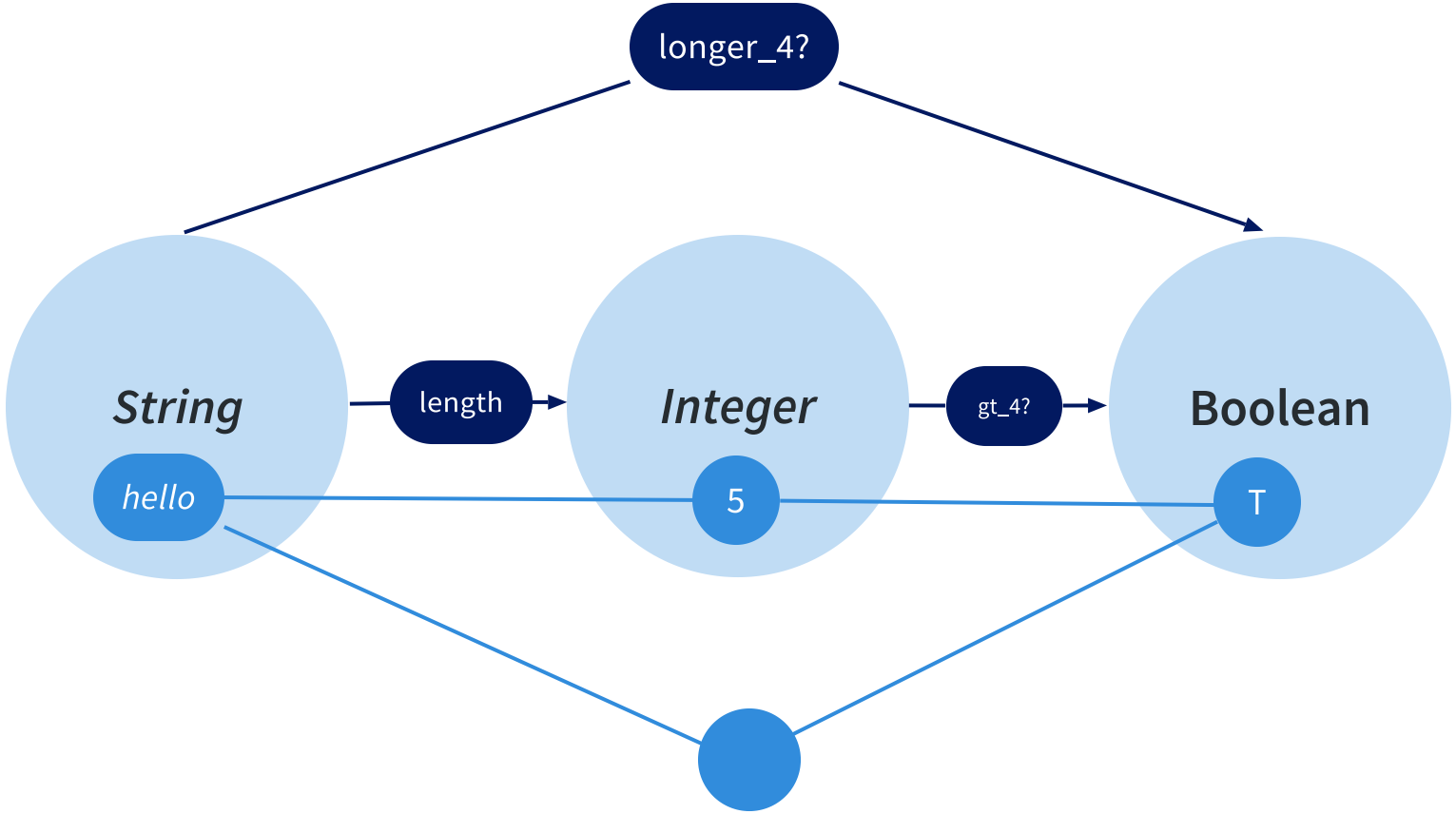
Hidden pre-conditions and side effects



	hidden pre-condition	function	result	side effect
1		*stock_for_product(id: 1)*	10	visits_counter += 1
2	added 10 items	*stock_for_product(id: 1)*	20	visits_counter += 1
3	bought 2 items	*stock_for_product(id: 1)*	18	visits_counter += 1

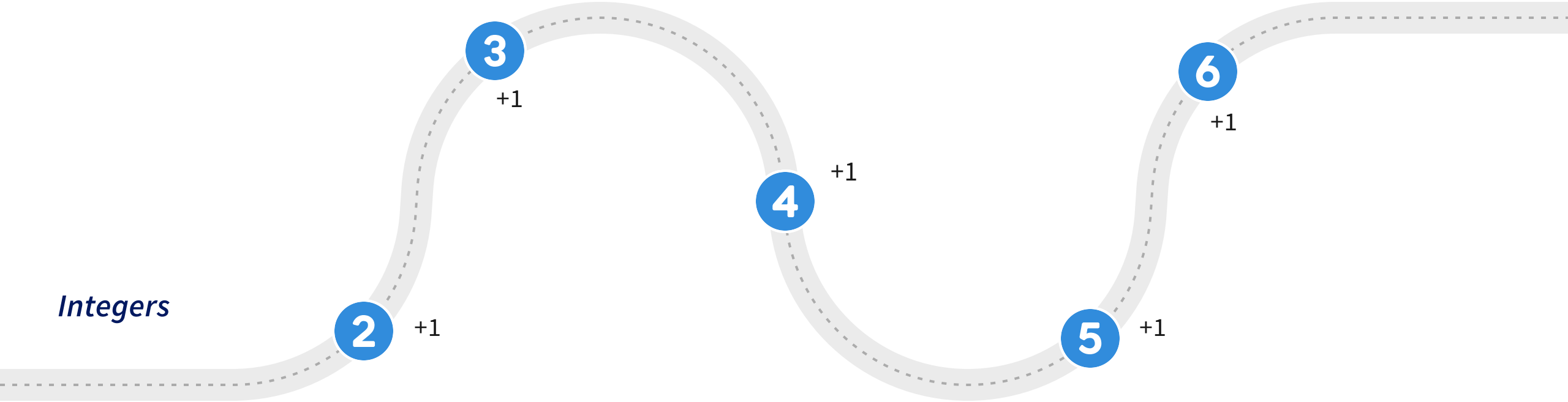
Function composition

Two functions compose when the output of the first belongs to the same set that the input of the second



Self-composition

Functions with the same type for input and output, self-compose an infinite number of times

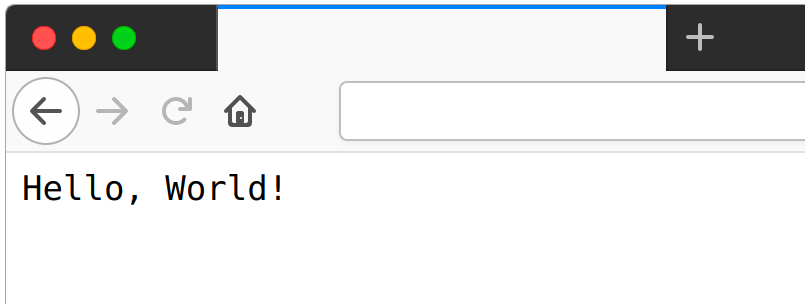


2

Rack

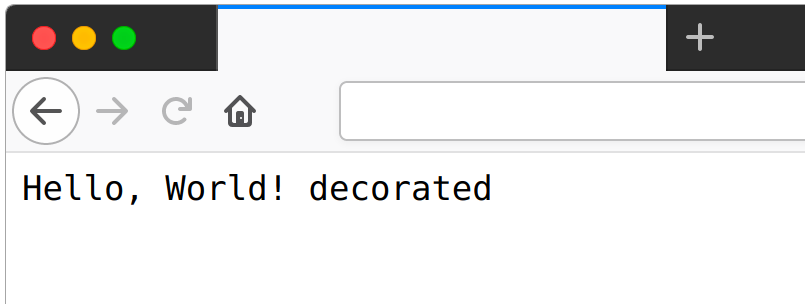
Rack's “Hello, World!”

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

run hello_world
```



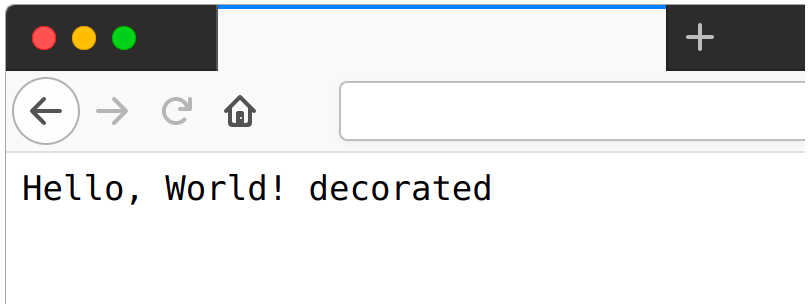
Rack's “Hello, World!” decorated

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

hello_world_decorated = lambda do |env|
  env['decorated'] = 'decorated'

  status, headers, body = hello_world.(env)

  [status, headers, ["#{body[0]} #{env['decorated']}"]]
end

run hello_world_decorated
```



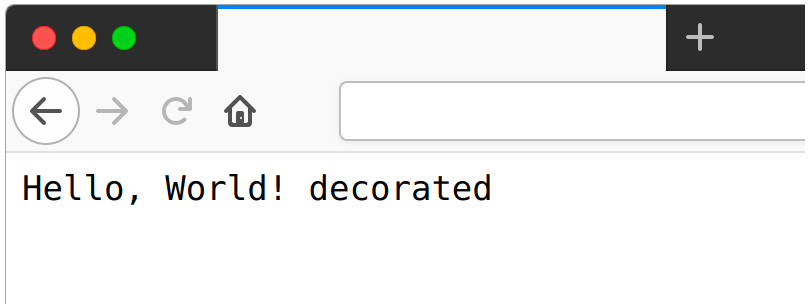
Rack's “Hello, World!” decorated (with curry)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

app_decorated = lambda do |app, env|
  env['decorated'] = 'decorated'

  status, headers, body = app.call(env)

  [status, headers, ["#{body[0]} #{env['decorated']}"]]
end

run app_decorated.curry.(hello_world)
```



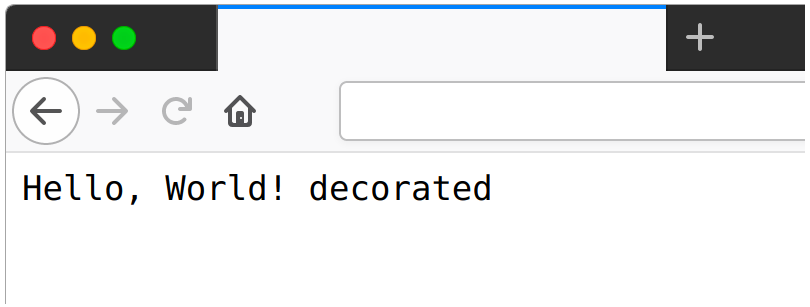
Rack's “Hello, World!” decorated (with middleware)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

class DecoratorMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    env['decorated'] = 'decorated'

    status, headers, body = @app.call(env)

    [status, headers, ["#{body[0]} #{env['decorated']}"]]
  end
end

run DecoratorMiddleware.new(hello_world)
```



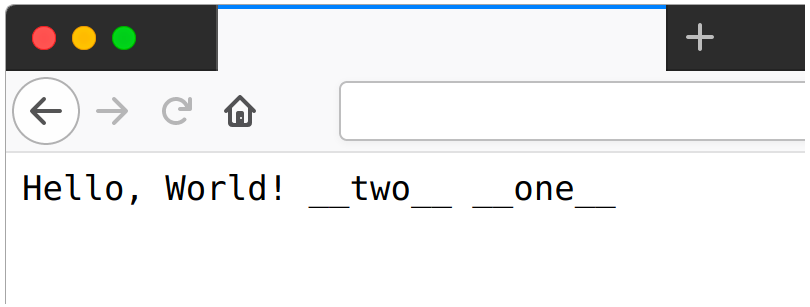
Rack's “Hello, World!” decorated (with middleware's DSL)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

class DecoratorMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    env['decorated'] = 'decorated'

    status, headers, body = @app.call(env)

    [status, headers, ["#{body[0]} #{env['decorated']}"]]
  end
end

use DecoratorMiddleware
run hello_world
```

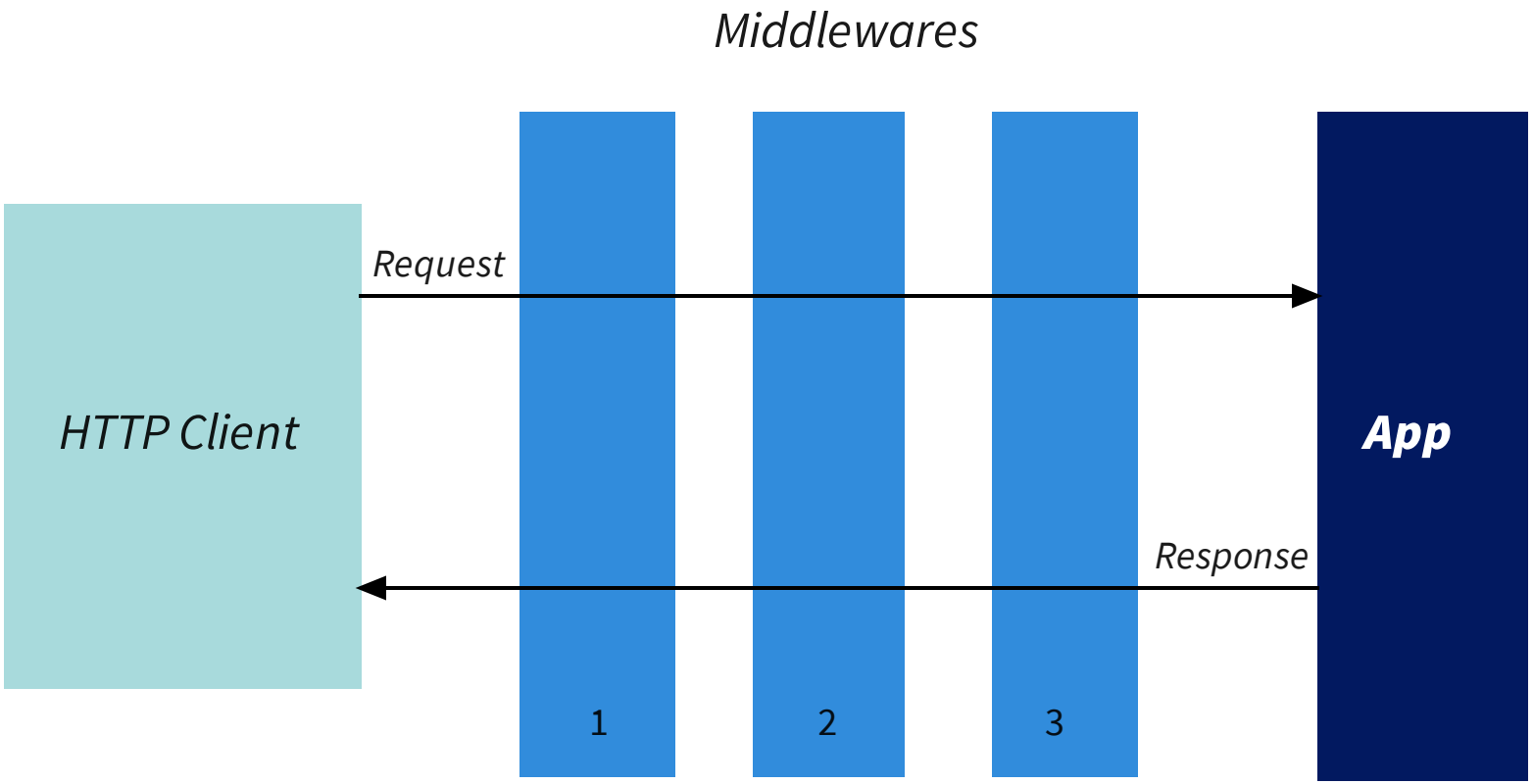


Rack's “Hello, World!” decorated (with middlewares stack)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

hello_world = lambda do |_env|
  [200, { 'Content-Type' => 'text/plain' }, ['Hello, World!']]
end

class DecoratorMiddleware
  def initialize(app, decorated_text)
    @app = app
    @decorated_text = decorated_text
  end

  def call(env)
    status, headers, body = @app.call(env)

    [status, headers, ["#{body[0]} #{@decorated_text}"]]
  end
end

use DecoratorMiddleware, '__one__'
use DecoratorMiddleware, '__two__'
run hello_world
```



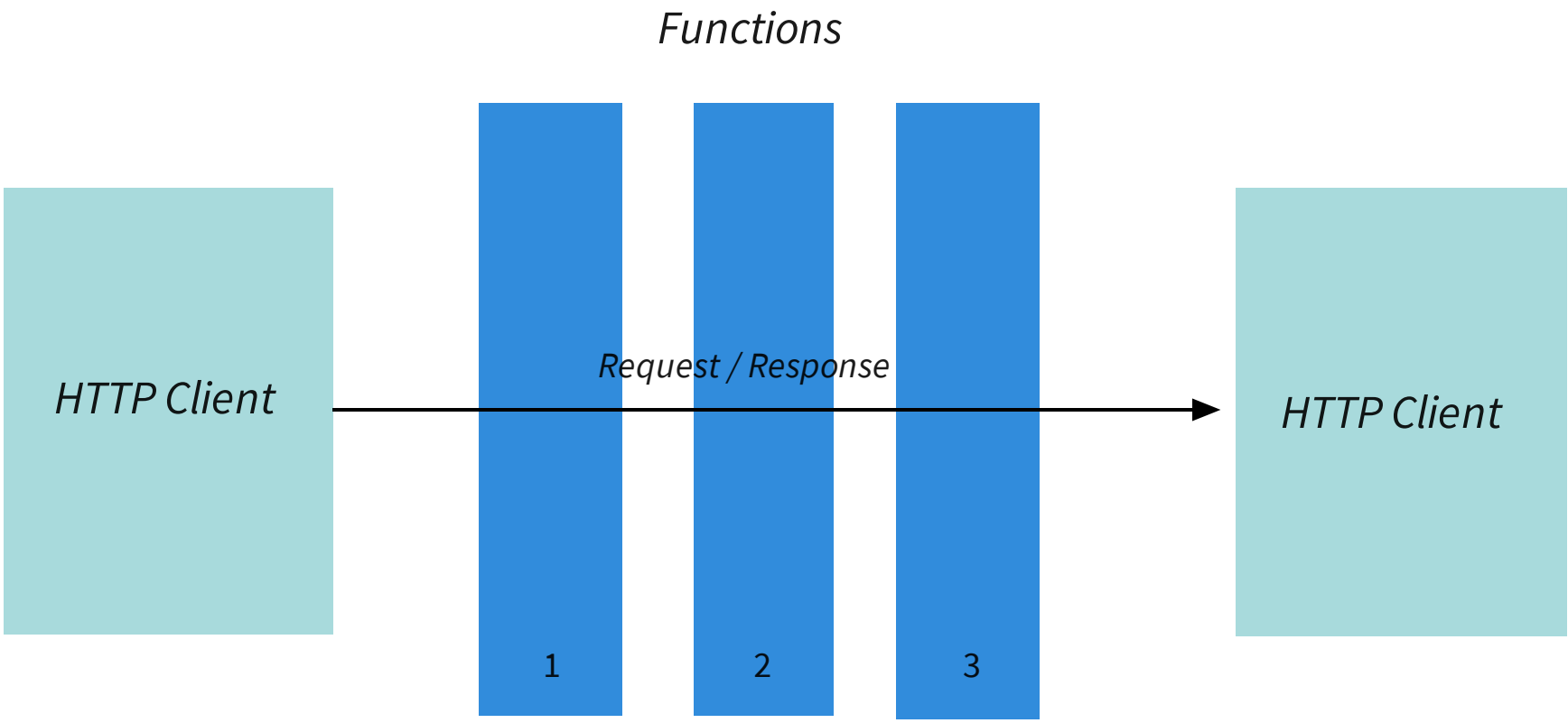
Rack's two-way pipe



3

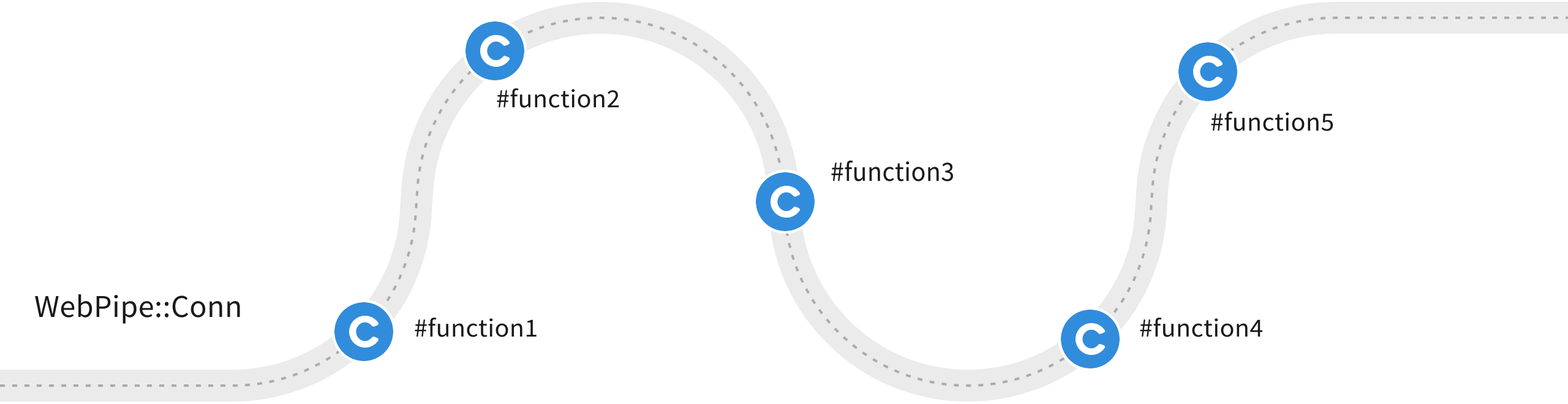
web_pipe

web_pipe's one-way pipe



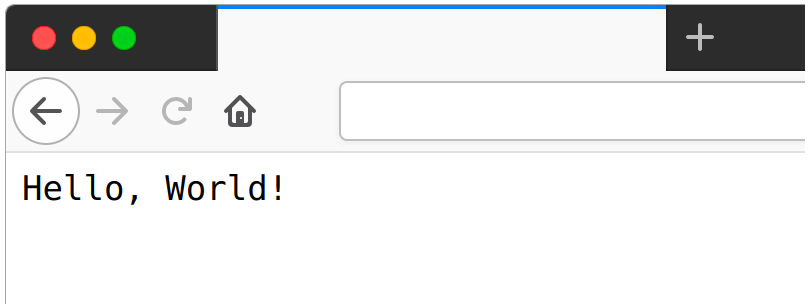
web_pipe's self-composition

WebPipe::Conn is a struct containing both the request and the response



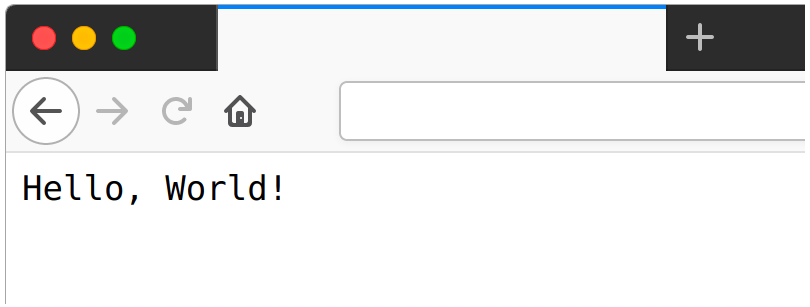
web_pipe's “Hello, World!”

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'

class HelloWorld
  include WebPipe

  plug :content_type do |conn|
    conn.add_response_header('Content-Type', 'text/plain')
  end

  plug :render do |conn|
    conn.set_response_body('Hello, World!')
  end
end

run HelloWorld.new
```



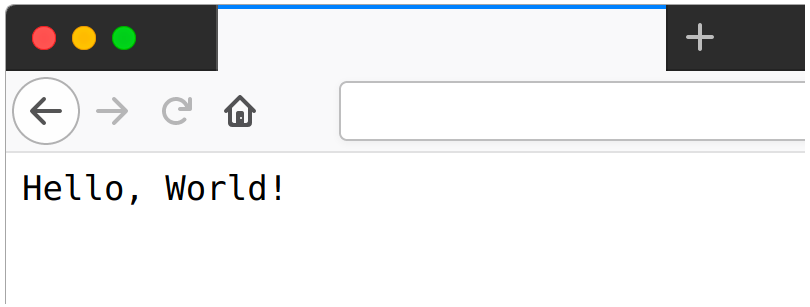
web_pipe's “Hello, World!” (methods)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'

class HelloWorld
  include WebPipe

  plug :content_type

  plug :render

  private

  def content_type(conn)
    conn.add_response_header('Content-Type', 'text/plain')
  end

  def render(conn)
    conn.set_response_body('Hello, World!')
  end
end

run HelloWorld.new
```



web_pipe's “Hello, World!” (dry-container)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'
require 'dry-container'

Container = Dry::Container.new
Container.register('plugs.content_type', lambda do |conn|
  conn.add_response_header('Content-Type', 'text/plain')
end)
Container.register('plugs.render', lambda do |conn|
  conn.set_response_body('Hello, World!')
end)

class HelloWorld
  include WebPipe.(container: Container)

  plug :content_type, 'plugs.content_type'

  plug :render, 'plugs.render'
end

run HelloWorld.new
```



WebPipe::Conn methods

Request methods

- **#scheme** *:http* or *:https*
- **#request_method** *:get*, *:post*...
- **#host** *'www.example.org'*, ...
- **#ip** *'192.168.1.1'*, ...
- **#port**: *80*, *443*, ...
- **#script_name** *'index.rb'*, ...
- **#path_info** *'/foor/bar'*, ...
- **#query_string** '*foo=bar&bar=foo*', ...
- **#request_body** *'{ id: 1 }'*, ...
- **#request_headers** *{ 'Accept-Charset' => 'utf8' }* , ...
- **#env** Rack's env hash.
- **#request** Rack::Request instance.

Response methods

- **#set_status(code)**
- **#set_response_body(body)**
- **#set_response_headers(headers)**
- **#add_response_header(key, value)**

Pipeline methods

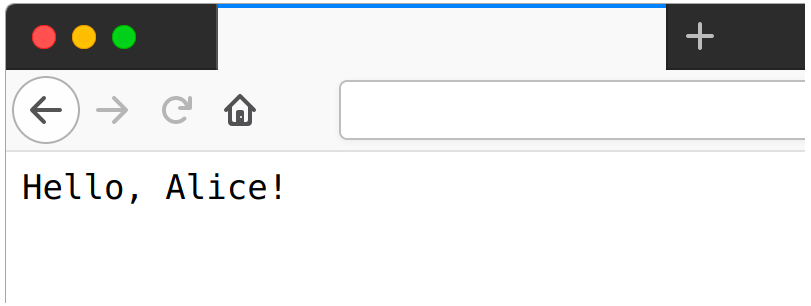
- **#add(key, value)**
- **#fetch(key)**

web_pipe's extensions

- container
- cookies
- flash
- dry_schema
- hanami_view
- not_found
- params
- rails
- redirect
- router_params
- session
- url

web_pipe's “Hello, #{name}!”

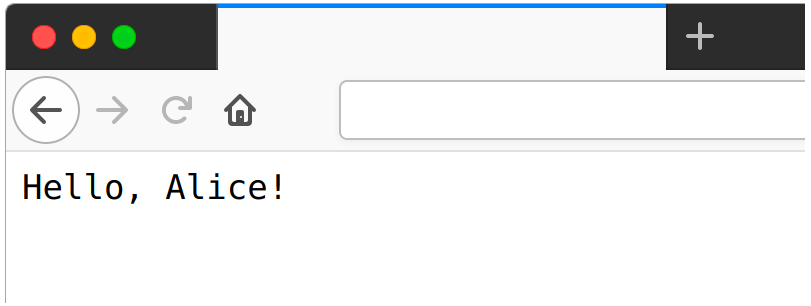
```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'
require 'web_pipe/plugs/content_type'

WebPipe.load_extensions(:params)

class HelloName
  include WebPipe

  plug :content_type, WebPipe::Plugs::ContentType.('text/plain')
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    conn.add(:name, conn.params['name'])
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch(:name)}!")
  end
end

run HelloName.new
```



http://localhost:9292/?name=Alice

web_pipe's “Hello, #{name}!” (symbolized keys)

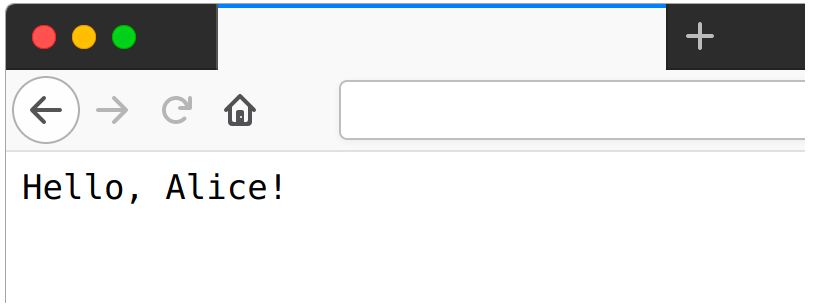
```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'
require 'web_pipe/plugs/config'
require 'web_pipe/plugs/content_type'

WebPipe.load_extensions(:params)

class HelloName
  include WebPipe

  plug :content_type, WebPipe::Plugs::ContentType.('text/plain')
  plug :config, WebPipe::Plugs::Config.(
    param_transformations: [:deep_symbolize_keys]
  )
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    conn.add(:name, conn.params[:name])
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch(:name)}!")
  end
end

run HelloName.new
```



http://localhost:9292/?name=Alice

web_pipe's “Hello, #{name}!” (composed)

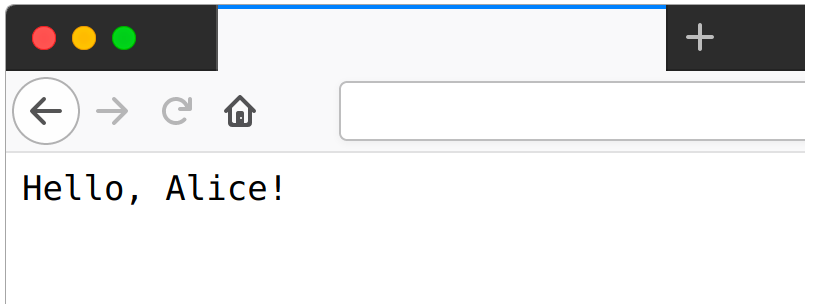
```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require 'web_pipe'
require 'web_pipe/plugs/config'
require 'web_pipe/plugs/content_type'

WebPipe.load_extensions(:params)

class Base
  include WebPipe

  plug :content_type, WebPipe::Plugs::ContentType.('text/plain')
  plug :config, WebPipe::Plugs::Config.(
    param_transformations: [:deep_symbolize_keys]
  )
end

class HelloName
  include WebPipe

  plug :base, Base.new
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    conn.add(:name, conn.params[:name])
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch(:name)}!")
  end
end

run HelloName.new
```



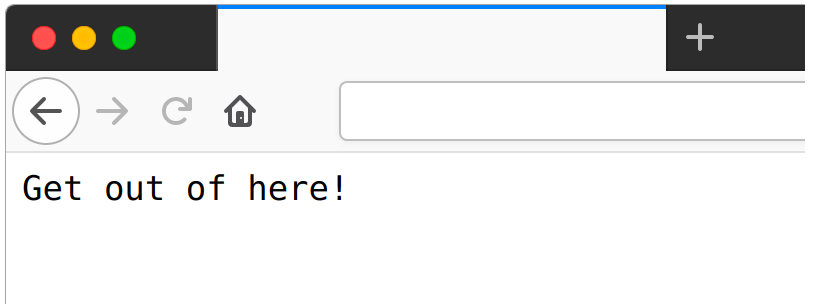
http://localhost:9292/?name=Alice

web_pipe's “Hello, #{name}!” (authorized)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require_relative 'base'

DB = {
  users: [
    Struct.new(:name).new('Alice'),
    Struct.new(:name).new('Joe')
  ].freeze
}.freeze

class HelloName
  include WebPipe

  plug :base, Base.new
  plug :fetch_name
  plug :authorize
  plug :render

  private

  def fetch_name(conn)
    conn.add(:name, conn.params[:name])
  end

  def authorize(conn)
    if DB[:users].map(&:name).include?(conn.fetch(:name))
      conn
    else
      conn
        .set_status(401)
        .set_response_body('Get out of here!')
        .halt
    end
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch(:name)}!")
  end
end

run HelloName.new
```



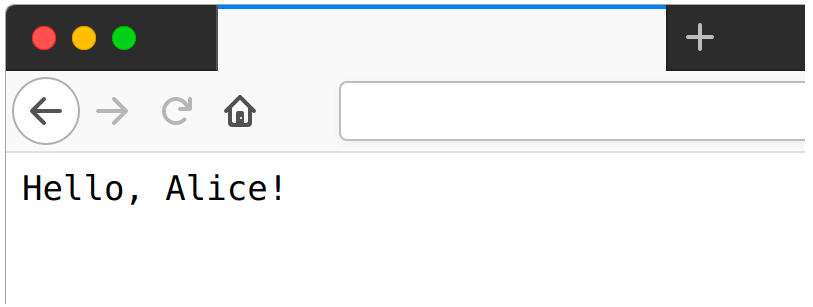
http://localhost:9292/?name=Alice



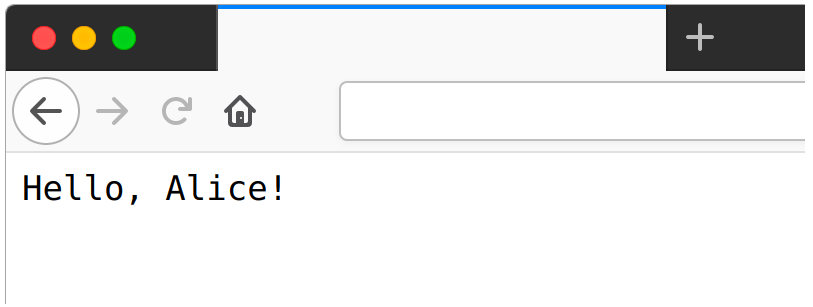
http://localhost:9292/?name=Donald

web_pipe's “Hello, #{name}!” (with session)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require_relative 'base'

WebPipe.load_extensions(:session)

class HelloName
  include WebPipe

  use :session, Rack::Session::Cookie, secret: 'dont_tell_anybody'
  plug :base, Base.new
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    if conn.params[:name]
      conn.add_session('name', conn.params[:name])
    else
      conn
    end
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch_session('name')}!")
  end
end

run HelloName.new
```



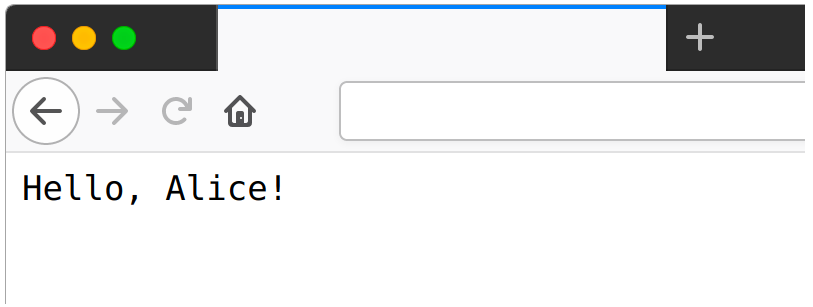
1. http://localhost:9292/?name=Alice



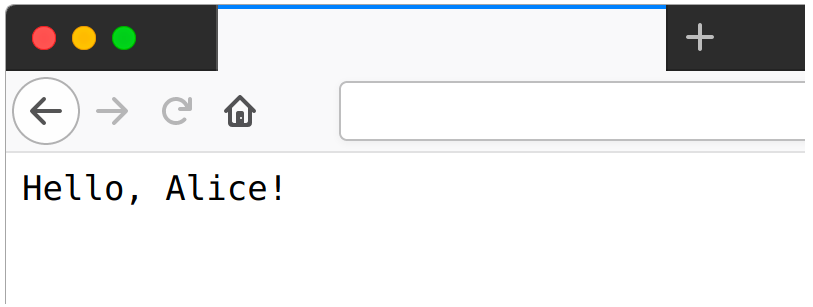
2. http://localhost:9292

web_pipe's “Hello, #{name}!” (with session - use)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require_relative 'base'

WebPipe.load_extensions(:session)

class HelloName
  include WebPipe

  use :base, Base.new
  plug :base, Base.new
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    if conn.params[:name]
      conn.add_session('name', conn.params[:name])
    else
      conn
    end
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch_session('name')}!")
  end
end

run HelloName.new
```



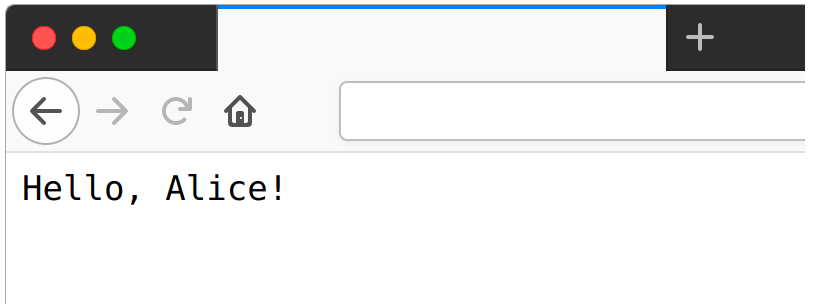
1. http://localhost:9292/?name=Alice



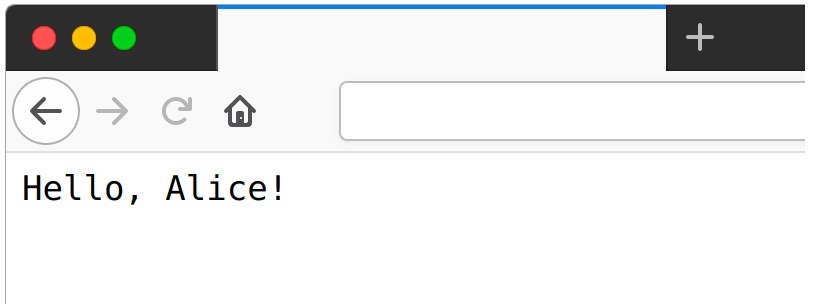
2. http://localhost:9292

web_pipe's “Hello, #{name}!” (with session - compose)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require_relative 'base'

WebPipe.load_extensions(:session)

class HelloName
  include WebPipe

  compose :base, Base.new
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    if conn.params[:name]
      conn.add_session('name', conn.params[:name])
    else
      conn
    end
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch_session('name')}!")
  end
end

run HelloName.new
```



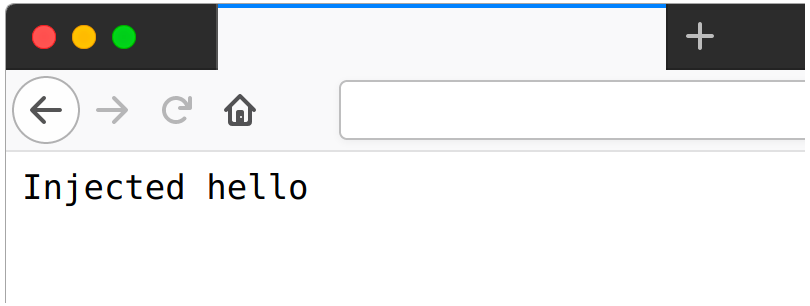
1. http://localhost:9292/?name=Alice



2. http://localhost:9292

web_pipe's “Hello, #{name}!” (injected)

```ruby
# frozen_string_literal: true

# config.ru
# Execute it with `rackup` and go to http://localhost:9292

require_relative 'base'

class HelloName
  include WebPipe

  plug :base, Base.new
  plug :fetch_name
  plug :render

  private

  def fetch_name(conn)
    conn.add(:name, conn.params[:name])
  end

  def render(conn)
    conn.set_response_body("Hello, #{conn.fetch(:name)}!")
  end
end

run HelloName.new(
  plugs: {
    render: ->(conn) { conn.set_response_body('Injected hello') }
  }
)
```



http://localhost:9292/?name=Alice

s/MVC/C/web_pipe

Rails: routing to a Rack (web_pipe) app

```ruby
# config/routes.rb
get '/my_route', to: MyRoute.new

# app/controllers/my_route.rb
class MyRoute
  include WebPipe

  plug :set_response_body

  private

  def set_response_body(conn)
    conn.set_response_body('Hello, World!')
  end
end
```

Rails: using the extension

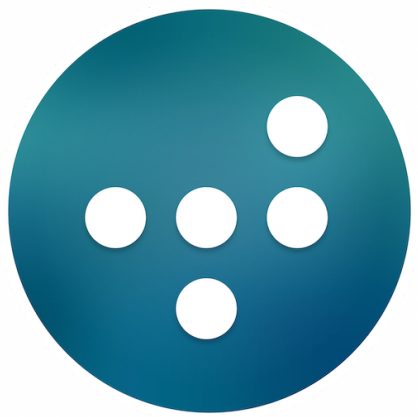
```ruby
# config/routes.rb
get '/articles', to: ArticlesIndex.new

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  # By default uses the layout in `layouts/application`
end

# app/controllers/articles_index.rb
require 'web_pipe/plugs/config'

WebPipe.load_extensions(:rails) # You can put it in an initializer

class ArticlesIndex
  include WebPipe

  plug :config, WebPipe::Plugs::Config.(
    rails_controller: ApplicationController
  )

  def render(conn)
    conn.render(
      template: 'articles/index',
      assigns: { articles: Article.all }
    )
  end
end
```

4

Hanami 2

Hanami 2

https://hanamirb.org



https://dry-rb.org

https://rom-rb.org

https://github.com/waiting-for-dev/hanami_2_web_pipe_todo_app

Hanami + web_pipe: TodoManager::Routes

```ruby
# frozen_string_literal: true

require "hanami/application/routes"

module TodoManager
  class Routes < Hanami::Application::Routes
    define do
      slice :main, at: "/" do
        root to: "todo.index"

        get "/todos/new", to: "todo.new"
        get "/todos/:id", to: "todo.show"
        get "/todos/:id/edit", to: "todo.edit"
        patch "/todos/:id", to: "todo.update"
        post "/todos", to: "todo.create"
        delete "/todos/:id", to: "todo.delete"
      end
    end
  end
end
```

Hanami + web_pipe: Main::Actions::Todo::Update

```ruby
# frozen_string_literal: true

require "dry/monads"
require "web_pipe"

module Main
  module Actions
    module Todo
      class Update
        include Dry::Monads[:result]
        include WebPipe
        include Deps[
          "application.transactions.update_todo",
          "application.repositories.todo_repo"
        ]

        compose :main, Main::Action.new

        plug :fetch_todo
        plug :transaction
        plug :render
```

```ruby
        private

        def fetch_todo(conn)
          Action::FETCH_ENTITY_BY_ID.(todo_repo, :todo).(conn)
        end

        def transaction(conn)
          todo = conn.fetch(:todo)

          case update_todo.(todo, conn.params[:todo])
          in Success
            conn
          in Failure[result]
            conn
              .view('views.todo.edit', id: todo.id, object: result)
              .halt
          end
        end

        def render(conn)
          conn
            .add_flash(:success, 'Item updated successfully')
            .redirect('/')
        end
      end
    end
  end
end
```

Hanami + web_pipe: Main::Action

```ruby
# auto_register: false
# frozen_string_literal: true

require "rack/csrf"
require "rack-flash"
require "rack/session/cookie"
require "todo_manager/action"
require "web_pipe"
require "web_pipe/plugs/config"
require "web_pipe/plugs/content_type"

module Main
  class Action
    include WebPipe

    FETCH_ENTITY_BY_ID = lambda do |repo, name, id: :id|
      lambda do |conn|
        entity = repo.by_id(conn.params[id])
        entity ? conn.add(name, entity) : conn.not_found
      end
    end.freeze

    compose :todo_manager, TodoManager::Action.new

    use :session, Rack::Session::Cookie,
        key: "todo_manager.session",
        secret: TodoManager::Application.settings["session_secret"],
        expire_after: 60 * 60 * 24 * 365 # 1 year
    use :csrf_protection, Rack::Csrf, raise: true
    use :flash, Rack::Flash

    plug :html, WebPipe::Plugs::ContentType.('text/html')
    plug :config, WebPipe::Plugs::Config.(
      container: TodoManager::Application.slices[:main],
      view_context: lambda do |conn|
        {
          current_path: conn.full_path,
          csrf_token: Rack::Csrf.token(conn.env),
          csrf_field: Rack::Csrf.field,
          flash: conn.flash,
          flash_success?: !conn.flash[:success].nil?
        }
      end
    )
  end
end
```

Hanami + web_pipe: Main::Actions::Todo::Update

```ruby
# frozen_string_literal: true

require "dry/monads"
require "web_pipe"

module Main
  module Actions
    module Todo
      class Update
        include Dry::Monads[:result]
        include WebPipe
        include Deps[
          "application.transactions.update_todo",
          "application.repositories.todo_repo"
        ]

        compose :main, Main::Action.new

        plug :fetch_todo
        plug :transaction
        plug :render
```

```ruby
        private

        def fetch_todo(conn)
          Action::FETCH_ENTITY_BY_ID.(todo_repo, :todo).(conn)
        end

        def transaction(conn)
          todo = conn.fetch(:todo)

          case update_todo.(todo, conn.params[:todo])
          in Success
            conn
          in Failure[result]
            conn
              .view('views.todo.edit', id: todo.id, object: result)
              .halt
          end
        end

        def render(conn)
          conn
            .add_flash(:success, 'Item updated successfully')
            .redirect('/')
        end
      end
    end
  end
end
```

Hanami + web_pipe: TodoManager::Transactions::UpdateTodo

```ruby
# frozen_string_literal: true

require "dry/monads"
require "dry/monads/do"

module TodoManager
  module Transactions
    class UpdateTodo
      include Dry::Monads[:result]
      include Dry::Monads::Do.for(:call)
      include Deps[
        "validation.update_todo_contract",
        "repositories.todo_repo"
      ]

      def call(todo, input)
        attributes = yield validate(input)
        update_todo(todo, attributes)
      end

      private

      def validate(input)
        result = update_todo_contract.(input)
        if result.success?
          Success(result.to_h)
        else
          Failure(result)
        end
      end

      def update_todo(todo, attributes)
        Success(
          todo_repo.update(todo.id, attributes)
        )
      end
    end
  end
end
```

Hanami + web_pipe: TodoManager::Transactions::UpdateTodo

```ruby
# frozen_string_literal: true

module TodoManager
  module Validation
    class UpdateTodoContract < Contract
      schema do
        optional(:title).filled(:string)
        optional(:description).filled(:string)
      end
    end
  end
end
```

Hanami + web_pipe: TodoManager::Repositories::TodoRepo

```ruby
# frozen_string_literal: true

module TodoManager
  module Repositories
    class TodoRepo < Repository[:todos]
      commands :create,
               update: :by_pk,
               delete: :by_pk

      def all
        todos.to_a
      end

      def by_id(id)
        todos.by_pk(id).one
      end
    end
  end
end
```

Hanami + web_pipe: Main::Actions::Todo::Update

```ruby
# frozen_string_literal: true

require "dry/monads"
require "web_pipe"

module Main
  module Actions
    module Todo
      class Update
        include Dry::Monads[:result]
        include WebPipe
        include Deps[
          "application.transactions.update_todo",
          "application.repositories.todo_repo"
        ]

        compose :main, Main::Action.new

        plug :fetch_todo
        plug :transaction
        plug :render
```

```ruby
        private

        def fetch_todo(conn)
          Action::FETCH_ENTITY_BY_ID.(todo_repo, :todo).(conn)
        end

        def transaction(conn)
          todo = conn.fetch(:todo)

          case update_todo.(todo, conn.params[:todo])
          in Success
            conn
          in Failure[result]
            conn
              .view('views.todo.edit', id: todo.id, object: result)
              .halt
          end
        end

        def render(conn)
          conn
            .add_flash(:success, 'Item updated successfully')
            .redirect('/')
        end
      end
    end
  end
end
```

THANKS!!!!

https://github.com/waiting-for-dev/rubyconf21_web_pipe

Marc Busqué Pérez - @waiting-for-dev