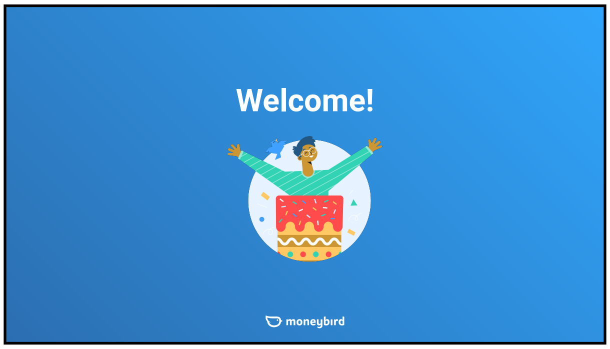
Ruby (on Rails) workshop

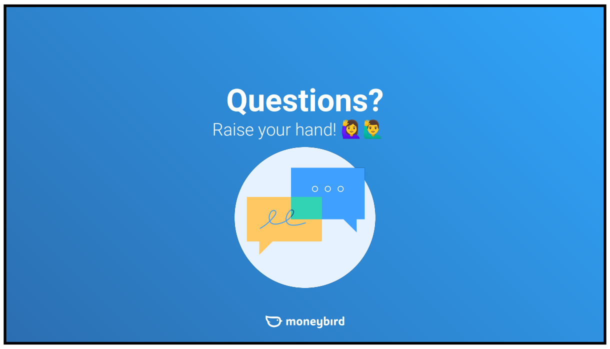
(psst! Does rails -v work in your terminal?)

moneybird



Welcome!

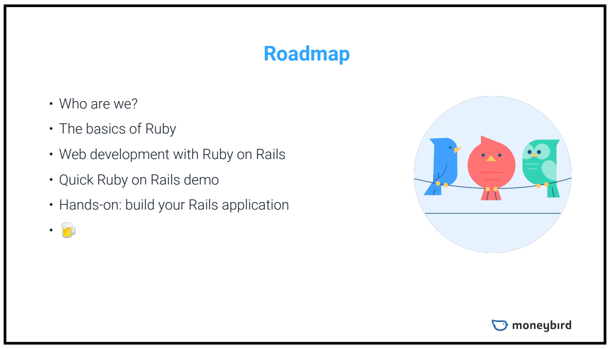
moneybird



Questions?

Raise your hand! 🙋‍♀️🙋‍♂️

moneybird



Roadmap

- Who are we?
- The basics of Ruby
- Web development with Ruby on Rails
- Quick Ruby on Rails demo
- Hands-on: build your Rails application
- 🍺

moneybird

Jeroen Weeink

- Graduated from Saxion in 2007
- Working with Ruby since 2012

moneybird

Yvonne Nieuwerth

- Graduated from Saxion in 2014
- At Moneybird since graduation

moneybird

The basics of Ruby

- Everything is an object
- Dynamic typed
- Quite similar to Python in many ways

moneybird

The basics of Ruby

```
10.times do
  puts “Hello world!”
end
```

moneybird

Data types: numbers

- Integers: `10, 20`
- Floats: `1.55`
- Bigdecimal: `BigDecimal("1.55")`

moneybird

Data types: numbers

- These all do what you'd expect in Python:
  - `2 + 3    # 5`
  - `3.0 / 2  # 1.5`
  - `10 % 3   # 1`
  - `2 ** 3   # 8`

moneybird

Data types: numbers

- But Ruby defines methods directly on these numbers:
  - `5.683.round(2) # 5.68`
  - `-10.abs        # 10`
  - `99.next        # 100`
- Note the missing brackets if no arguments are given!

moneybird

Data types: strings

- Work pretty much the same as in Python:
  - `"Hello"`
  - `'Hello'`
  - `"Money" + "bird"   # "Moneybird"`
  - `"Moneybird".length # 9`

moneybird

Data types: true, false and nil

- `True` is `true`, `False` is `false`
- `None` is `nil`
- `nil` and `false` are falsy, everything else is truthy
- Empty collections and zero numbers are also truthy!

- `puts “truthy” if 100 == 100`
- `puts “truthy” if 0`
- `puts “falsy” if nil`

Variables

- Don’t have to be declared and can be re-assigned
  - `number = 10`
  - `number = number * 2`
  - `number = 30`
  - `number = “Hello”`

Collections: Array

- Similar to a list in Python
- Values can be added, removed and replaced
- The order of values in an Array is fixed
- Ruby returns `nil` if the index is out of range rather than raising an error!

Collections: Array

```
candidates = [“Alice”, “Bob”]
candidates[0]  # “Alice”
candidates[-1] # “Bob”
candidates[2]  # nil, not an IndexError!
candidates.push(“Cristine”).push(“Dylan”) # Chaining methods
```

Collections: Hash

- Similar to a dictionary in Python
- Notation slightly different
- Each key is unique
- Keys can be any other type
- The order is implementation-specific

moneybird

Collections: Hash

```
word_to_number = {
  “one”: 1,
  “ten”: 10,
  “hundred”: 100
}
word_to_number[“ten”] # 10
```

➡

```
word_to_number = {
  “one” => 1,
  “ten” => 10,
  “hundred” => 100
}
word_to_number[“ten”] # 10
```

moneybird

Collections: Hash

- Gotcha: Hashes with Symbols
- Yet another notation
- Symbol is different from a String!
- Both accepted by most places in Rails

```
word_to_number = {
  one: 1,
  ten: 10,
  hundred: 100
}
word_to_number[:ten]   # 10
word_to_number[“ten”] # nil
```

moneybird

Control flow: if/else

- `elif` is `elsif` in Ruby
- Always ends with `end`

```
if number < 5
  puts “Small number”
elsif number < 10
  puts “Medium number”
else
  puts “Large number”
end
```

moneybird

Iterating over collections

- Calling `each` on a collection is the default way to iterate
- Similar to using `for…in` loop in Python
- The `do…end` block is executed for each value
- Blocks are comparable to `lambda` in Python

moneybird

Iterating over collections

```
numbers = [1, 2, 3]

for num in numbers:
  print(num)
```

➡

```
numbers = [1, 2, 3]

numbers.each do |num|
  puts num
end
```

moneybird

Iterating over collections

- Use map to create a new collection with transformed values

```
numbers = [1, 2, 3]

list(map(
  lambda num: num * 2,
  numbers
))
# [2, 4, 6]
```

➡

```
numbers = [1, 2, 3]

numbers.map do |num|
  num * 2
end
# [2, 4, 6]
```

moneybird

Iterating over collections

- Instead of do … end, { … } can also be used to write blocks
- Useful for single-line blocks
- Not to be confused with Hashes! 😅

```
numbers = [1, 2, 3]

numbers.map do |num|
  num * 2
end
# [2, 4, 6]
```

➡

```
numbers = [1, 2, 3]

numbers.map { |num| num * 2 }
# [2, 4, 6]
```

moneybird

Methods

- Also starts with `def` keyword, but doesn't end with a `:`
- Parenthesis are optional, but better to use them with parameters
- Last statement is implicitly returned
- Method ends with `end`
- Since everything is an object, functions don't exist in Ruby

moneybird

Methods

```
def double_sum(a, b, c):
  sum = a + b + c

  return sum * 2
```

➡

```
def double_sum(a, b, c)
  sum = a + b + c

  sum * 2
end
```

moneybird

Methods

- Parenthesis are optional, but good practice to use them with arguments
- Also good practice to leave them out if no arguments are given

```
"Moneybird".index("e") # 3
"Moneybird".length     # 9
```

moneybird

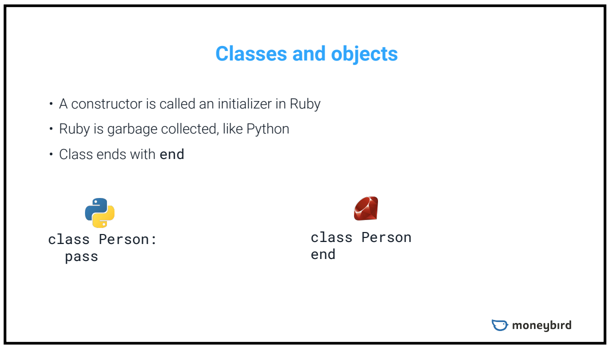
Methods

- Some additional conventions:
  - methods with `?` return `true` or `false`
  - Methods with `!` change something
- Ruby doesn't check this

```
[].none? # true

name = "Moneybird"
name.sub!("Money", "")
name # "bird"
```

moneybird



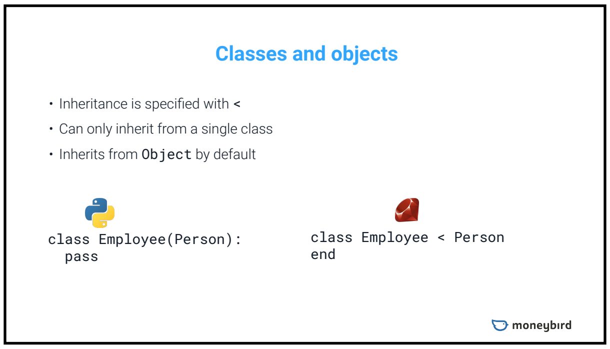
Classes and objects

- A constructor is called an initializer in Ruby
- Ruby is garbage collected, like Python
- Class ends with `end`

```python
class Person:
  pass
```

```ruby
class Person
end
```

moneybird



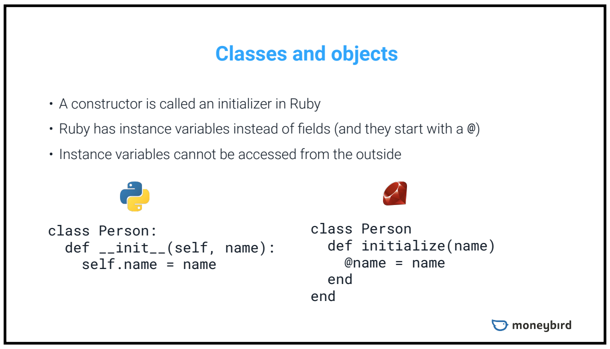
Classes and objects

- Inheritance is specified with `<`
- Can only inherit from a single class
- Inherits from `Object` by default

```python
class Employee(Person):
  pass
```

```ruby
class Employee < Person
end
```

moneybird



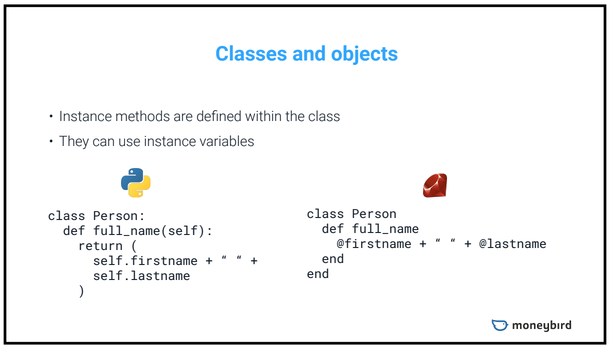
Classes and objects

- A constructor is called an initializer in Ruby
- Ruby has instance variables instead of fields (and they start with a `@`)
- Instance variables cannot be accessed from the outside

```python
class Person:
  def __init__(self, name):
    self.name = name
```

```ruby
class Person
  def initialize(name)
    @name = name
  end
end
```

moneybird



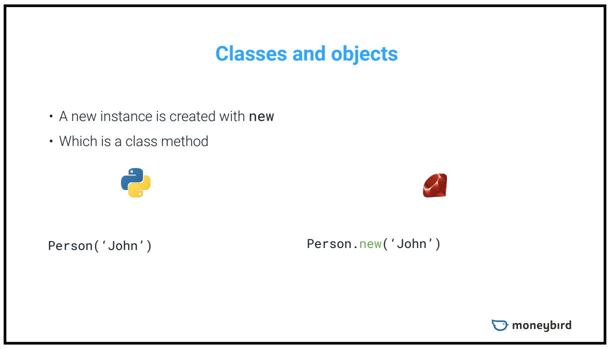
Classes and objects

- Instance methods are defined within the class
- They can use instance variables

```python
class Person:
  def full_name(self):
    return (
      self.firstname + “ “ +
      self.lastname
    )
```

```ruby
class Person
  def full_name
    @firstname + “ “ + @lastname
  end
end
```

moneybird

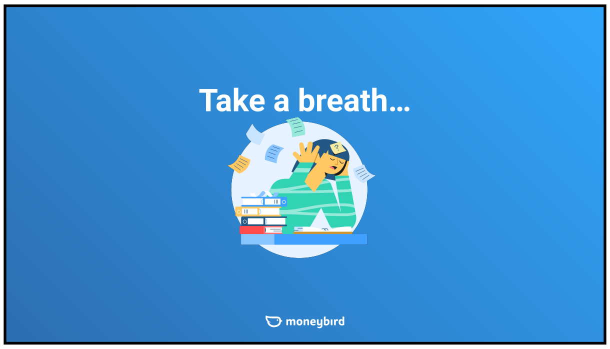


Classes and objects

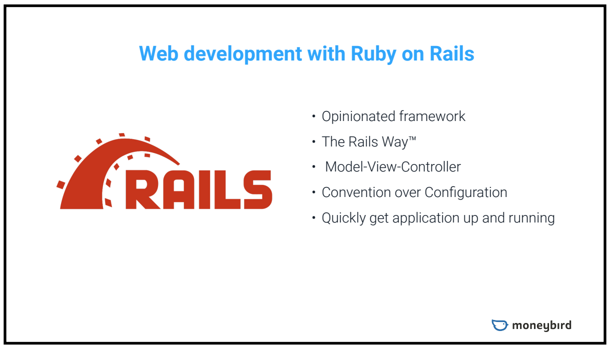
- A new instance is created with `new`
- Which is a class method

```
Person('John')
```

```
Person.new('John')
```

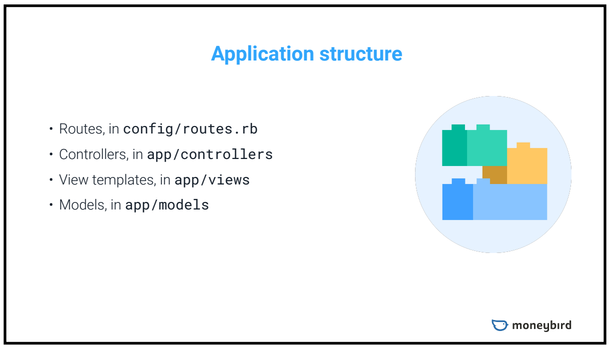


Take a breath...



Web development with Ruby on Rails

- Opinionated framework
- The Rails Way™
- Model-View-Controller
- Convention over Configuration
- Quickly get application up and running



Application structure

- Routes, in `config/routes.rb`
- Controllers, in `app/controllers`
- View templates, in `app/views`
- Models, in `app/models`

Routes

- Maps request to a controller action
- Routes are defined in a separate file, `config/routes.rb`
- A route has a URL pattern and a format (HTML by default)
- In this case, the `index` action of `GreetingsController`

```
Rails.application.routes.draw do
  get “/greetings”, to: “greetings#index”
  post “/greetings”, to: “greetings#create”
end
```

moneybird

Routes

- Rails defines a helper method for each route
- Execute `bin/rails routes` in the terminal
- Look at the prefix, and prepend `_path` to it
- In this case, `greetings_path` for `index` action of `GreetingsController`

```
$> bin/rails routes
Prefix       Verb URI Pattern             Controller#Action
greetings    GET  /greetings(.:format)    greetings#index
             POST /greetings(.:format)    greetings#create
```

moneybird

Routes

- The standard actions in Rails controllers are CRUD operations:
  - index, new, create, show, edit, update, destroy
- `resource` is a useful shorthand to add a route for each CRUD action

```
Rails.application.routes.draw do
  resource :greetings
end
```

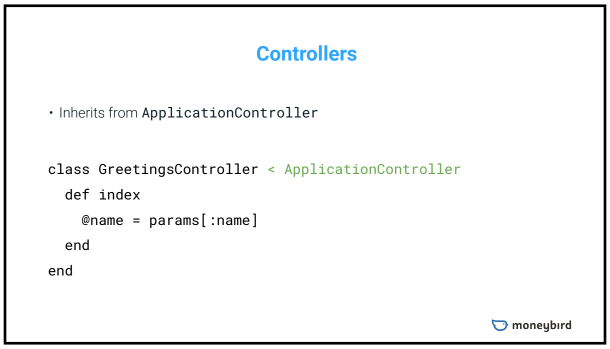
moneybird

Controllers

- Handles a request
- Prepares data for the view
- Convention: controller name is plural, and ends with `Controller`

```
class GreetingsController < ApplicationController
  def index
    @name = params[:name]
  end
end
```

moneybird

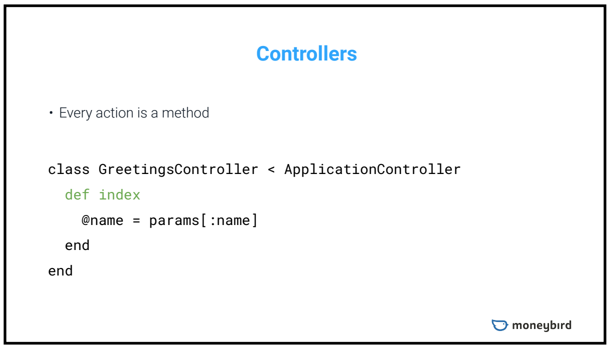


Controllers

- Inherits from `ApplicationController`

```
class GreetingsController < ApplicationController
  def index
    @name = params[:name]
  end
end
```

moneybird

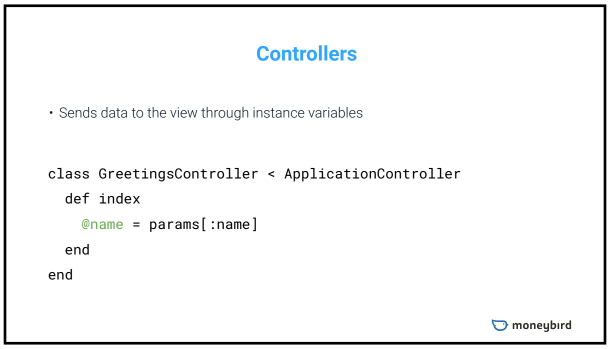


Controllers

- Every action is a method

```
class GreetingsController < ApplicationController
  def index
    @name = params[:name]
  end
end
```

moneybird

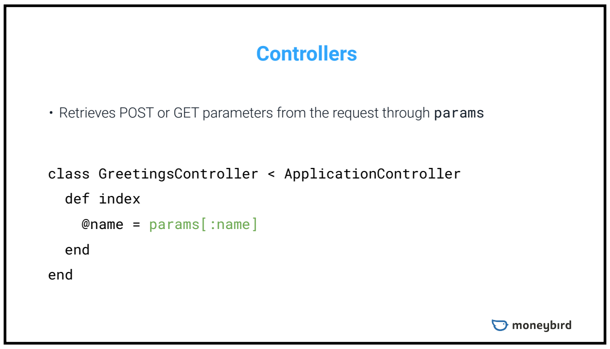


Controllers

- Sends data to the view through instance variables

```
class GreetingsController < ApplicationController
  def index
    @name = params[:name]
  end
end
```

moneybird



Controllers

- Retrieves POST or GET parameters from the request through `params`

```
class GreetingsController < ApplicationController
  def index
    @name = params[:name]
  end
end
```

moneybird

Controllers

- By default renders a view template based on controller and action name
- In this case, `app/views/greetings/index.html.erb`, or `app/views/greetings/index.json.erb`
- Use `render` to render another view template

```
class GreetingsController < ApplicationController
  def index
    render(:something_else)
  end
end
```

Controllers

- `redirect_to` sends the user to another page, causing a new request
- Never used together with `render`!

```
class GreetingsController < ApplicationController
  def index
    redirect_to(new_greeting_path)
  end
end
```

Controllers

- `respond_to` performs a different action for each request format (HTML, JSON)

```
class GreetingsController < ApplicationController
  def index
    respond_to do |format|
      format.html { redirect_to(root_path) }
      format.json { render(:some_template) }
    end
  end
end
```

Views

- Uses ERB (Embedded Ruby)
- Similar to Jinja
- Everything non-ERB is printed to output unchanged

```
<h1>Hello world!</h1>
```

Views

- Use `<% … %>` to evaluate statement
- Same as `{% … %}` in Jinja
- All Ruby code allowed
- `if` also needs an `end`
- Body is never printed to output

```
<% if first_time_here? %>
  <h1>Welcome!</h1>
<% else %>
  <h1>Hello again!</h1>
<% end %>
```

moneybird

Views

- Use `<%= … %>` to evaluate an expression and print it to output
- Same as `{{ … }}` in Jinja

```
<h1>Hello <%= 'world' %>!</h1>
```

moneybird

Views

- Instance variables from controller are accessed as instance variables in view template

```
<h1>Hello <%= @name %>!</h1>
```

moneybird

Views

- More complicated example

```
<h1>Hello <%= @name %>!</h1>
```

moneybird

Views

- Views can also render other views
- These are called *partials*
- The filename of a partial begins with an underscore
- In this case, `app/views/greetings/_foo.html.erb`

```
<h1><%= render('foo') %>!</h1>
```

moneybird

Models

- Models interact with the database
- Uses ActiveRecord as its ORM
- Model name is always singular
- ActiveRecord handles a lot of the magic

```
class Greeting < ApplicationRecord
end
```

moneybird

Models

- Models interact with the database
- Uses ActiveRecord as its Object Relational Mapper
- Model name is always singular
- In this case, `Greeting` maps to `greetings` table

```
class Greeting < ApplicationRecord
end
```

moneybird

Models

- Validations are used to check the attributes
- `presence` checks if the attribute has a value
- The model is invalid if a validation fails

```
class Greeting < ApplicationRecord
  validates :message, presence: true
end
```

moneybird

Models

- A new model can persisted with `save`
- If `save` returns `false`, a validation failed

```
Greeting.new(message: 'Hi!').save # true
Greeting.new(message: '').save    # false
```

moneybird

Models

- After being persisted, the model gets an ID
- The ID can be used to read the model from the database later

```
greeting = Greeting.new(message: 'Hi!')

greeting.id      # nil
greeting.save    # true
greeting.id      # 1

Greeting.find(1)
```

moneybird

Models

- An existing model can be updated with `update`
- Also returns `false` if a validation failed

```
greeting = Greeting.find(1)

greeting.message                   # "Hi!"
greeting.update(message: "Hello!") # true
greeting.message                   # "Hello!"

greeting.update(message: "")       # false
```

moneybird

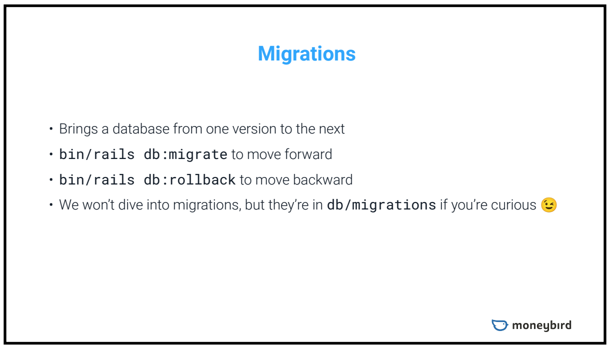
Models: relationships

- Relationships between models are defined in the models too.
- For example, if the `employees` table has a `company_id` column:

```
class Company < ApplicationRecord
  has_many :employees
end
```

```
class Employee < ApplicationRecord
  belongs_to :company
end
```

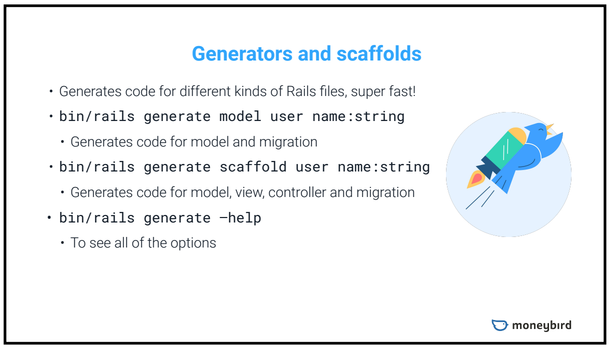
moneybird



Migrations

- Brings a database from one version to the next
- `bin/rails db:migrate` to move forward
- `bin/rails db:rollback` to move backward
- We won't dive into migrations, but they're in `db/migrations` if you're curious 😉

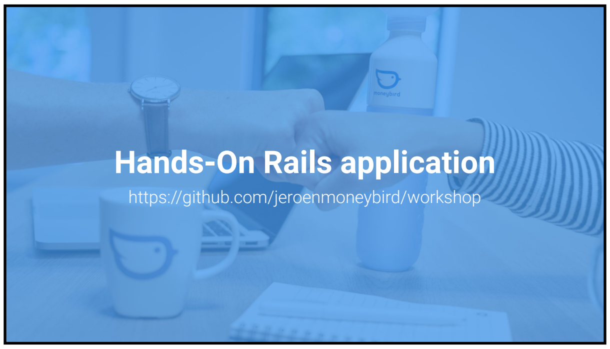
moneybird



Generators and scaffolds

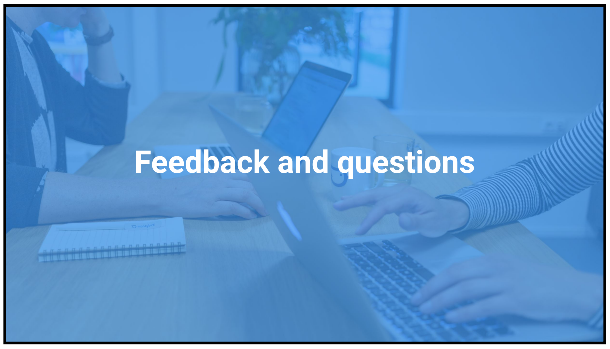
- Generates code for different kinds of Rails files, super fast!
- `bin/rails generate model user name:string`
  - Generates code for model and migration
- `bin/rails generate scaffold user name:string`
  - Generates code for model, view, controller and migration
- `bin/rails generate —help`
  - To see all of the options

moneybird

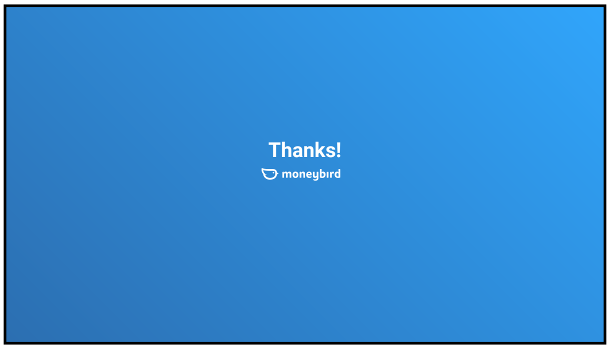


Hands-On Rails application

https://github.com/jeroenmoneybird/workshop



Feedback and questions



Thanks!

moneybird