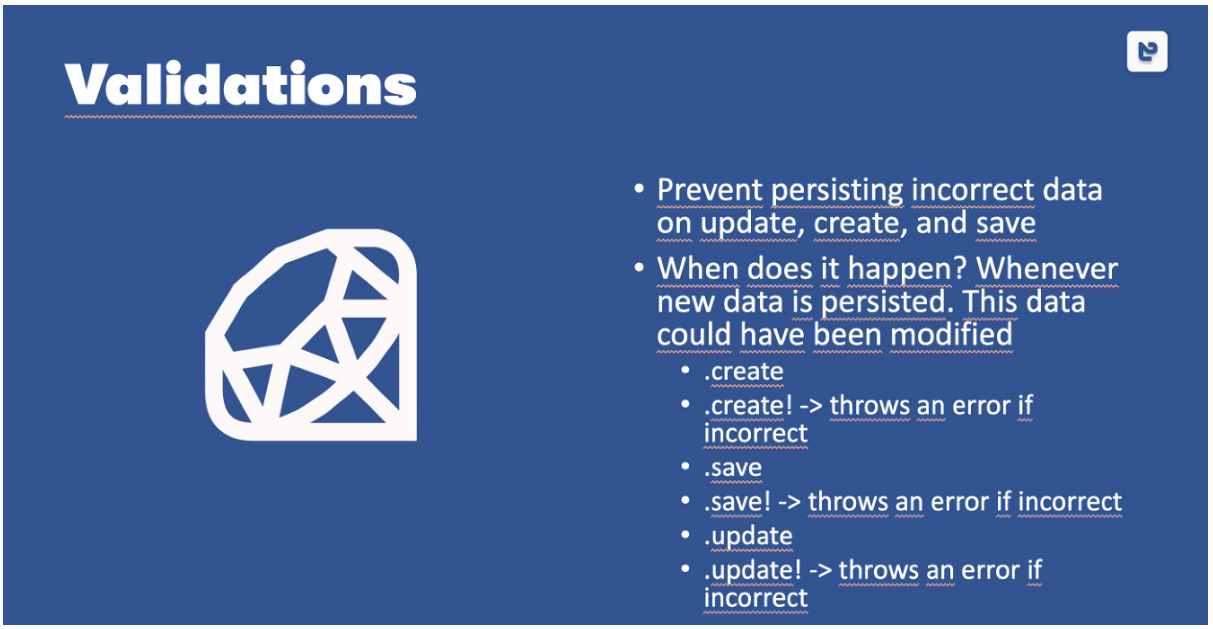
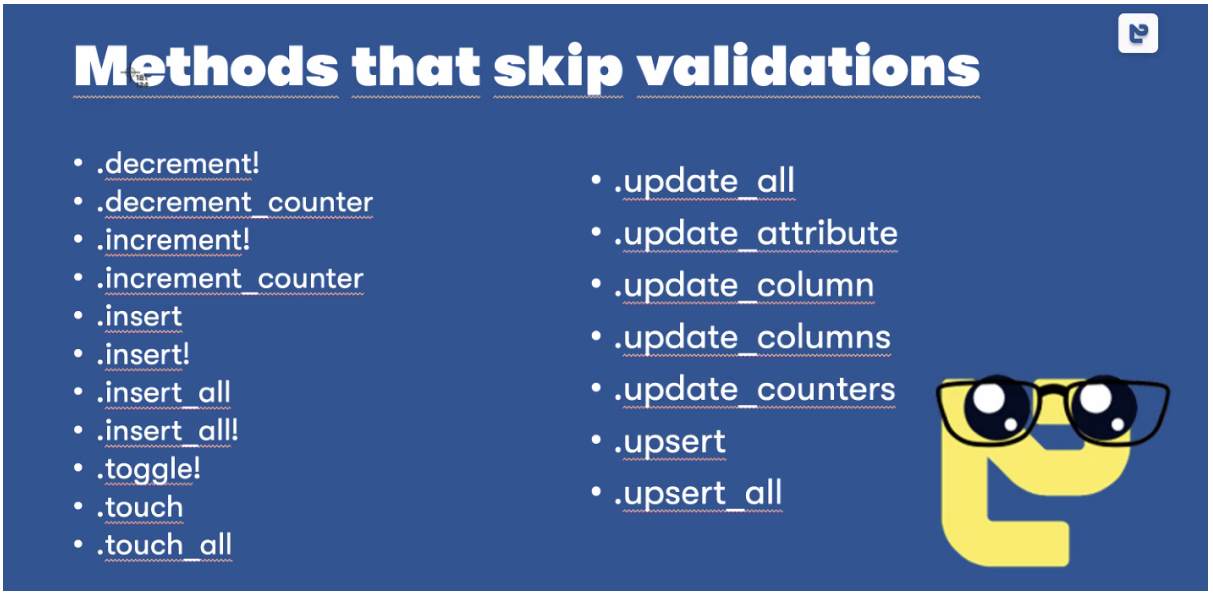
VALIDATIONS



Do not confuse the before_action methods with custom validations.
Those ones are for controllers, the other ones for models.

Validations is the way rails have to avoid entering bad data into the database.

There are methods that skips validations



They also skip callbacks.
You don't want to use them apart of some exceptions.

Skip validation with save method

Other way to skip a validation

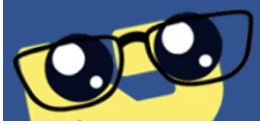
- `save(validate: false)`

How do validations work?

- Before persisting data, validations are run
- If any errors are produced, the object is not persisted
- How to check if the data was saved?
  - Use .valid? after trying to persist.

```
Person.create(name: "John Doe").valid? # => true
Person.create(name: nil).valid? # => false
```

Getting the error messages, if any



```
>> p.valid?
# => false
>> p.errors.messages
# => {name:["can't be blank"]}
```

Adding your own errors

```ruby
class Person < ApplicationRecord
  def a_method_used_for_validation_purposes
    errors.add(:name, "cannot contain the characters !@#%*()_-
+=")
  end
end

person = Person.create(name: "!@#")

person.errors[:name]
 # => ["cannot contain the characters !@#%*()_-+="]

person.errors.full_messages
 # => ["Name cannot contain the characters !@#%*()_-+="]
```

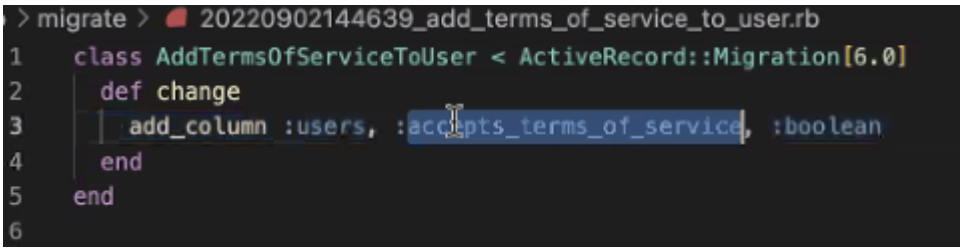

Clearing the errors

- Use .errors.clear
- You can use the full messages view helper (available in views as well)

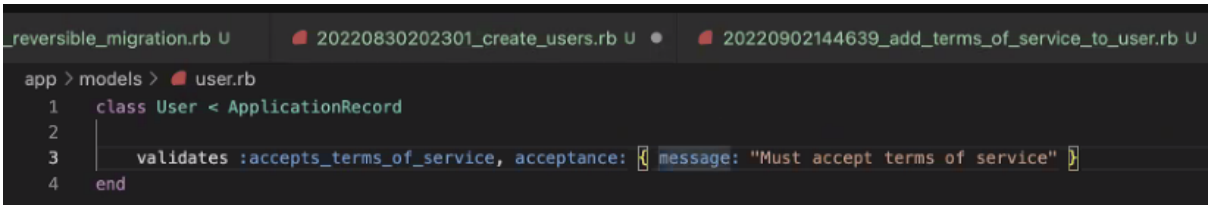
```erb
<% if @article.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@article.errors.count, "error") %>
prohibited this article from being saved:</h2>

    <ul>
    <% @article.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
<% end %>
```

Here we are creating validations for a class.

rails g migration addTermsOfServiceToUser accepts_terms_of_service:boolean



Normally we represent a boolean value on a form as a checkbox.



Acceptance-message is an custom error message
These validations are only working for forms ???

Validating associated models

- Validating related objects that may be in a has_many, belongs_to, has_one...

```
class Library < ApplicationRecord
  has_many :books
  validates_associated :books
end
```

It is Important for rails.
If you create a library and if any of the books are not valid then the library or any of the books won't be added.
Don't use both sides of the validations, only one side, if not there is an infinite loop.

Validations – confirmations

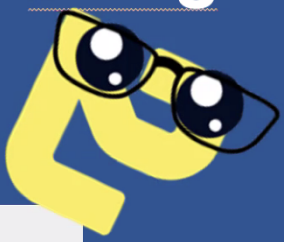
- Creates a second field that should be exactly the same as the origina

```
class Person < ApplicationRecord
  validates :email, confirmation: { case_sensitive: false }
end
```

It checks if two values are the same.

Validations – exclusions in a string

- Accepts Regex



```
class Account < ApplicationRecord
  validates :subdomain, exclusion: { in: %w(www us ca jp),
    message: "%{value} is reserved." }
end
```

Validations – string length

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```