Section Review

Learn to Code with Ruby

Blocks

- Methods in Ruby support **blocks**, which customize some part of the method's behavior.
- The method can encapsulate the constant parts of a procedure while enabling the dynamic parts to vary.
- For example, the **each** method handles array iteration while allowing the Ruby developer to customize what happens to each element. The **map** method handles building up a new array of values while allowing the Ruby developer to customize how each element is derived.

The yield Keyword

- The **yield** keyword transfers control to a block. We can pass any number of objects to the block. The block accepts them via **block parameters**.
- Ruby will raise an exception if we use the **yield** keyword and do not provide a block during method invocation.
- The **block_given?** predicate method can validate if a block was provided.

Procs

- A **Proc** is an object that represents a procedure. Think of it like a reusable a block.
- A block is disposable. It's designed more for one-time use.
- Declare a Proc with **Proc.new** and a block representing the procedure. Ruby will use that block to represent the internal execution logic.
- An alternative option is the **proc** keyword.

Methods with Proc Parameters

- We can define methods that accept **Procs**. Prefix the Proc parameter with a **&**. The proc parameter does not need a & in the method body.
- Inside the method body, use the **call** method to run the Proc (this transfers control to the Proc that is passed during method invocation).
- A method with a **Proc** parameter can still accept a block. Ruby will convert the block to a Proc.

Intro to Lambdas

- Lambdas behave like a stricter Proc. We can define a method that accepts a lambda by prefixing a parameter with **&**. Use **call** to invoke the lambda.
- To declare a lambda, use the **lambda** keyword followed by a block with the procedural logic.
- An alternative option is the **->(parameter) { ... }** syntax.

Differences between Procs and Lambdas

- A **lambda** will raise an exception if we pass it the wrong number of arguments from the method. A **Proc** will ignore any extra arguments and assign **nil** to any missing ones.
- When a **lambda** uses **return**, it transfers control back to the calling method. When a **Proc** uses **return**, it terminates the calling method as well.