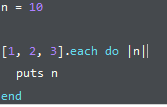
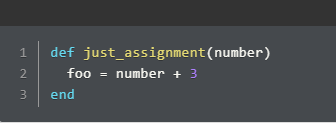
# Part 1: Study Guide for Test

#### Specific Topics of Interest

Be able to explain clearly the following topics:

* local variable scope
  + Determines where in  a program a variable is available for use
  + Defined  by where the variable is initialized or created
  + In Ruby, variable scope is defined by a *block*
  + A block is a piece of code following a method invocation, usually delimited by either curly braces {} or do/end
* especially how local variables interact with method invocations with blocks and method definitions
  + Visual mental model:
    - **Inner scope can access variables initialized in an outer scope, but not vice versa.**
    - **Local variables initialized outside of a block delimited by do...end are like people who live in the suburbs.  They go into the city (the block delimited by do...end) to work.  Therefore, those local variables are accessible outside of the block.**
    - **Local variables initialized inside the block however, are like people who live in the city (the block delimited by do..end).  People in the city do not work in the suburbs.  Therefore, you will raise an error if you try to use a local variable originally initialized within the block delimited by do..end outside of the block.**
* especially how local variables interact with method definitions
  + Fortress
  + A method definition--you can’t break into it EXCEPT when you pass it as an argument to it.
* Variable shadowing
  + Def: two local variables in the inner scope with the same name
  + **variable shadowing** occurs when a variable declared within the inner scope of a block delimited by do..end has the same name as a variable declared in an outer scope.
  + **variable shadowing(Wikipedia definition)** occurs when a variable declared within a certain [scope](https://en.wikipedia.org/wiki/Scope_(computer_science)) (decision block, method, or inner class) has the same name as a variable declared in an outer scope.
  + But what if we had a variable named n in the outer scope? We know that the inner scope has access to the outer scope, so we'd essentially have two local variables in the inner scope with the same name. When that happens, it's called *variable shadowing*, and it prevents access to the outer scope local variable
  + The puts n will use the block parameter n and disregard the outer scoped local variable. Variable shadowing also prevents us from making changes to the outer scoped n:
  + 
  + Srdjan’s Variable Shadow Definition:
    - This is the concept called **variable shadowing** and it happens when parameter name of the block is the same as the name of the local variable which was initialized outside of the block.
    - Prioritize own name
    - naming the same as outerscope, making it so block can’t access outer scope variable
* how passing an object into a method definition can or cannot permanently change the object
  + Refer to Christian Larwood’s study guide
* working with collections (Array, Hash, String),
* and popular collection methods (each, map, select, etc). Review the two lessons on these topics thoroughly.
  + - #select returns a new array based on the block's return value. If the block's return value evaluates to true, then the element is selected
  + - #map returns a new array based on the blocks return value. Each element is   transformed based on the return value (block parameter does not have to be used)
  + - #each returns the original array the method was called on. The blocks return value does not influence what the method each returns
* [variables as pointers](https://launchschool.com/books/ruby/read/more_stuff" \l "variables_as_pointers)
  + See articles
  + Variables point a specific place in memory
* [puts vs return](https://launchschool.com/books/ruby/read/methods" \l "putsvsreturnthesequel)
  + See articles
  + puts is outputs a string to the screen and returns nil
  + Return is the evaluated result of the executed code; the last expression in a method is returned; has chaining abilities;
  + 
    - The value of just\_assignment(2) is going to be 5 because the assignment **expression evaluates to** 5, therefore that's what's returned.
* false vs nil and the idea of "truthiness"
  + In Ruby: true is true
    - false is falsy
    - Nil is falsy
    - All other objects are truthy
* So what are mutating methods in Ruby?
  + Shortest answer would be, that those are methods which are changing the value of a calling object.
* method definition and method invocation

##### Method Definition and Method Invocation

* + When discussing methods, particularly in terms of how blocks and methods interact with local variables, we want you to explain this in terms of method definition and method invocation. You can review [this assignment](https://launchschool.com/lessons/a0f3cd44/assignments/9e9e907c) for an outline of the mental model to use.
    - Methods are defined using the keywords def and end
      * Has its own variable scope
      * Returns the last evaluated expression
    - Methods are called/invoked by writing out the method name;
      * Some methods require block parameters, others don’t
  + implicit return value of method invocations and blocks
    - Implicit return value of method invocations
      * Is simply the last evaluated expression of the invoked method
    - Implicit return value of method blocks
      * Is simply the last evaluated expression of the block
        + The block is defined when it is called on the method
* how the Array#sort method works
  + Utilizes the ⇔ operator?
    - Value to the left -1, 0, 1
    - uses the return value where in the Array to place the element;
    - also know that if b < = > a, it would be descending order
  + Review docs
  + Returns a new array created by sorting self.
  + Comparisons for the sort will be done using the <=> operator or using an optional code block.
  + The block must implement a comparison between a and b and return an integer less than 0 when b follows a, 0 when a and b are equivalent, or an integer greater than 0 when a follows b.
  + The sort method uses the spaceship operator and compares one by one, adjacent objects.
    - It determines if the first object is less than, equal to, or greater than the second number.
    - If the first number is less than the second, returns -1
    - Equal → returns 0
    - First number greater than second, returns 1
    - Based on return values, flips numbers/strings until they are from least to greatest or earliest in the alphabet to latest
    - http://cs.armstrong.edu/liang/animation/web/BubbleSortNew.html

Truthiness

In the assessment we want you to be very clear about the distinction between *truthy* and the boolean true (and similarly the distinction between *falsey* and the boolean false).

In Ruby, every value apart from false and nil, *evaluates to true* in a boolean context. We can therefore say that in Ruby, every value apart from false and nil is *truthy*; we can also say that false and nil are *falsey*. This is **not** the same as saying every value apart from false and nil **is** true, or **is equal to** true. These may seem like subtle distinctions but they are important ones.

To sum up:

* Use "evaluates to true" or "is truthy" when discussing an expression that evaluates to true in a boolean context
* Do not use "is true" or "is equal to true" unless specifically discussing the boolean true

#### Variable References and Object Mutability Articles

We wrote a series of blog posts that thoroughly cover variable references and object mutability:

* [Variable References and Mutability of Ruby Objects](https://launchschool.com/blog/references-and-mutability-in-ruby)
* [Mutating and Non-Mutating Methods in Ruby](https://launchschool.com/blog/mutating-and-non-mutating-methods)
* [Object Passing in Ruby - Pass by Reference or Pass by Value](https://launchschool.com/blog/object-passing-in-ruby)

with the method definition the parameter is being set up to be initialized,

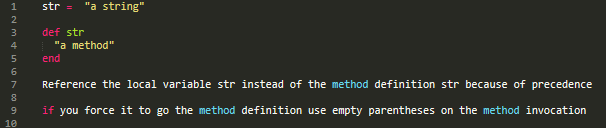
string = 'Hello'

* + method(string)
  + The local variable string is passed to method as an argument. At method invocation, the method local variable param is initialized to the object referenced by string. Both string and param now point to the same object, 'Hello'.

Spend some time reviewing these articles to ensure that you have a good understanding of these topics.

#### Assessment Prep Videos

We did a Beginning Ruby series that will serve as great review for this test. Specifically, the sessions that are relevant to this assessment are:

* [Part 1](https://launchschool.com/blog/live-session-beginning-ruby)
  + 
* [Part 2](https://launchschool.com/blog/live-session-beginning-ruby-part-2)
* [Part 3](https://launchschool.com/blog/live-session-beginning-ruby-part-3)

Pay careful attention to the way the instructor explains concepts, and use that vocabulary to describe code on the test.

Several Launch School students have blogged about their assessment experiences:

Srđan prepared a four-part [blog series](https://medium.com/how-i-started-learning-coding-from-scratch/advices-for-109-written-assessment-part-1-6f7fa821cf84) in which he reviews the core concepts covered in course RB101 that should be mastered prior to taking the assessment.