**Becoming a Rubyist with Style**

Let's take a moment to highlight some syntactical and stylistic features of Ruby that make it a very readable and expressive language. Ruby is flexible in that it allows many "shortcuts" that can make code cleaner. These "shortcuts" are initially uncomfortable for programmers to read and write, especially if they are coming from another language with stricter syntax. Since you have completed our Intro to Ruby course already, we can now take off the training wheels and begin to write code like a true Rubyist! Being a Rubyist means using the faculties of the language to your advantage. You might have experience with the programming concepts in other languages, but in this course it’ll be essential to understand how to implement these concepts in Ruby!

It's okay if you are unfamiliar with some of the methods we example below, we'll learn them over time. However, you should be able to gather what they do from their names alone.

Proper style in our code is important because it is the mark of high quality work and an experienced programmer!

**Implicit Returns**

Methods in Ruby will automatically return the evaluation of their last executed expression. You will still need to use the explicit return keyword to do an early return.

# Less preferred

def get\_avg(num\_1, num\_2)

return (num\_1 + num\_2) / 2

end

# Preferred by a Rubyist

def get\_avg(num\_1, num\_2)

(num\_1 + num\_2) / 2

end

**Omitting parentheses for method calls with no arguments**

When calling a method without passing any arguments, we'll often drop the parentheses altogether.

def say\_hi

puts "hi"

end

# Less preferred

say\_hi()

# Preferred by a Rubyist

say\_hi

**Use single line conditionals when possible**

When we have a single line in the body of a simple if statement (that is not attached to an elsif or else), we can turn it into a one-liner:

raining = true

# Less preferred

if raining

puts "don't forget an umbrella!"

end

# Preferred by a Rubyist

puts "don't forget an umbrella!" if raining

**Use built-in methods**

There are many methods in Ruby that can make your life easier. Use them:

num = 6

# Less preferred

p num % 2 == 0

# Preferred by a Rubyist

p num.even?

people = ["Joey", "Bex", "Andrew"]

# Less preferred

p people[people.length - 1]

# Preferred by a Rubyist

p people[-1]

p people.last

**Use enumerables to iterate**

There are many enumerables in Ruby that have specific use cases. These tools can really make the code read like english. Often times, you can avoid using a while loop in favor of a more readable enumerable.

# Less preferred

def repeat\_hi(num)

i = 0

while i < num

puts "hi"

i += 1

end

end

# Preferred by a Rubyist

def repeat\_hi(num)

num.times { puts "hi" }

end

Given a problem, not all enumerables are equal. Some methods will immediately solve the problem at hand elegantly.

# Less preferred

def all\_numbers\_even?(nums)

nums.each do |num|

return false if num % 2 != 0

end

true

end

# Preferred by a Rubyist

def all\_numbers\_even?(nums)

nums.all? { |num| num.even? }

end