CIT 483/583 Lab 02 – Built-in Essentials

Fall 2017

**Instructions**

Open and save this file in any MS word-compatible format as Lab02\_*Firstname*\_*Lastname*.<ext> and place your answers in that document. Do a **Save-As** and retain all of my content. Keep the document safe in case your submission fails, or you discover an error prior to the due date and wish to re-submit. Submit your document to the Lab02 dropbox in Canvas. The due date and any other pertinent information are noted in the Canvas item.

This lab should be completed on students.cs.nku.edu as a reference implementation and as a means of testing your answers. It can be done using IRB only, or you may create .rb files to test some or all of your code, but do not submit .rb files (programs).

**Place the answers in or immediately following each question and make sure your answers stand out from the questions by using a different font color.**

1. Literal construction is the only way to construct certain objects. Other objects that are very easy to construct as literals seem awkward when using the *new* method. Still others have constructor forms that do not use the *new* method but are not literals. Run the following commands and capture the output. **Not all of them should be successful, so simply show the error that occurs in that case.**

10.object\_id

n = Fixnum.new

"A String object".object\_id

s = String.new

s = String.new("This is the long way to make a String")

f = Float.new

f = Float(2)

n = Fixnum(2)

r = Rational(1, 2)

r.class

NilClass.new

nil.class

1. Provide the "non-sugared" versions of the following expressions. Show both the altered expression and the result.
2. 2 + 3
3. 2 \* 3
4. x == y
5. arr[x] = 7
6. Bang (!) methods are typically paired with non-bang methods: sort/sort!, chomp/chomp! For each expression, determine whether the method either does or does not alter the original object using a "before vs. after" display.

The \_id\_\_ method can be helpful in some cases, but not all destructive methods return the ID of the object that was altered. If the \_\_id\_\_ method confirms that the object in question has been altered, highlight the ID pairs that demonstrate this.

WARNING: Not all destructive methods use a ! suffix and not all methods with the same name have the same destructive/nondestructive behavior for different object types. For example, delete for a String and delete for an Array are different.

(Set up for questions a-e)

s = "Test string"

s.\_\_id\_\_

s.reverse

s.reverse.\_\_id\_\_

puts s

s.reverse!

s.reverse!.\_\_id\_\_

puts s

s.downcase

s.downcase.\_\_id\_\_

puts s

s.downcase!.\_\_id\_\_ # the first time

s.downcase!.\_\_id\_\_ # second time it is called is different(???)

nil.\_\_id\_\_

puts s

s.delete("t") # in contrast to delete! for Strings

s.delete("e").\_\_id\_\_

puts s

1. (Unrelated to a-e)

arr = [ 3, 2, 1, 0 ]

arr.\_\_id\_\_

p arr

arr.delete(0).\_\_id\_\_

p arr

arr.delete(1).\_\_id\_\_

1.\_\_id\_\_

1. Most conversions are self-explanatory given the object (the receiver) and the method (the message). The more important aspect is to understand the type of the object returned and whether it has any relationship to the original or to other objects of the same type. Run the following commands and capture the output.

s = "Hello"

sym = :Hello

s.class

s.\_\_id\_\_

s.to\_sym

s.to\_sym.\_\_id\_\_

:Hello.class

:Hello.\_\_id\_\_

sym.\_\_id\_\_

p s.to\_sym

sym == s.to\_sym

sym.equal?(s.to\_sym)

1. String interpolation allows any expression whose value responds to the *to\_s* method to be placed in String literals. Run the following commands and capture the output.

puts "I know these things to be #{ obj = Object.new; obj.methods.include?(:eql?) }"

my\_hash = { :OH=>"Ohio", :IN=>"Indiana", :KY=>"Kentucky" }

my\_hash.each { |k,v| puts "Key: #{k}, value: #{v}" }

my\_hash.each { |k,v| puts "Key: #{k.downcase}, value: #{v.upcase}" }

my\_hash.each { |k,v| puts "Key type: #{k.class}, value type: #{v.class}" }