Q1. What does string interpolation mean? Provide an example.

**String Interpolation**

It is essential to be able to replace placeholders within a string with values they represent. In the programming paradigm, this is called "string interpolation".

*Example Code:*



a = 1

b = 4

puts "The number #{a} is less than #{b}"

class answer:

Q2. What is the difference between single quoted strings and double quoted strings in Ruby?

what difference is there between single quotes and double quotes in Ruby? In the above code, there's no difference. However, consider the following code:

puts "Betty's pie shop" puts 'Betty\'s pie shop'

Because "Betty's" contains an apostrophe, which is the same character as the single quote, in the second line we need to use a backslash to escape the apostrophe so that Ruby understands that the apostrophe is *in* the string literal instead of marking the end of the string literal. The backslash followed by the single quote is called an[escape sequence](https://en.wikipedia.org/wiki/escape_sequence" \o "w:escape sequence" \t "_blank).

Because "Betty's" contains an apostrophe, which is the same character as the single quote, in the second line we need to use a backslash to escape the apostrophe so that Ruby understands that the apostrophe is *in* the string literal instead of marking the end of the string literal. The backslash followed by the single quote is called an[escape sequence](https://en.wikipedia.org/wiki/escape_sequence" \o "w:escape sequence" \t "_blank).

Class answer: cant do string interpolation in single quotes

Q3. Why do we call chomp on gets? What does it do?

The main different between “gets” and “gets.chomp” in ruby is, when you get the input from user, using “gets” method the outputs are ‘ll come like “…/n”. i mean the output ‘ll come whatever you gave and at the same time a “/n” also ‘ll come next to the output, because of when we enter the “Enter” button the “gets” method also assume  “Enter” is like as a new line input.

So only, we using the “gets.chomp” method. “chomp” means like “cut”. Because when we using “.chomp” method, it cuts the last parameter like “/n” from input.

Class answer: when someone presses enter after giving their input it starts a new line the chomp stops that

Q4. What is a R-E-P-L?

Read Evaluate Print Loop. Design pattern in software

Q5. What’s the difference between the following 2 lines of Ruby code?

puts "it's not true!" if !true

puts "it's not true!" unless true

### prettty much the same thing, the ruby style is to use the unless.

### Q6

Is the following ruby code valid? If so, describe what it’s doing. (Hint: Try it out in IRB)

Pooped = true

Underwear = if pooped

:soiled

else

:safe

end

puts underwear

underwear is soiled.

Lecture

## Plan (tentative)

1. Take up [homework from previous day](http://compass.lighthouselabs.ca/days/w1d2/activities/63" \t "_blank)
2. Hashes / Dictionaries
3. Arrays
4. Symbols
   * How do they compare to strings?
   * When are they used?
5. How scopes work in Ruby wrt blocks, methods and files

ARRAY QUESTION 1: are arrays ordered and sorted or just sorted or ordered

They are ordered. Order remains constant, sorted changes. Ordered are ordred according to the index – google.

* Arrays are objects --- actually almost everything in ruby are objects – ruby is a object oriented language
* Names = Array.new(20)
* Puts names.size # This returns 20
* Puts names.length # This also returns 20
* To get the last index value of the array 🡪 names[length -1]
* There could be ways where size and length differs but he hasn’t come across it yet

Do and end replaces curly braces and holds a block of code

Nums = Array.[](1,2,3)

Nums = Array[1,2,3,4]

Nums = [1,2,3]

All the same

Basic array w loops

[1,2,3].each do |number|

puts number

end

nums.reverse.each do |n|

puts “value: #{n}”

end

names = [“adam” daniel, “jon”]

puts names[0]

puts names[1]

puts names[2]

puts names[3]

when you run this puts names 3 will be blank

but if we change puts to p itll give nil. 🡪 p gives the true value of

Intermediate array access

Puts names[-1]

Puts names.last

Puts names.at(2)

# Mixed Array – in lecture notes

the array contains other arrays and a hash.

We access the hash by doing puts random [3]. But if we want to hit the other email which is the second value in the hash we would

Puts random[3].class 🡪 if you run it now itll say it’s a hash, but we cant look at hash values with aray lookups. So what we can do is

Data\_hash = random[3]

Puts data\_hash[:other\_email]

We created a arbitrary variable data hash to access the 3rd index value of random. Then puts the symbol out

We can also do email\_2 = random[3][:other\_email]

If we want the first email we do – first\_email = random[3][“email”]

**\*\*\*\* the #{[x,y,z]} is to print it horizontally not vertically \*\*\*\*\***

**Random.each do |r| ----- line 111 – this array is referring to the mixed array from earlier**

* This gets the class of the random array we created.. there are 3 possible classes – array, hash and fix num.
* This shows if it’s a fix num just print it, if array then print it like ----- x. if hash then print it like key: #{key}: value: #{value}
* The elsif c == Array – p “---“ + “ “ + element.to\_s 🡪 were changing this to a string because if you’re an array with integers inside it it cant concatenate integers.. so we have to convert to strings.

**# ARRAY BUILT IN METHODS ----- line 142ish**

* What this did was concatenate both arrays (add [<<]). Not individual values but combined both into one array.

**puts “1,2,3”.split(“,”).join(“/”)**

* What this does is split each integer at the comma into an array from a string and then bringing it back together into a string but now separated with /’s instead of commas.
* When we do “1,2,3”.split(“,”) it gives [“1”, “2”, “3”] 🡪 “1,2,3”.split(“,”).class = Array
* puts “1,2,3”.split(“,”).join(“/”).class = String (back to string)

**A set:**

**Puts “#{[1,2,3,4,5] & [4,5,6]} 🡪 gives [4,5] – returns common values in both**

**Puts “#{[1,2,3,4,5] | [4,5,6]} 🡪 gives [1,2,3,4,5,6] – returns a new array by joining array with other\_array, removing duplicates.**

Puts “#{[1,2,3] \* 3}” 🡪 gives [1,2,3,1,2,3,1,2,3]

* gives 1 array with the 3 index values printed 3 times

puts “#{[4,5,6,7] – [6 – 7]}” 🡪 gives [4,5]

# Line 229 ish – if [[1,2], “0”, {key: “value”}] == [[1,2], “0”, {key: “value”}]

puts true

end

* This not only compares whether each are arrays with 3 indexes, it also checks the value of the indexes are equal… so if strings compares both strings, if hash, are the key and value the same.

Letters = [“a”, “b”, c”, d”, e”]

Puts letters [2,4]

This gives [‘c’,’d’,’e’] – gives from 2nd index to the 4th index.

Puts letters.delete\_at(3) 🡪 deletes the 3rd index value, and puts it.

Puts letters.index(“b”) 🡪 gives 1 .. b is in the first index

Puts letters.push(“x”) 🡪

If ur in your messages on iphone, in the messages app ur in the 0th index, click on somone and that’s the 1st index then click details that’s the 2nd index…. If you wanna go back you go to 1st index then 0th index..

This brings up the topic of LIFO (last in first out) 🡪 [].push(1) gives [1] 🡪 [1] .push(2) 🡪 [1,2] etc..

This is essentially a stack. -- > this allows us to push stuff into an array and then pop it off.

If you add you add to the last and if you wanna pop it off you pop off the last one..

Puts letters.pop 🡪 the pop calls .delete\_at(-1).. deletes the last index value…

Then – puts letters will show that it has popped off.

Puts “#{letters}”

Puts letters.reject! { |item| item ==”c” }

Puts “#{letters}”

The 2nd line uses a method on letters then creates a loop to the item c.

Puts “#{[2,6,3,7,8,2,1,4,6,8,9,4,2,5,7].uniq.sort.reject! {|item| item % 2 ==0}}”

* This removes all repeating values, then sorts the numbers and rejects (removes) all values evenly divisble by 2.. so youre left with 🡪 [1,3,5,7,9]

Symbol – immutable – value remains constant during the entirety of the program.

Ex – name = “adam” --- gives adam .. then name = “jon” ---- gives jon.

As for symbols … name = :adam 🡪 gives :adam --- name = :jon 🡪 gives :jon

They don’t change

Symbols are performance optimizers.. they use way less memory because for every operation they don’t have to create different strings, they work off of the same symbols.

Comparison

Different Ways to create hashes – check lecture –

At the example with person 5 – jen toronto –

To get the persons home phone number you do –

Person5[:phone\_numbers][:home]

Tutorials point for ruby.

You can also ask for all the keys in a hash.. like person5.keys

Or person5.clear 🡪 would wipe everything from the hash

Creating Hashes

Contacts = Hash.new

Contacts = Hash.new( “contact” ) --- actuallygiving a default value

Puts contacts[“contacts”]

Will just give a default key value that says contacts.