**Question 1**

**What is a Class vs an instance of a Class?**

An object is data, and the methods with which to act on that data.

Class is a prototype of the methods which we will use on an instance of data.

And instance is a set of data, to which we are applying the methods of a class.

**Class answer: A class is like the idea of a dog, and an instance is like the dog itself. So, when you create an array, it is an instance of a class.. With a house you might have a blueprint of a house (class) – someones specific house is an instance.**

**Question 2**

**What is the difference between regular (local) variables and @instance variables?**

A local variable is only available to the method in which it was defined.

An instance variable is available to all methods within an instance of a class.

**Class answer: Local variables are only scoped to the method they are in and instance variables are available to all methods for that object.**

**Question 3**

**What does attr\_accessor do for us? What's an alternative to using attr\_accessor which would achieve the same outcome?**

attr\_accessor allows you to read and write to variables from outside an instance of a class.

You could manually write your own getter and setter methods.

def value=(something)

end

**Question 4**

**What is the purpose of the initialize method used in classes? Do you have to specify an initialize for one of your classes?**

The initialize method is the one method that is guaranteed to run when a new instance of a class is created. It can accept parameters which you can use to assist in setting up your instance. It can also run any other associated code that is necessary to instantiate the object.

It is not required. But if it is there, it will be run when you create a new instance of an object.

**Question 5**

**Assuming a class Computer exists in our program, write out the one-line code to create an instance of it and have a variable called computer pointing to that instance.**

computer = Computer.new

**Lecture – Friday April 1st 2016**

**Topics**

* Take up homework
* OOP Syntax
  + instance variables
  + Ruby's self
  + attr\_accessor
  + initialize
  + public vs private methods
  + class variables
  + class methods
* Example – imagine you’re creating a program that uses your bank account number. You want to create methods that can access it and use it but you don’t want that ability to access it anywhere in your program just some places. This is one of the key reasons objects are very useful especially when your file starts to get complicated.
* In practice, Objects just help organize these methods.
* Creating a program using your bank account
* Class method syntax --- class BankAccount --- use camelcase.
* Once created we think what are we going to do with this bank account – in some other file just require\_relative ‘bank\_account’ which is the name of the program we are working on.
* Bank account – this is good because you can see whats going on with it while working on it.
* Tech interview --- Whats the class of class ---answer…. Class
* Lets say you created a class way earlier in your code like 20 scrolls up .. you can see all of their ancestors by using – for example: in the code below you can write TFSA.ancestor and it would return [TFSA,BankAccount, Object, …….]
* Inheritance. 🡪 You created one class already and now you want to create another with the same methods as the earlier class.
* ClassTwo < ClassOne
* The super method calls a method in an class from the class they inherited the method from. – example below- It always refers to the method that you are trying to overwrite. – In the first use in the example you are overwriting the initialize method so it calls the initialize from the BankAccount class and in the second example you are overwriting deposit and the super calls the deposit method from the BankAccount class.
* Creating a method that literally puts out your variable is so common that ruby has a easy way to do it. So in the example instead of writing def balance; @balance; end – we can literally just write attr\_reader :balance and if you want to add more just comma and repeat
* Sometimes you want to directly change an instance variable from the outside. So in the example – we want to create a membership to our bank or whatever example. – we do bob = Membership.new(“Bob”, “1234”)
* In the same way, lets say this Bob wants to change his pin from 1234 to 4321. We don’t want the program to go back and change it up there. There is a way we can change it without having to go back and find it. 🡪 attr\_writer
* If we want to be able to do both of these things, read and write 🡪 we have attr\_accessor
* .
* .
* .
* when there are @@something – they belong to the class and instances
* when it says def self.foo – that belongs directly to the class.