# 1.

|  |
| --- |
| class Person  attr\_reader :name    def set\_name  @name = 'Bob'  end end  bob = Person.new p bob.name |

# What is output and why? What does this demonstrate about instance variables that differentiates them from local variables?

# 2.

|  |
| --- |
| module Swimmable  def enable\_swimming  @can\_swim = true  end end  class Dog  include Swimmable   def swim  "swimming!" if @can\_swim  end end  teddy = Dog.new p teddy.swim |

# What is output and why? What does this demonstrate about instance variables?

# 3.

|  |
| --- |
| module Describable  def describe\_shape  "I am a #{self.class} and have #{SIDES} sides."  end end  class Shape  include Describable   def self.sides  self::SIDES  end    def sides  self.class::SIDES  end end  class Quadrilateral < Shape  SIDES = 4 end  class Square < Quadrilateral; end  p Square.sides  p Square.new.sides  p Square.new.describe\_shape |

# What is output and why? What does this demonstrate about constant scope? What does `self` refer to in each of the 3 methods above?

# 4.

|  |
| --- |
| class AnimalClass  attr\_accessor :name, :animals    def initialize(name)  @name = name  @animals = []  end    def <<(animal)  animals << animal  end    def +(other\_class)  animals + other\_class.animals  end end  class Animal  attr\_reader :name    def initialize(name)  @name = name  end end  mammals = AnimalClass.new('Mammals') mammals << Animal.new('Human') mammals << Animal.new('Dog') mammals << Animal.new('Cat')  birds = AnimalClass.new('Birds') birds << Animal.new('Eagle') birds << Animal.new('Blue Jay') birds << Animal.new('Penguin')  some\_animal\_classes = mammals + birds  p some\_animal\_classes |

# What is output? Is this what we would expect when using `AnimalClass#+`? If not, how could we adjust the implementation of `AnimalClass#+` to be more in line with what we'd expect the method to return?

# 5.

|  |
| --- |
| class GoodDog  attr\_accessor :name, :height, :weight   def initialize(n, h, w)  @name = n  @height = h  @weight = w  end   def change\_info(n, h, w)  name = n  height = h  weight = w  end   def info  "#{name} weighs #{weight} and is #{height} tall."  end end   sparky = GoodDog.new('Spartacus', '12 inches', '10 lbs')  sparky.change\_info('Spartacus', '24 inches', '45 lbs') puts sparky.info  # => Spartacus weighs 10 lbs and is 12 inches tall. |

# We expect the code above to output `”Spartacus weighs 45 lbs and is 24 inches tall.”` Why does our `change\_info` method not work as expected?

# 6.

|  |
| --- |
| class Person  attr\_accessor :name   def initialize(name)  @name = name  end    def change\_name  name = name.upcase  end end  bob = Person.new('Bob') p bob.name  bob.change\_name p bob.name |

# In the code above, we hope to output `'BOB'` on `line 16`. Instead, we raise an error. Why? How could we adjust this code to output `'BOB'`?

# 7.

|  |
| --- |
| class Vehicle  @@wheels = 4   def self.wheels  @@wheels  end end  p Vehicle.wheels   class Motorcycle < Vehicle  @@wheels = 2 end  p Motorcycle.wheels  p Vehicle.wheels   class Car < Vehicle; end  p Vehicle.wheels p Motorcycle.wheels  p Car.wheels |

# What does the code above output, and why? What does this demonstrate about class variables, and why we should avoid using class variables when working with inheritance?

# 8.

|  |
| --- |
| class Animal  attr\_accessor :name   def initialize(name)  @name = name  end end  class GoodDog < Animal  def initialize(color)  super  @color = color  end end  bruno = GoodDog.new("brown")  p bruno |

# What is output and why? What does this demonstrate about `super`?

# 9.

|  |
| --- |
| class Animal  def initialize  end end  class Bear < Animal  def initialize(color)  super  @color = color  end end  bear = Bear.new("black") |

# What is output and why? What does this demonstrate about `super`?

# 10.

|  |
| --- |
| module Walkable  def walk  "I'm walking."  end end  module Swimmable  def swim  "I'm swimming."  end end  module Climbable  def climb  "I'm climbing."  end end  module Danceable  def dance  "I'm dancing."  end end  class Animal  include Walkable   def speak  "I'm an animal, and I speak!"  end end  module GoodAnimals  include Climbable   class GoodDog < Animal  include Swimmable  include Danceable  end    class GoodCat < Animal; end end  good\_dog = GoodAnimals::GoodDog.new p good\_dog.walk |

# What is the method lookup path used when invoking `#walk` on `good\_dog`?

# 11.

|  |
| --- |
| class Animal  def eat  puts "I eat."  end end  class Fish < Animal  def eat  puts "I eat plankton."  end end  class Dog < Animal  def eat  puts "I eat kibble."  end end  def feed\_animal(animal)  animal.eat end  array\_of\_animals = [Animal.new, Fish.new, Dog.new] array\_of\_animals.each do |animal|  feed\_animal(animal) end |

# What is output and why? How does this code demonstrate polymorphism?

# 12.

|  |
| --- |
| class Person  attr\_accessor :name, :pets   def initialize(name)  @name = name  @pets = []  end end  class Pet  def jump  puts "I'm jumping!"  end end  class Cat < Pet; end  class Bulldog < Pet; end  bob = Person.new("Robert")  kitty = Cat.new bud = Bulldog.new  bob.pets << kitty bob.pets << bud   bob.pets.jump |

# We raise an error in the code above. Why? What do `kitty` and `bud` represent in relation to our `Person` object?

# 13.

|  |
| --- |
| class Animal  def initialize(name)  @name = name  end end  class Dog < Animal  def initialize(name); end   def dog\_name  "bark! bark! #{@name} bark! bark!"  end end  teddy = Dog.new("Teddy") puts teddy.dog\_name |

# What is output and why?

# 14.

|  |
| --- |
| class Person  attr\_reader :name   def initialize(name)  @name = name  end end  al = Person.new('Alexander') alex = Person.new('Alexander') p al == alex # => true |

# In the code above, we want to compare whether the two objects have the same name. `Line 11` currently returns `false`. How could we return `true` on `line 11`?

# Further, since `al.name == alex.name` returns `true`, does this mean the `String` objects referenced by `al` and `alex`'s `@name` instance variables are the same object? How could we prove our case?

# 

# 15.

|  |
| --- |
| class Person  attr\_reader :name   def initialize(name)  @name = name  end   def to\_s  "My name is #{name.upcase!}."  end end  bob = Person.new('Bob') puts bob.name puts bob puts bob.name |

# What is output on `lines 14, 15, and 16` and why?

# 16.

# Why is it generally safer to invoke a setter method (if available) vs. referencing the instance variable directly when trying to set an instance variable within the class? Give an example.

# 17.

# Give an example of when it would make sense to manually write a custom getter method vs. using `attr\_reader`.

# 

# 18.

|  |
| --- |
| class Shape  @@sides = nil   def self.sides  @@sides  end   def sides  @@sides  end end  class Triangle < Shape  def initialize  @@sides = 3  end end  class Quadrilateral < Shape  def initialize  @@sides = 4  end end |

# What can executing `Triangle.sides` return? What can executing `Triangle.new.sides` return? What does this demonstrate about class variables?

# 19.

# What is the `attr\_accessor` method, and why wouldn’t we want to just add `attr\_accessor` methods for every instance variable in our class? Give an example.

# 20.

# What is the difference between states and behaviors?

# 21.

# What is the difference between instance methods and class methods?

# 22.

# What are collaborator objects, and what is the purpose of using them in OOP? Give an example of how we would work with one.

# 23.

# How and why would we implement a fake operator in a custom class? Give an example.

# 24.

# What are the use cases for `self` in Ruby, and how does `self` change based on the scope it is used in? Provide examples.

# 25.

|  |
| --- |
| class Person  def initialize(n)  @name = n  end    def get\_name  @name  end end  bob = Person.new('bob') joe = Person.new('joe')  puts bob.inspect # => #<Person:0x000055e79be5dea8 @name="bob"> puts joe.inspect # => #<Person:0x000055e79be5de58 @name="joe">  p bob.get\_name # => "bob" |

# What does the above code demonstrate about how instance variables are scoped?

# 26.

# How do class inheritance and mixing in modules affect instance variable scope? Give an example.

# 27.

# How does encapsulation relate to the public interface of a class?

# 28.

|  |
| --- |
| class GoodDog  DOG\_YEARS = 7   attr\_accessor :name, :age   def initialize(n, a)  self.name = n  self.age = a \* DOG\_YEARS  end end  sparky = GoodDog.new("Sparky", 4) puts sparky |

# What is output and why? How could we output a message of our choice instead?

# How is the output above different than the output of the code below, and why?

|  |
| --- |
| class GoodDog  DOG\_YEARS = 7   attr\_accessor :name, :age   def initialize(n, a)  @name = n  @age = a \* DOG\_YEARS  end end  sparky = GoodDog.new("Sparky", 4) p sparky |

# 29.

# When does accidental method overriding occur, and why? Give an example.

# 30.

# How is Method Access Control implemented in Ruby? Provide examples of when we would use public, protected, and private access modifiers.

# 31.

# Describe the distinction between modules and classes.

# 32.

# What is polymorphism and how can we implement polymorphism in Ruby? Provide examples.

# 33.

# What is encapsulation, and why is it important in Ruby? Give an example.

# 34.

|  |
| --- |
| module Walkable  def walk  "#{name} #{gait} forward"  end end  class Person  include Walkable   attr\_reader :name   def initialize(name)  @name = name  end   private   def gait  "strolls"  end end  class Cat  include Walkable   attr\_reader :name   def initialize(name)  @name = name  end   private   def gait  "saunters"  end end  mike = Person.new("Mike") p mike.walk  kitty = Cat.new("Kitty") p kitty.walk |

# What is returned/output in the code? Why did it make more sense to use a module as a mixin vs. defining a parent class and using class inheritance?

# 35.

# What is Object Oriented Programming, and why was it created? What are the benefits of OOP, and examples of problems it solves?

# 36.

# What is the relationship between classes and objects in Ruby?

# 37.

# When should we use class inheritance vs. interface inheritance?

# 38.

|  |
| --- |
| class Cat end  whiskers = Cat.new ginger = Cat.new paws = Cat.new |

# If we use `==` to compare the individual `Cat` objects in the code above, will the return value be `true`? Why or why not? What does this demonstrate about classes and objects in Ruby, as well as the `==` method?

# 39.

|  |
| --- |
| class Thing end  class AnotherThing < Thing end  class SomethingElse < AnotherThing end |

# Describe the inheritance structure in the code above, and identify all the superclasses.

# 40.

|  |
| --- |
| module Flight  def fly; end end  module Aquatic  def swim; end end  module Migratory  def migrate; end end  class Animal end  class Bird < Animal end  class Penguin < Bird  include Aquatic  include Migratory end  pingu = Penguin.new pingu.fly |

What is the method lookup path that Ruby will use as a result of the call to the `fly` method? Explain how we can verify this.

# 41.

|  |
| --- |
| class Animal  def initialize(name)  @name = name  end   def speak  puts sound  end   def sound  "#{@name} says "  end end  class Cow < Animal  def sound  super + "moooooooooooo!"  end end  daisy = Cow.new("Daisy") daisy.speak |

# What does this code output and why?

# 42.

|  |
| --- |
| class Cat  def initialize(name, coloring)  @name = name  @coloring = coloring  end   def purr; end   def jump; end   def sleep; end   def eat; end end  max = Cat.new("Max", "tabby") molly = Cat.new("Molly", "gray") |

# Do `molly` and `max` have the same states and behaviors in the code above? Explain why or why not, and what this demonstrates about objects in Ruby.

# 43.

|  |
| --- |
| class Student  attr\_accessor :name, :grade   def initialize(name)  @name = name  @grade = nil  end    def change\_grade(new\_grade)  grade = new\_grade  end end  priya = Student.new("Priya") priya.change\_grade('A') priya.grade |

# In the above code snippet, we want to return `”A”`. What is actually returned and why? How could we adjust the code to produce the desired result?

# 44.

|  |
| --- |
| class MeMyselfAndI  self   def self.me  self  end   def myself  self  end end  i = MeMyselfAndI.new |

# What does each `self` refer to in the above code snippet?

# 45.

|  |
| --- |
| class Student  attr\_accessor :grade   def initialize(name, grade=nil)  @name = name  end  end  ade = Student.new('Adewale') p ade # => #<Student:0x00000002a88ef8 @grade=nil, @name="Adewale"> |

# Running the following code will not produce the output shown on the last line. Why not? What would we need to change, and what does this demonstrate about instance variables?

# 46.

|  |
| --- |
| class Character  attr\_accessor :name   def initialize(name)  @name = name  end   def speak  "#{@name} is speaking."  end end  class Knight < Character  def name  "Sir " + super  end end  sir\_gallant = Knight.new("Gallant") p sir\_gallant.name  p sir\_gallant.speak |

# What is output and returned, and why? What would we need to change so that the last line outputs `”Sir Gallant is speaking.”`?

# 47.

|  |
| --- |
| class FarmAnimal  def speak  "#{self} says "  end end  class Sheep < FarmAnimal  def speak  super + "baa!"  end end  class Lamb < Sheep  def speak  super + "baaaaaaa!"  end end  class Cow < FarmAnimal  def speak  super + "mooooooo!"  end end  p Sheep.new.speak p Lamb.new.speak p Cow.new.speak |

# What is output and why?

# 48.

|  |
| --- |
| class Person  def initialize(name)  @name = name  end end  class Cat  def initialize(name, owner)  @name = name  @owner = owner  end end  sara = Person.new("Sara") fluffy = Cat.new("Fluffy", sara) |

# What are the collaborator objects in the above code snippet, and what makes them collaborator objects?

# 49.

|  |
| --- |
| number = 42  case number when 1 then 'first' when 10, 20, 30 then 'second' when 40..49 then 'third' end |

# What methods does this `case` statement use to determine which `when` clause is executed?

# 50.

|  |
| --- |
| class Person  TITLES = ['Mr', 'Mrs', 'Ms', 'Dr']   @@total\_people = 0   def initialize(name)  @name = name  end   def age  @age  end end |

# What are the scopes of each of the different variables in the above code?

# 51.

# The following is a short description of an application that lets a customer place an order for a meal:

# - A meal always has three meal items: a burger, a side, and drink.

# - For each meal item, the customer must choose an option.

# - The application must compute the total cost of the order.

# 1. Identify the nouns and verbs we need in order to model our classes and methods.

# 2. Create an outline in code (a spike) of the structure of this application.

# 3. Place methods in the appropriate classes to correspond with various verbs.

# 52.

|  |
| --- |
| class Cat  attr\_accessor :type, :age   def initialize(type)  @type = type  @age = 0  end   def make\_one\_year\_older  self.age += 1  end end |

# In the `make\_one\_year\_older` method we have used `self`. What is another way we could write this method so we don't have to use the `self` prefix? Which use case would be preferred according to best practices in Ruby, and why?

# 53.

|  |
| --- |
| module Drivable  def self.drive  end end  class Car  include Drivable end  bobs\_car = Car.new bobs\_car.drive |

# What is output and why? What does this demonstrate about how methods need to be defined in modules, and why?

# 54.

|  |
| --- |
| class House  attr\_reader :price   def initialize(price)  @price = price  end end  home1 = House.new(100\_000) home2 = House.new(150\_000) puts "Home 1 is cheaper" if home1 < home2 # => Home 1 is cheaper puts "Home 2 is more expensive" if home2 > home1 # => Home 2 is more expensive |

# What module/method could we add to the above code snippet to output the desired output on the last 2 lines, and why?

# 