LS 120 Questions

What is OOP and why is it important?	Link
What is encapsulation?	Link
How does encapsulation relate to the public interface of a class?	Link
What is an object?	Link
What is a class?	Link
What is instantiation?	Link
What is polymorphism?	Link, link
Explain two different ways to implement polymorphism.	Link
How does polymorphism work in relation to the public interface?	Link
What is duck typing? How does it relate to polymorphism - what problem does it solve?	Link
What is inheritance?	Link, link
What is the difference between a superclass and a subclass?	Link
What is a module?	Link, link
What is a mixin?	Link, link
When is it good to use inheritance?	Link #1
In inheritance, when would it be good to override a method?	Link #1
What is the method lookup path?	Link, Link #4
When defining a class, we usually focus on state and behaviors. What is the difference between these two concepts?	Link
How do you initialize a new object?	Link
What is a constructor method?	Link
What is an instance variable, and how is it related to an object?	Link
What is an instance method?	Link
How do objects encapsulate state?	Link
What is the difference between classes and objects?	Link
How can we expose information about the state of the object using instance methods?	Link
What is a collaborator object, and what is the purpose of using collaborator objects in OOP?	Link

<table>
<tr><td><b>What is an accessor method?</b></td><td>Link</td></tr>
<tr><td>What is a getter method?</td><td>Link</td></tr>
<tr><td>What is a setter method?</td><td>Link</td></tr>
<tr><td>What is attr_accessor?</td><td>Link</td></tr>
<tr><td><b>How do you decide whether to reference an instance variable or a getter method?</b></td><td>Link</td></tr>
<tr><td><pre>
class GoodDog
  attr_accessor :name, :height, :weight

  def initialize(n, h, w)
    @name = n
    @height = h
    @weight = w
  end

  def speak
    "#{name} says arf!"
  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.change_info('Spartacus', '24 inches', '45 lbs')
puts sparky.info
# => Sparky weighs 10 lbs and is 12 inches tall.

# Why does the .change_info method not work as expected here?
</pre></td><td>Link</td></tr>
<tr><td>When would you call a method with self?</td><td>Link</td></tr>
<tr><td>What are class methods?</td><td>Link</td></tr>
<tr><td>What is the purpose of a class variable?</td><td>Link</td></tr>
<tr><td>What is a constant variable?</td><td>Link</td></tr>
<tr><td>What is the default to_s method that comes with Ruby, and how do you override this?</td><td>Link</td></tr>
<tr><td>What are some important attributes of the to_s method?</td><td>Link</td></tr>
<tr><td>From within a class, when an instance method uses self, what does it reference?</td><td>Link</td></tr>
</table>

What happens when you use self inside a class but outside of an instance method?	Link
Why do you need to use self when calling private setter methods?	Link
Why use self, and how does self change depending on the scope it is used in?	Link
What is inheritance, and why do we use it?	Link
Give an example of how to use class inheritance.	Link
Give an example of overriding. When would you use it?	Link
Give an example of using the super method. When would we use it?	Link
Give an example of using the super method with an argument.	Link
When creating a hierarchical structure, under what circumstance would a module be useful?	Link
What is interface inheritance, and under what circumstance would it be useful in comparison to class inheritance?	Link
How is the method lookup path affected by module mixins and class inheritance?	Link
What is namespacing?	Link
How does Ruby provide the functionality of multiple inheritance?	Link
Describe the use of modules as containers.	Link
Why should a class have as few public methods as possible?	Link
What is the private method call used for?	Link
What is the protected keyword used for?	Link
What are two rules of protected methods?	Link
Why is it generally a bad idea to override methods from the Object class, and which method is commonly overridden?	Link
What is the relationship between a class and an object?	Link
Explain the idea that a class groups behaviors.	Link
Objects do not share state between other objects, but do share behaviors	Link
The values in the objects' instance variables (states) are different, but they can call the same instance methods (behaviors) defined in the class.	Link
Classes also have behaviors not for objects (class methods).	Link
sub-classing from parent class. Can only sub-class from 1 parent; used to model hierarchical relationships	Link
mixing in modules. Can mix in as many modules as needed; Ruby's way of	Link

<table>
<tr><td>implementing multiple inheritance</td><td></td></tr>
<tr><td>understand how sub-classing or mixing in modules affects the method lookup path</td><td>Link</td></tr>
<tr><td>

What will the following code output?

```ruby
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
```

</td><td>Link #11</td></tr>
<tr><td>

```ruby
class Person
  attr_writer :first_name, :last_name

  def full_name
    # omitted code
  end
end

mike = Person.new
mike.first_name = 'Michael'
mike.last_name = 'Garcia'
mike.full_name # => 'Michael Garcia'
```

What code snippet can replace the "omitted code" comment to produce the indicated result?

</td><td>Link #15</td></tr>
<tr><td>

```ruby
class Student
  attr_accessor :name, :grade

  def initialize(name)
    @name = name
    @grade = nil
  end
end
```

</td><td>Link #16</td></tr>
</table>

```
priya = Student.new("Priya")
priya.change_grade('A')
priya.grade # => "A"
```

The last line in the above code should return "A". Which method(s) can we add to the Student class so the code works as expected?

In the example above, why would the following not work?

```
def change_grade(new_grade)
  grade = new_grade
end
```

Link #16

On which lines in the following code does self refer to the instance of the MeMyselfAndI class referenced by i rather than the class itself? Select all that apply.

```
class MeMyselfAndI
  self

  def self.me
    self
  end

  def myself
    self
  end
end

i = MeMyselfAndI.new
```

Link #19

Given the below usage of the Person class, code the class definition.

```
bob = Person.new('bob')
bob.name                  # => 'bob'
bob.name = 'Robert'
bob.name                  # => 'Robert'
```

Link #1

Modify the class definition from above to facilitate the following methods. Note that there is no name= setter method now.

```
bob = Person.new('Robert')
bob.name                  # => 'Robert'
bob.first_name            # => 'Robert'
bob.last_name             # => ''
bob.last_name = 'Smith'
bob.name                  # => 'Robert Smith'
```

Hint: let first_name and last_name be "states" and create an instance method called name that uses those states.

Link #2

Now create a smart name= method that can take just a first name or a full name, and knows how to set the first_name and last_name appropriately.

Link #3

<table>
<tr><td><pre>
bob = Person.new('Robert')
bob.name   # => 'Robert'
bob.first_name   # => 'Robert'
bob.last_name   # => ''
bob.last_name = 'Smith'
bob.name   # => 'Robert Smith'

bob.name = "John Adams"
bob.first_name   # => 'John'
bob.last_name   # => 'Adams'
</pre></td><td></td></tr>
<tr><td>Using the class definition from step #3, let's create a few more people -- that is, Person objects.
<pre>
bob = Person.new('Robert Smith')
rob = Person.new('Robert Smith')
</pre>
If we're trying to determine whether the two objects contain the same name, how can we compare the two objects?</td><td>Link #4</td></tr>
<tr><td>Continuing with our Person class definition, what does the below print out?
<pre>
bob = Person.new("Robert Smith")
puts "The person's name is: #{bob}"
</pre></td><td>Link #5a</td></tr>
<tr><td>Let's add a to_s method to the class:
<pre>
class Person
  # ... rest of class omitted for brevity

  def to_s
    name
  end
end
</pre>
Now, what does the below output?
<pre>
bob = Person.new("Robert Smith")
puts "The person's name is: #{bob}"
</pre></td><td>Link #5b</td></tr>
<tr><td>Create an empty class named Cat.</td><td>Link</td></tr>
<tr><td>Using the code from the previous exercise, create an instance of Cat and assign it to a variable named kitty.</td><td>Link</td></tr>
<tr><td><pre>
class Wedding
  attr_reader :guests, :flowers, :songs

  def prepare(preparers)
    preparers.each do |preparer|
      case preparer
      when Chef
        preparer.prepare_food(guests)
      when Decorator
        preparer.decorate_place(flowers)
</pre></td><td>Link</td></tr>
</table>

```
      when Musician
        preparer.prepare_performance(songs)
      end
    end
  end
end

class Chef
  def prepare_food(guests)
    # implementation
  end
end

class Decorator
  def decorate_place(flowers)
    # implementation
  end
end

class Musician
  def prepare_performance(songs)
    #implementation
  end
end

# The above code would work, but it is problematic. What is wrong with
this code, and how can you fix it?
```

What happens when you call the p method on an object? And the puts method?	Link
What is a spike?	Link
When writing a program, what is a sign that you’re missing a class?	Link
What are some rules/guidelines when writing programs in OOP?	Link

```
class Student
  attr_accessor :grade

  def initialize(name, grade=nil)
    @name = name
  end
end

ade = Student.new('Adewale')
ade # => #<Student:0x00000002a88ef8 @grade=nil, @name="Adewale">
# Why does this code not have the expected return value?
```

Link #2, D

```
class Character
  attr_accessor :name

  def initialize(name)
    @name = name
  end
```

Link #4

```
  def speak
    "#{@name} is speaking."
  end
end

class Knight < Character
  def name
    "Sir " + super
  end
end

sir_gallant = Knight.new("Gallant")
sir_gallant.name # => "Sir Gallant"
sir_gallant.speak # => "Sir Gallant is speaking."
# What change(s) do you need to make to the above code in order to get
the expected output?
```

Link #6

```
class FarmAnimal
  def speak
    "#{self} says "
  end
end

class Sheep < FarmAnimal
  def speak
    super + "baa!"
  end
end

class Lamb < Sheep
  def speak
    "baaaaaaa!"
  end
end

class Cow
  def speak
    super + "mooooooo!"
  end
end

Sheep.new.speak # => "Sheep says baa!"
Lamb.new.speak # => "Lamb says baa!baaaaaaa!"
Cow.new.speak # => "Cow says mooooooo!"
# Make the changes necessary in order for this code to return the
expected values.
```

Link #8

```
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)
Identify all custom defined objects that act as collaborator objects within the code.**	

How does equivalence work in Ruby?	Link
How do you determine if two variables actually point to the same object?	Link
What is == in Ruby? How does == know what value to use for comparison?	Link
Is it possible to compare two objects of different classes?	Link
What do you get “for free” when you define a == method?	Link

**arr1 = [1, 2, 3]
arr2 = [1, 2, 3]
arr1.object_id == arr2.object_id   # => ??
 	
sym1 = :something
sym2 = :something
sym1.object_id == sym2.object_id   # => ??
 	
int1 = 5
int2 = 5
int1.object_id == int2.object_id   # => ??
# What will the code above return and why?** 	
	Link

What is the === method?	Link
What is the eql? method?	Link
What is the scoping rule for instance variables?	Link

**class Person
  def get_name
    @name   # the @name instance variable is not initialized anywhere
  end
end
 	
bob = Person.new
bob.get_name   # => ??
# What is the return value, and why?** 	
	Link

What are the scoping rules for class variables? What are the two main behaviors of class variables?	Link
What are the scoping rules for constant variables?	Link
How does sub-classing affect instance variables?	Link

<table>
<tr>
<td>

```
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 will this return, and why?
```

</td>
<td>Link</td>
</tr>
<tr>
<td>

```
module Swim
  def enable_swimming
    @can_swim = true
  end
end

class Dog
  include Swim

  def swim
    "swimming!" if @can_swim
  end
end

teddy = Dog.new
teddy.swim
# How do you get this code to return “swimming”? What does this
demonstrate about instance variables?
```

</td>
<td>Link</td>
</tr>
<tr>
<td>Are class variables accessible to sub-classes?</td>
<td>Link</td>
</tr>
<tr>
<td>Why is it recommended to avoid the use of class variables when working with inheritance?</td>
<td>Link</td>
</tr>
<tr>
<td>

```
class Vehicle
  @@wheels = 4

  def self.wheels
    @@wheels
  end
end

Vehicle.wheels                              # => ??

class Motorcycle < Vehicle
  @@wheels = 2
end
```

</td>
<td>Link</td>
</tr>
</table>

Motorcycle.wheels # => ?? Vehicle.wheels # => ?? class Car < Vehicle end Car.wheels # => ?? # What would the above code return, and why?	
Is it possible to reference a constant defined in a different class?	Link
What is the namespace resolution operator?	Link
How are constants used in inheritance?	Link

module Maintenance
  def change_tires
    "Changing #{WHEELS} tires."
  end
end
 	
class Vehicle
  WHEELS = 4
end
 	
class Car < Vehicle
  include Maintenance
end
 	
a_car = Car.new
a_car.change_tires
# Describe the error and provide two different ways to fix it. 	
	Link

What is lexical scope?	Link
When dealing with code that has modules and inheritance, where does constant resolution look first?	Link

class Person
  attr_accessor :name, :age
 	
  def initialize(name, age)
    @name = name
    @age = age
  end
End
 	
bob = Person.new("Bob", 49)
kim = Person.new("Kim", 33)
puts "bob is older than kim" if bob > kim
# How can you make this code function? How is this possible? 	
	Link

my_hash = {a: 1, b: 2, c: 3}
my_hash << {d: 4}
# What happens here, and why? 	
	Link

When do shift methods make the most sense?	Link

class Team
  attr_accessor :name, :members
 	
  def initialize(name)
    @name = name
    @members = []
  end
 	
  def <<(person)
    members.push person
  end
 	
  def +(other_team)
    members + other_team.members
  end
end
 	
# we'll use the same Person class from earlier 	
 	
cowboys = Team.new("Dallas Cowboys")
cowboys << Person.new("Troy Aikman", 48)
 	
niners = Team.new("San Francisco 49ers")
niners << Person.new("Joe Montana", 59)
dream_team = cowboys + niners   # what is dream_team?
# What does the Team#+ method currently return? What is the problem with this? How could you fix this problem? 	
	Link

Explain how the element getter (reference) and setter methods work, and their corresponding syntactical sugar.	Link
How is defining a class different from defining a method?	Link
How do you create an instance of a class?	Link
What are two different ways that the getter method allows us to invoke the method in order to access an instance variable?	Link
When you have a mixin and you use a ruby shorthand accessor method, how do you write the code (what order do you write the getter/setters and the mixin)? What about using a constant?	Link, Link
How do you define a class method?	Link

class Cat
  attr_accessor :name
 	
  def initialize(name)
    @name = name
  end
 	
  def rename(new_name)
    name = new_name
  end
	Link

end
 	
kitty = Cat.new('Sophie')
p kitty.name # "Sophie"
kitty.rename('Chloe')
p kitty.name # "Chloe"
# What is wrong with the code above? Why? What principle about getter/setter methods does this demonstrate? 		

Self refers to the ______ _______.	Link
How do you print the object so you can see the instance variables and their values along with the object?	Link
When writing the name of methods in normal/markdown text, how do you write the name of an instance method? A class method?	Link
How do you override the to_s method? What does the to_s method have to do with puts?	Link

# Using the following code, allow Truck to accept a second argument upon instantiation. Name the parameter bed_type and implement the modification so that Car continues to only accept one argument. 	
 	
class Vehicle
  attr_reader :year
 	
  def initialize(year)
    @year = year
  end
end
 	
class Truck < Vehicle
end
 	
class Car < Vehicle
end
 	
truck1 = Truck.new(1994, 'Short')
puts truck1.year
puts truck1.bed_type
	Link

# Given the following code, modify #start_engine in Truck by appending 'Drive fast, please!' to the return value of #start_engine in Vehicle. The 'fast' in 'Drive fast, please!' should be the value of speed. 	
 	
class Vehicle
  def start_engine
    'Ready to go!'
  end
end
 	
class Truck < Vehicle
  def start_engine(speed)
	Link

<table>
<tr><td><pre>
  end
end

truck1 = Truck.new
puts truck1.start_engine('fast')

# Expected output:

# Ready to go! Drive fast, please!
</pre></td><td></td></tr>
<tr><td>When do you use empty parentheses with super?</td><td>Link</td></tr>
<tr><td>How do you find the lookup path for a class? (lookup path stops when you find it)</td><td>Link, Link, Link</td></tr>
<tr><td>What is namespacing, and how do you instantiate a class contained in a module?</td><td>Link</td></tr>
<tr><td>When using getters and setters, in what scenario might you decide to only use a getter, and why is this important?</td><td>Link</td></tr>
<tr><td>When might it make sense to format the data or prevent destructive method calls changing the data by using a custom getter or setter method?</td><td>Link, Link, Link, Link, Link</td></tr>
<tr><td></td><td></td></tr>
</table>