**Todo Board Project**

In this project, we'll be creating a Todo Board to track tasks that need to be completed. You'll be creating this entire project from scratch, no skeleton or specs!

We'll be writing two phases of this project, with the second expanding functionality with a refactor:

1. A board that manages a single todo list
2. A board that manages multiple todo lists with extended features for lists and items

**Background**

Let's go over the basic structure of our application, a TodoBoard. We expect a single "user" to have their own TodoBoard. In it's final form, a TodoBoard can contain many Lists. For example, perhaps a user wants one list to track bugs that they must fix in a program and another list to track things to pick up at the grocery store. A List can contain many Items. A single Item represents a task to be done. For example, the previously mentioned grocery store list may have an Item reminding the user to buy cheese.

Begin by creating a todo\_board directory to house your project. This will be our working directory. Create the following files:

todo\_board/

├── item.rb

├── list.rb

└── todo\_board.rb

We'll produce our code in a bottom-up fashion, this means we'll begin with the "smallest" component first. In the instructions below, we'll use ::method\_name to refer to class methods and #method\_name to refer to instance methods.

**Phase 1: One true list**

Our first iteration of this project will have the TodoBoard only manage a single List which contains many Items.

**Item**

**Item::valid\_date?(date\_string)**

This method should accept a string and return a boolean indicating if it is valid date of the form YYYY-MM-DD where Y, M, and D are numbers, such as 1912-06-23. The month should be a number from 1 to 12 and the day should be a number from 1 to 31. Here are examples of the expected behavior:

Item.valid\_date?('2019-10-25') # true

Item.valid\_date?('1912-06-23') # true

Item.valid\_date?('2018-13-20') # false

Item.valid\_date?('2018-12-32') # false

Item.valid\_date?('10-25-2019') # false

For simplicity, we'll be using well-formed strings to represent our dates. However, if we were building this application for real users, then we would use a more robust solution like Ruby's [Date](https://ruby-doc.org/stdlib-2.6.1/libdoc/date/rdoc/Date.html) objects. This would allow us to implement advanced features like time of day, timezone, and other niceties. We'll save that battle for another day.

**Item#initialize(title, deadline, description)**

This method should accept a title, deadline, and a description as arguments. These three pieces of data should be stored as instance variables of an Item. If the deadline is not a valid date, raise an error.

For example, here are examples of two items being created successfully and one invalid item:

Item.new('Fix login page', '2019-10-25', 'The page loads too slow.')

Item.new(

'Buy Cheese',

'2019-10-21',

'We require American, Swiss, Feta, and Mozzarella cheese for the Happy hour!'

)

Item.new(

'Fix checkout page',

'10-25-2019',

'The font is too small.'

) # raises error due to invalid date

**Item#title**

This method should return the item's title.

**Item#title=(new\_title)**

This method should set the item's title.

**Item#deadline**

This method should return the item's deadline.

**Item#deadline=(new\_deadline)**

This method should set the item's deadline. However, if the new deadline is not a valid date, then it should raise an error and not change the deadline.

**Item#description**

This method should return the item's description.

**Item#description=(new\_description)**

This method should set the item's description.

**Checkpoint - Using pry to test**

At this point we've written all the methods we'll need for an item. Before moving on, we should test our work to avoid accumulating a deficit of buggy code that will be difficult to fix later. Let's use pry to test our work quickly.

You should cd into the project folder and run pry. Then, use load 'item.rb' to load your code into the environment, this should return true if the file was successfully loaded. From there, play around with your class by creating an instance and calling some methods on it! If you edit the file you are going to have to load it into the environment again.

Here is how we can *begin* to test our items, but you should be more thorough:

[1] pry(main)> load 'item.rb'

=> true

[2] pry(main)> my\_item = Item.new('Fix login page', '2019-10-42', 'It loads slow.')

RuntimeError: deadline is not valid

[3] pry(main)> my\_item = Item.new('Fix login page', '2019-10-22', 'It loads slow.')

=> #<Item:0x

@deadline="2019-10-22",

@description="It loads slow.",

@title="Fix login page">

[4] pry(main)> my\_item.title

=> "Fix login page"

[5] pry(main)> my\_item.description = 'It loads waaaaaay too slow!'

=> "It loads waaaaaay too slow!"

[6] pry(main)> my\_item

=> #<Item:0x

@deadline="2019-10-22",

@description="It loads waaaaaay too slow!",

@title="Fix login page">

[7] pry(main)> my\_item.deadline = "10-23-2019"

RuntimeError: deadline is not valid

[8] pry(main)> my\_item.deadline

=> "2019-10-22"

[9] pry(main)> my\_item.deadline = "2019-10-23"

=> "2019-10-23"

[10] pry(main)> my\_item.deadline

=> "2019-10-23"

[11] pry(main)> my\_item

=> #<Item:0x

@deadline="2019-10-23",

@description="It loads waaaaaay too slow!",

@title="Fix login page">

As you work through the rest of this project, be sure to test your code often and not just when we tell you to!

**List**

An instance of List may contain many items. To do this, our List will have an underlying array. A list will also have a label.

**List#initialize(label)**

This method should accept a label as an argument and store the label of the list as an instance variable. Also initialize instance variable called items to an empty array. The array will eventually contain instances ofItem.

**List#label**

This method should return the list's label.

**list#label=(new\_label)**

This method should set the list's label.

**List#add\_item(title, deadline, description)**

This method should accept a title, deadline, and optional description as arguments. This method should create a new Item with the given information and add it to the end of the items array. When no description is passed in, the new item should have an empty description. If the given deadline is not valid, then no item should be added and the method should return false. If the item is successfully added, then the method should return true. Don't forget to require your 'item.rb' file into 'list.rb'.

**List#size**

This method should return the number of items currently in the list.

**List#valid\_index?(index)**

This method should return a boolean indicating whether or not the given number is a valid position in the list. The first item of the list has index 0, the second item has index 1, and so on. A number is not a valid index if it is negative or too large given the current amount of items in the list.

Here's some food for thought. This method is similar to Item::valid\_date?. However, why did we decide to make Item::valid\_date? a class method but List#valid\_index? an instance method? The reason is that #valid\_index? requires knowledge about a specific List instance. That is, an index may be valid or invalid depending on the size of the list it is to be used on. On the flip side, valid\_date? does not require any knowledge of a specific Item. That is, we do not need to know any information about any item to know if the date is formatted correctly. Ah, such practical design decisions!

**List#swap(index\_1, index\_2)**

This method should swap the position of the items at the given indices in the list. If either index is invalid, then don't swap any items and return false. If the indices are valid, then swap the items and return true.

**Bracket method for list**List#[](index)

This method should return the item of the list that is stored at the given index. If the index is invalid, then return nil. It is important that we define access to items of the list with a custom bracket method and not expose a reader or writer for the instance variable directly (@items). This is to prevent a user from manually manipulating the list in silly ways, like adding non Item instances to the list or having empty entries in the list. We encourage users to only manipulate the list according to the rules and methods that *we* have defined.

**List#priority**

This method should return the item at the the top of the list.

**Checkpoint - Basic list functionality**

At this point, we have written a few methods that are critical to our application. Let's test these thoroughly in pry before moving on. As always, here are some basic scenarios that we used to test our List, but you should be more exhaustive:

[1] pry(main)> load 'list.rb'

=> true

[2] pry(main)> my\_list = List.new('Groceries')

=> #<List:0x @items=[], @label="Groceries">

[3] pry(main)> my\_list.size

=> 0

[4] pry(main)> my\_list.add\_item('cheese', '2019-10-25', 'Get American and Feta for good measure.')

=> true

[5] pry(main)> my\_list.add\_item('toothpaste', '2019-10-25')

=> true

[6] pry(main)> my\_list.add\_item('shampoo', '10-24-2019') # invalid date

=> false

[7] pry(main)> my\_list.add\_item('shampoo', '2019-10-24')

=> true

[8] pry(main)> my\_list.add\_item('candy', '2019-10-31', '4 bags should be enough')

=> true

[9] pry(main)> my\_list

=> #<List:0x

@items=

[#<Item:0x @title="cheese", @deadline="2019-10-25", @description="Get American and Feta for good measure.">,

#<Item:0x @title="toothpaste", @deadline="2019-10-25", @description="">,

#<Item:0x @title="shampoo", @deadline="2019-10-24", @description="">,

#<Item:0x @title="candy", @deadline="2019-10-31", @description="4 bags should be enough">],

@label="Groceries">

[10] pry(main)> my\_list.size

=> 4

[11] pry(main)> my\_list.priority

=> #<Item:0x @title="cheese", @deadline="2019-10-25", @description="Get American and Feta for good measure.">

[12] pry(main)> my\_list.swap(0, 2)

=> true

[13] pry(main)> my\_list.priority

=> #<Item:0x @title="shampoo", @deadline="2019-10-24", @description="">

[14] pry(main)> my\_list

=> #<List:0x

@items=

[#<Item:0x @title="shampoo", @deadline="2019-10-24", @description="">,

#<Item:0x @title="toothpaste", @deadline="2019-10-25", @description="">,

#<Item:0x @title="cheese", @deadline="2019-10-25", @description="Get American and Feta for good measure.">,

#<Item:0x @title="candy", @deadline="2019-10-31", @description="4 bags should be enough">],

@label="Groceries">

[15] pry(main)> my\_list.swap(1, 7) # invalid index 7

=> false

[16] pry(main)> my\_list[1]

=> #<Item:0x @title="toothpaste", @deadline="2019-10-25", @description="">

[17] pry(main)> my\_list[3]

=> #<Item:0x @title="candy", @deadline="2019-10-31", @description="4 bags should be enough">

[18] pry(main)> my\_list.swap(1, 3)

=> true

[19] pry(main)> my\_list[1]

=> #<Item:0x @title="candy", @deadline="2019-10-31", @description="4 bags should be enough">

[20] pry(main)> my\_list[3]

=> #<Item:0x @title="toothpaste", @deadline="2019-10-25", @description="">

**List#print**

This method should print the label of the list and each item's title and deadline. For usability, also print the position of each item in the list. For example, here is a nice way to format the output:

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | toothpaste | 2019-10-25

2 | shampoo | 2019-10-24

3 | candy | 2019-10-31

------------------------------------------

You don't need to spend too much time making the formatting pretty like we did above. However, if you are interested, we used [String#ljust](https://ruby-doc.org/core-2.2.0/String.html#method-i-ljust) to pad strings with spaces to ensure that the columns have equal length. We also used [string interpolation](http://ruby-for-beginners.rubymonstas.org/bonus/string_interpolation.html) to avoid tedious concatenations.

**List#print\_full\_item(index)**

This method should print all information for the item at the given index, including the title, deadline, and description. The method should not print anything if the index not valid. Here is how we formatted our output for #print\_full\_item:

------------------------------------------

candy 2019-10-31

4 bags should be enough

------------------------------------------

**List#print\_priority**

The method should print all information for the item at the top of the list.

**List#up(index, amount)**

This method should move the item at the given index up the list by continually swapping it with the item directly above it the given amount of times. If the given index is not valid, then it should return false without modifying the list. If the index is valid, then it should return true. When the specified item needs to move up further, but is already at the top of the list, then it should remain at the top. If an amount is not passed in, then move the item up 1 position.

**List#down(index, amount)**

This method should move the item at the given index down the list by continually swapping it with the item directly below it the given amount of times. If the given index is not valid, then it should return false without modifying the list. If the index is valid, then it should should return true. When the specified item needs to move down further, but is already at the bottom of the list, then it should remain at the bottom. If an amount is not passed in, then move the item down 1 position.

**Checkpoint - Up, down, all around**

The last few methods required some complex logic. Let's verify their functionality in pry before we go any further down the rabbit hole. Create a list with some items and try moving them around with #up and #down, use #print frequently to check the outcome. Here is how we began to test our list:

[1] pry(main)> load 'list.rb'

=> true

[2] pry(main)> l = List.new('Groceries')

=> #<List:0x00007fac66bedf38 @items=[], @label="Groceries">

[3] pry(main)> l.add\_item('cheese', '2019-10-25')

=> true

[4] pry(main)> l.add\_item('toothpaste', '2019-10-25')

=> true

[5] pry(main)> l.add\_item('shampoo', '2019-10-24')

=> true

[6] pry(main)> l.add\_item('candy', '2019-10-31')

=> true

[7] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | toothpaste | 2019-10-25

2 | shampoo | 2019-10-24

3 | candy | 2019-10-31

------------------------------------------

=> nil

[8] pry(main)> l.down(0)

=> true

[9] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | toothpaste | 2019-10-25

1 | cheese | 2019-10-25

2 | shampoo | 2019-10-24

3 | candy | 2019-10-31

------------------------------------------

=> nil

[10] pry(main)> l.down(0, 2)

=> true

[11] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | shampoo | 2019-10-24

2 | toothpaste | 2019-10-25

3 | candy | 2019-10-31

------------------------------------------

=> nil

[12] pry(main)> l.up(3, 10)

=> true

[13] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | candy | 2019-10-31

1 | cheese | 2019-10-25

2 | shampoo | 2019-10-24

3 | toothpaste | 2019-10-25

------------------------------------------

=> nil

[14] pry(main)> l.up(7, 3) # invalid index 7

=> false

Our lists are looking pretty great! Only one more method to implement on this class.

**List#sort\_by\_date!**

This method should sort the items in the list according to their deadlines. The method should mutate the list's underlying array.

We can use Ruby's [Array#sort\_by!](https://ruby-doc.org/core-2.5.0/Array.html#method-i-sort_by-21) to perform this task. In general, the Array#sort\_by! method is called on an array and will sort the array in place according to the criteria specified by the block. Here is an example:

dogs = [

{name: 'Dezik', age: 5},

{name: 'Tsygan', age: 6},

{name: 'Lisa', age: 3}

]

dogs.sort\_by! { |dog| dog[:age] }

p dogs #

# [ {:name=>"Lisa", :age=>3},

# {:name=>"Dezik", :age=>5},

# {:name=>"Tsygan", :age=>6} ]

dogs.sort\_by! { |dog| dog[:name] }

p dogs #

# [ {:name=>"Dezik", :age=>5},

# {:name=>"Lisa", :age=>3},

# {:name=>"Tsygan", :age=>6} ]

It's worth mentioning that when the block for Array#sort\_by! returns a number, the array will be sorted in increasing order according to those number values. When the block returns a string, the array will be sorted in "alphabetical" order according to those string values. If a string contains a numeric character, it can still be interpreted as having an "alphabetical" order. That is, the character '1' comes before the character '2', alphabetically.

Up until this point, we've been very strict in enforcing what strings are stored as deadlines in our items. Now we get reap the benefit! Here is how your method might behave:

[1] pry(main)> load 'list.rb'

=> true

[2] pry(main)> l = List.new('Groceries')

=> #<List:0x @items=[], @label="Groceries">

[3] pry(main)> l.add\_item('cheese', '2019-10-25')

=> true

[4] pry(main)> l.add\_item('toothpaste', '2019-10-25')

=> true

[5] pry(main)> l.add\_item('shampoo', '2019-10-24')

=> true

[6] pry(main)> l.add\_item('candy', '2019-10-31')

=> true

[7] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | toothpaste | 2019-10-25

2 | shampoo | 2019-10-24

3 | candy | 2019-10-31

------------------------------------------

=> nil

[8] pry(main)> l.sort\_by\_date!

=> [#<Item:0x @deadline="2019-10-24", @description="", @title="shampoo">,

#<Item:0x @deadline="2019-10-25", @description="", @title="cheese">,

#<Item:0x @deadline="2019-10-25", @description="", @title="toothpaste">,

#<Item:0x @deadline="2019-10-31", @description="", @title="candy">]

[9] pry(main)> l.print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | shampoo | 2019-10-24

1 | cheese | 2019-10-25

2 | toothpaste | 2019-10-25

3 | candy | 2019-10-31

------------------------------------------

=> nil

**TodoBoard**

This is where things comes together. The TodoBoard class will be responsible for receiving user input and performing the correct action on the stored list. In this phase of the app, the board will only manage a single list. Because of this, you may find it trivial to implement these board methods since they simply delegate responsibility to the list methods. We design it in this way in order to have an extensible class that we can easily refactor for more features in phase two.

You should be using many List methods in the following steps. There is a method to the madness!

**TodoBoard#initialize(label)**

For now, a TodoBoard will only need create a single List with the given label and store it as an instance variable.

**TodoBoard#get\_command**

This method should prompt the user to enter a command and perform the appropriate action on the list. The user may enter any of the following commands with arguments separated with spaces:

* mktodo <title> <deadline> <optional description>
  + makes a todo with the given information
* up <index> <optional amount>
  + raises the todo up the list
* down <index> <optional amount>
  + lowers the todo down the list
* swap <index\_1> <index\_2>
  + swaps the position of todos
* sort
  + sorts the todos by date
* priority
  + prints the todo at the top of the list
* print <optional index>
  + prints all todos if no index is provided
  + prints full information of the specified todo if an index is provided
* quit
  + returns false

For every command except quit, the method should return true after performing the command. We will leverage this boolean later to control if we should ask the user for another command or exit.

We recommend you use the splat operator \* to easily handle the variable number of user arguments. We'll also require a lot of conditional logic to implement the command handling. For scenarios like this, using a [case statement](https://ruby-doc.org/docs/keywords/1.9/Object.html#method-i-case) is much more preferable than a verbose chain of elsifs. For example, here is how we might write a partial #get\_command that only supports mktodo and quit:

class TodoBoard

# ...

def get\_command

print "\nEnter a command: "

cmd, \*args = gets.chomp.split(' ')

case cmd

when 'mktodo'

@list.add\_item(\*args)

when 'quit'

return false

else

print "Sorry, that command is not recognized."

end

true

end

end

A small detail we can worry about later is that the user will enter their command with spaces separating their arguments. This will cause issues if they use any spaces within the text for the todo's title or description. We'll ignore this caveat for now.

**TodoBoard#run**

This method will contain our main loop that will keep prompting the user until #get\_command returns false.

Use pry to test by calling #run on a board. Then, you can enter commands to test the app. Here is a list of commands that we entered to check out our board.

mktodo cheese 2019-10-25 foooooood

mktodo toothpaste 2019-10-23

mktodo candy 2019-10-31 sugar-free

mktodo shampoo 2019-10-25

print

sort

print

swap 0 3

print

print 1

priority

down 0

print

up 3 2

print

Here is what that workflow looked like on our TodoBoard:

[1] pry(main)> load 'todo\_board.rb'

=> true

[2] pry(main)> my\_board = TodoBoard.new('groceries')

=> #<TodoBoard:0x @list=#<List:0x @items=[], @label="groceries">>

[3] pry(main)> my\_board.run

Enter a command: mktodo cheese 2019-10-25 foooooood

Enter a command: mktodo toothpaste 2019-10-23

Enter a command: mktodo candy 2019-10-31 sugar-free

Enter a command: mktodo shampoo 2019-10-25

Enter a command: print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | toothpaste | 2019-10-23

2 | candy | 2019-10-31

3 | shampoo | 2019-10-25

------------------------------------------

Enter a command: sort

Enter a command: print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | toothpaste | 2019-10-23

1 | cheese | 2019-10-25

2 | shampoo | 2019-10-25

3 | candy | 2019-10-31

------------------------------------------

Enter a command: swap 0 3

Enter a command: print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | candy | 2019-10-31

1 | cheese | 2019-10-25

2 | shampoo | 2019-10-25

3 | toothpaste | 2019-10-23

------------------------------------------

Enter a command: print 1

------------------------------------------

cheese 2019-10-25

foooooood

------------------------------------------

Enter a command: priority

------------------------------------------

candy 2019-10-31

sugar-free

------------------------------------------

Enter a command: lower 0

Sorry, that command is not recognized.

Enter a command: down 0

Enter a command: print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | candy | 2019-10-31

2 | shampoo | 2019-10-25

3 | toothpaste | 2019-10-23

------------------------------------------

Enter a command: up 3 2

Enter a command: print

------------------------------------------

GROCERIES

------------------------------------------

Index | Item | Deadline

------------------------------------------

0 | cheese | 2019-10-25

1 | toothpaste | 2019-10-23

2 | candy | 2019-10-31

3 | shampoo | 2019-10-25

------------------------------------------

Enter a command: printerino

Sorry, that command is not recognized.

You should take a moment to test all of the different commands on your board. **You must have a fully working Phase 1 before moving onto Phase 2. Otherwise, you are going to have a bad time.**

**Phase 2: So little time, so much todo**

In this phase of the project, we'll refactor our code to support the following features:

1. marking items as done
2. deleting items
3. managing multiple lists in a single board

**Extending items and lists**

**Item#initialize(title, deadline, description)**

Refactor this existing method to also initialize a boolean instance variable to track whether or not the item is "done". An item should be not "done" by default.

**Item#toggle, List#toggle\_item(index)**

This new method should "flip" the current state the item. That is, if the item is done, then change it to not done; if it is not done, then change it to done.

Don't expose a writer method for the instance variable that tracks the done state. We want to properly encapsulate all of the data within an item. Imagine that the user is out to break your program! If they could write to the instance variable, they could assign it to some non-boolean value. Encapsulation means that the user can only manipulate the data in ways that the *programmer* intends.

We need to fully integrate this new feature into our app. Create a corresponding List#toggle\_item(index) and modify TodoBoard#get\_command to allow a user to utilize this new feature with the input command toggle <index>.

**List#print, #print\_full, #print\_priority**

Since our items contains a new field, update the output of these methods to also print out a column indicating whether or not the item is done.

**List#remove\_item(index)**

This method should permanently delete the item of the list at the given index. If the index is invalid, the method should return false. If the deletion is successful, then the method should return true. Implement this method so that there are no awkward gaps in the list after an item is removed. Allow a user to utilize this new feature by supporting the input command rm <index>.

**List#purge**

This method should remove all items that are currently marked as done. Correctly implementing this may be tricky.

**Checkpoint - Are we "done" yet?**

You know the drill. Test your code thoroughly before moving forward! Here is how we tested ours:

[1] pry(main)> load 'todo\_board.rb'

[2] pry(main)> my\_board = TodoBoard.new('cool-tech-to-learn')

=> #<TodoBoard:0x

@list=

#<List:0x

@items=[],

@label="cool-tech-to-learn">>

[3] pry(main)> my\_board.run

Enter a command: mktodo ruby 2019-12-04

Enter a command: mktodo sql 2019-12-20

Enter a command: mktodo rails 2020-01-06

Enter a command: mktodo javascript 2020-01-08

Enter a command: mktodo react 2020-02-08

Enter a command: mktodo redux 2020-02-15

Enter a command: print

-------------------------------------------------

COOL-TECH-TO-LEARN

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-12-04 | [ ]

1 | sql | 2019-12-20 | [ ]

2 | rails | 2020-01-06 | [ ]

3 | javascript | 2020-01-08 | [ ]

4 | react | 2020-02-08 | [ ]

5 | redux | 2020-02-15 | [ ]

-------------------------------------------------

Enter a command: toggle 0

Enter a command: print 0

-------------------------------------------------

ruby 2019-12-04 [✓]

-------------------------------------------------

Enter a command: toggle 0

Enter a command: print

-------------------------------------------------

COOL-TECH-TO-LEARN

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-12-04 | [ ]

1 | sql | 2019-12-20 | [ ]

2 | rails | 2020-01-06 | [ ]

3 | javascript | 2020-01-08 | [ ]

4 | react | 2020-02-08 | [ ]

5 | redux | 2020-02-15 | [ ]

-------------------------------------------------

Enter a command: toggle 0

Enter a command: toggle 2

Enter a command: toggle 3

Enter a command: print

-------------------------------------------------

COOL-TECH-TO-LEARN

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-12-04 | [✓]

1 | sql | 2019-12-20 | [ ]

2 | rails | 2020-01-06 | [✓]

3 | javascript | 2020-01-08 | [✓]

4 | react | 2020-02-08 | [ ]

5 | redux | 2020-02-15 | [ ]

-------------------------------------------------

Enter a command: rm 4

Enter a command: print

-------------------------------------------------

COOL-TECH-TO-LEARN

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-12-04 | [✓]

1 | sql | 2019-12-20 | [ ]

2 | rails | 2020-01-06 | [✓]

3 | javascript | 2020-01-08 | [✓]

4 | redux | 2020-02-15 | [ ]

-------------------------------------------------

Enter a command: purge

Enter a command: print

-------------------------------------------------

COOL-TECH-TO-LEARN

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | sql | 2019-12-20 | [ ]

1 | redux | 2020-02-15 | [ ]

-------------------------------------------------

Play with your board for a bit. It's really coming together, huh?

**Implementing support for multiple lists**

Now it's time for the big one! We've been following some important object oriented practices up until this point and now it's time to get the pay-off. Our goal is to now manage multiple Lists in our TodoBoard. Because we have separated our logic wisely thus far, we should only have to modify our boards.

**TodoBoard#initialize**

Our board should no longer initialize with a single list. Instead, initialize a board to contain an empty hash. We will allow the user to create lists at runtime. This hash will eventually contain list labels as the keys and list instances as the values. A hash is a great choice here because keys are unique, much like how we'll want the user to refer to their different lists with unique labels.

**TodoBoard#get\_command**

The entirety of this new feature has to do with how the user interfaces with their many lists, so our only task is to refactor this method. It is quite an interesting refactor though. For our existing commands, the user now needs to specify **which** list to apply the action to.

We'll leave it to you to refactor this method. Here are all of the commands you need to support (note that some of these are new):

* mklist <new\_list\_label>
  + make a new list with the given label
* ls
  + print the labels of all lists
* showall
  + print all lists
* mktodo <list\_label> <item\_title> <item\_deadline> <optional\_item\_description>
  + add a new item to the specified list with the given information
* toggle <list\_label> <item\_index>
  + toggle "done" for the specified item on the given list
* rm <list\_label> <item\_index>
  + remove the specified item on the given list
* purge <list\_label>
  + remove all "done" items on the given list
* up <list\_label> <item\_index> <optional\_amount>
  + move the specified item higher on the given list
* down <list\_label> <item\_index> <optional\_amount>
  + move the specified item higher on the given list
* swap <list\_label> <item\_index\_1> <item\_index\_2>
  + swap the positions of the specified items on the given list
* sort <list\_label>
  + sort the given list by deadline
* priority <list\_label>
  + print the all information for the item at the top of the given list
* print <list\_label> <optional\_index>
  + print all items of the given list if index is not provided
  + print the specific item of the given list if index is provided
* quit

Finish out your application by making it runnable outside of pry. Simply create and #run an instance of TodoBoard at the bottom of todo\_board.rb. Then you can execute your app with a simple ruby todo\_board.rb from the terminal.

Here is an example of our final product in action:

Enter a command: mklist groceries

Enter a command: mklist tech

Enter a command: ls

groceries

tech

Enter a command: mktodo groceries butter 2019-10-24

Enter a command: mktodo groceries milk 2019-10-23

Enter a command: mktodo groceries candy 2019-10-31 sugar-free

Enter a command: mktodo groceries toothpaste 2019-10-29

Enter a command: print groceries

-------------------------------------------------

GROCERIES

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | butter | 2019-10-24 | [ ]

1 | milk | 2019-10-23 | [ ]

2 | candy | 2019-10-31 | [ ]

3 | toothpaste | 2019-10-29 | [ ]

-------------------------------------------------

Enter a command: print groceries 2

-------------------------------------------------

candy 2019-10-31 [ ]

sugar-free

-------------------------------------------------

Enter a command: mktodo tech ruby 2019-10-24

Enter a command: mktodo tech rails 2019-11-02

Enter a command: mktodo tech javascript 2019-11-30

Enter a command: showall

-------------------------------------------------

GROCERIES

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | butter | 2019-10-24 | [ ]

1 | milk | 2019-10-23 | [ ]

2 | candy | 2019-10-31 | [ ]

3 | toothpaste | 2019-10-29 | [ ]

-------------------------------------------------

-------------------------------------------------

TECH

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-10-24 | [ ]

1 | rails | 2019-11-02 | [ ]

2 | javascript | 2019-11-30 | [ ]

-------------------------------------------------

Enter a command: toggle groceries 1

Enter a command: toggle groceries 3

Enter a command: toggle tech 2

Enter a command: showall

-------------------------------------------------

GROCERIES

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | butter | 2019-10-24 | [ ]

1 | milk | 2019-10-23 | [✓]

2 | candy | 2019-10-31 | [ ]

3 | toothpaste | 2019-10-29 | [✓]

-------------------------------------------------

-------------------------------------------------

TECH

-------------------------------------------------

Index | Item | Deadline | Done

-------------------------------------------------

0 | ruby | 2019-10-24 | [ ]

1 | rails | 2019-11-02 | [ ]

2 | javascript | 2019-11-30 | [✓]

-------------------------------------------------