Setup

Estimated time: 1+ hour

To complete this project, you'll need to make sure you have all the necessary tools at your disposal. You'll find the resources below especially useful if you are new to programming in Ruby in a desktop environment. If you're familiar with these topics and feel like you have a good handle on text editors, running Ruby programs from the command line, and a recent version of Ruby installed, feel free to breeze through and move on to the next step.

* 1. A **text editor** will be necessary to write the code that eventually will become your program. Read [this article](https://www.codecademy.com/articles/text-editor) to set up your text editor on your own computer.
  2. The **command line interface** is a program you’ll use to interact with the Ruby Todo List. Read [this article](https://www.codecademy.com/articles/command-line-interface) to get your command line interface up and running.
  3. Lastly, let’s make sure that **Ruby** is alive and well on your computer. Read[this article](https://www.codecademy.com/articles/ruby-setup)to get Ruby running on your machine.

**You Had Me At “Hello, Ruby!”**

Now it’s time to start talking to Ruby. Create a file named test.rb in Sublime Text, and write the following line: puts "Hello, Ruby!", then hit save.

Next, go to the command line, make sure you are in the same directory as test.rb, and typeruby test.rb. This tells Ruby to interpret test.rb, and to run it. You should see puts "Hello, Ruby!"” printed into the command line. You’re all set up to start creating your very own Ruby programs!

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/01.html>>

Ruby Sandbox

Estimated time: 30 minutes

Ruby Sandbox

Before we start designing the Todo List program, we’re going to clarify some foundational skills that will be helpful in this program. This step is referred to as a “sandbox” because you’re encouraged to experiment and play. The more comfortable you are with these new concepts, the easier it will be to build the Todo List application.

**Input And Output**

An essential skill in programming is reading and writing to files. This will play an important role in our Todo List program, since you’ll want to read an existing Todo List, and then add new things to it.

If you have not used Ruby's IO class before, you'll still be able to build this program. Right now is great time to understand conceptually how it works.

**Using The Command Line For User Input**

Another skill that’s important for this project is grabbing text that the user types into the command line interface, and using that to, for instance, add another task to your list. Now is a great time to brush up on Ruby's gets method for user input since it is core to many of Ruby's command line applications.

**Going Beyond The Sandbox**

If you’ve given yourself time to explore some of these new ideas in Ruby, like the IO class and the Ruby gets method, we're ready to start building.

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/02.html>>

Program Design

Estimated time: 30 minutes

Beginner programmers are often surprised to learn that program design involves very little coding. This stage is all about programmatic thinking: understanding the real-world problem your program is attempting to address, then breaking that down into small, manageable steps that a computer can solve.

**Object-Oriented Thinking - Classes, Methods, And Actions**

Now for the tough part: let’s translate our ideas into the building blocks that will become our code in the next step.

* 1. The Todo List will have a two different major constructs: lists and tasks. You can make these objects in Ruby by writing them as a classes. Remember, classes are just bundles of methods that you can make to do just about anything you want (you can review classes [here](https://www.codecademy.com/courses/ruby-beginner-en-MFiQ6/0/1?curriculum_id=5059f8619189a5000201fbcb)).
  2. The Todo List should also be able to do some essential actions for us too. This is where our user stories come in. They will help us determine what kinds of methods and actions we will write later on to make Todo List come to life.

Make An Outline Of Your Classes, Methods, And Actions

**Suggested Classes And Methods:**

* 1. List Class
  2. Create a list
  3. Add task to list
  4. Show all tasks
  5. Read a task from a file
  6. Write a list to a file
  7. Delete a task
  8. Update a task
  9. Task Class
  10. Create a task item

**Actions**

These can be mapped directly from the brainstorming you did in your outline. A number of examples for useful methods await you further down in this guide in case you forgot any.

**Single File Program Organization**

A quick word on organizing your code: The Todo List code examples in this project guide assume that you’ll be writing your program as a single file. We’ve named ours todo.rb You can of course name yours whatever makes sense to you. It’s perfectly fine for a program of this size to label different sections with comments. For example, your classes can begin under a line with the comment #Classes, and your actions (which should be written below the Classes) can start under a line that says #actions

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/03.html>>

Create A Todo List

Estimated time: 20 minutes

User Story 1:**As a user I can create a Todo List**

Houses are built brick by brick. We have our plan for Todo List, and now it is time to build. Let’s create a workflow to adhere to as we add features and iterate upon them:

**Todo List Workflow**

* 1. Build a piece, then test it, build another piece, then test it.
  2. Know that each brick is sound and stable before you put another one on top of it.

The instructions below describe just one possible approach to designing a Todo List application. Follow along, simulating our code samples in your text editor if you wish. When you have a solid grasp of the techniques illustrated here, feel free to return to your own program design and to integrate elements from this approach.

List Class Guidelines

If we think on a high level about Todo List, the most important thing it should be able to do is create a list. So, let’s build that part of the program first, test it to make sure it works as we expect, then move to the next user story.

We can accomplish this by creating a List class that makes new List instances when called with the .new method. If these concepts seem a little hazy, that’s okay. Check out [Object Oriented Programming I](https://www.codecademy.com/courses/ruby-beginner-en-MFiQ6/0/1?curriculum_id=5059f8619189a5000201fbcb) for a quick refresher.

**1.**Create a Ruby file to serve as your main code file for this project.

You can call it todo.rbor a name of your choice that makes sense to you. Remember that this will be the official filename for the program.

**2.**Create a List class.

Recall that Classes are the blueprints of each instance we create. It will maintain the attributes and behavior of the List.

**Stuck?** View solution code.

**3.**We now have a class List. We'll continue to add methods to the class, but since we’ve addressed the first user story, we have the bulk of the code we need for this step.

Beneath the class List write code to run the program. We'll refer to this as the program runner through the course of this lesson.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 my\_list = List.new  
 puts 'You have created a new list'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/04.html>>

**4.** Run the program from the command line. Recall that you can do this by typing rubyfollowed by the name of the file.

**Stuck?** View solution code.

Nice work! We've now completed the first user story; we're ready to move on the the next one.

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/04.html>>

Add Tasks

Estimated time: 1+ hour

User Story 2: **As a user I can add tasks to the todo list**

By this point you've built functionality to allow a user to create a Todo List. The next step to build the next user story, to allow a user to add tasks to the list.

Truly, what is a Todo List without tasks? You can think of the class List as that which manages the behavior of an individual list, and the class Task as that which manages the behavior of an individual task. Understanding the difference between the two will help you implement the program behaviors in a sound, logical manner.

Task Class Guidelines

**1.**Create a Task class, just as you did for List. Recall that the class will manage the behavior and attributes of each Task.

**Stuck?** View solution code.

**2.**The Task class will allow us to create instances of Task class objects. In order to give tasks a description like "Make Breakfast" or "Wash Breakfast Dishes", we'll need to store this as an attribute of a task.

Create an attribute reader method for the description so we can easily find out what any task’s description is. Recall that [attribute methods](https://www.codecademy.com/courses/ruby-beginner-en-zfe3o/0/5?curriculum_id=5059f8619189a5000201fbcb) are used to read and/or write attributes for each instance of a class.

**Stuck?** View solution code.

class Task

attr\_reader :description  
   
 ...

end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

**3.**It would be nice if a task had a description upon creation. Create an initialize method that initializes each task object with a description.

**Stuck?** View solution code.

class Task

def initialize(description)  
 @description = description  
 end

end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

This is a terrific start. We have a class Task, an attr\_reader method, and an initialize method.

**4.**Now that we have a class Task, how do we add tasks to a list? Again, while a class Taskmanages actions that can be implemented on a task object, the class List handles actions implemented on a List object.

In the class List, create an initialize method that will initialize a List object with an array that stores all tasks called all\_tasks.

**Stuck?** View solution code.

class List

def initialize  
 @all\_tasks = []  
 end

end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

**5.**Add an attribute reader that will allow you to read the attribute that stores all tasks.

**Stuck?** View solution code.

class List  
 attr\_reader :all\_tasks

...

end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

**6.**In the class List, create an add method. The method should add a task to the array that stores all tasks.

**Stuck?** View solution code.

def add(task)  
 all\_tasks << task  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

Nice work. We can use the add method to add a new Task object in the program.

**7.** Complete the program runner so that it includes the add method. Call the add method on the list, instantiating a new instance of Task class.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 my\_list = List.new  
 puts 'You have created a new list'  
 my\_list.add(Task.new('Make Breakfast')  
 puts 'You have added a task to the Todo List'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

**Stuck?** View solution code.

**8.**Run the program from the command line. You now have the ability to add a task to a list.

Feel free to add more tasks of your own, such as 'Wash Breakfast Dishes' or 'Have little brother Billy wash breakfast dishes while I play outside.' It's your program; be sure to have fun with it!

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/05.html>>

See All Tasks

Estimated time: 1+ hour

User Story 3: **As a user I can see all the tasks in a list**

Our Todo List now allows a user to create a list and add tasks to the list. Wouldn't it be nice if we could see all the tasks in the list, through a simple command line action?

We can do this by creating a show method. The attribute methods we wrote in the last section will also be really helpful when creating this method.

Show All Tasks Guidelines

**1.**In class List create a show method that displays all tasks. Feel free to refer to the attribute methods you created earlier for the List class.

**Stuck?** View solution code.

Recall that attribute methods allow us to literally read the attribute whenever we call the method.

class List

def show  
 all\_tasks  
 end

...

end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/06.html>>

**2.**Add the show method to the program runner.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 my\_list = List.new  
 puts 'You have created a new list'  
 my\_list.add(Task.new('Make Breakfast')  
 puts 'You have added a task to the Todo List'  
 puts my\_list.show  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/06.html>>

Notice the difference in how you call the show method in comparison to the add method.

Alternatively, the following is a more rigorous way to test that the tasks are showing as a string instead of a Task object.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 my\_list = List.new  
 puts 'You have created a new list'  
 my\_list.add(Task.new('Make Breakfast'))  
 my\_list.add(Task.new('Wash Breakfast Dishes'))  
 my\_list.add('Send Birthday Gift to Mom')  
 puts 'You have added a task to the Todo List'  
 if my\_list.show.join.include?('#<')  
 print [  
 'Are you sure you are handling your task object correctly for showing',  
 "as a string?\n"  
 ]  
 end  
 puts 'Your task list:'  
 puts my\_list.show  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/06.html>>

Keep in mind that any place we have access to a Task object, we also have access to all its properties (i.e., all its instance variables and methods). We would not be able to do that if tasks were only stored as strings. This is crucial for us! It will allow us to bring the user story to life in the code.

**3.**Go ahead and run the program from the command line. You should be able to see all tasks in the list, just as expected. Isn't that neat?

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/06.html>>

Menu

Estimated time: 2+ hours

User Story 4: **As a user, I can perform actions on the Todo List through a menu.**

So far we're create the ability for a user to create a Todo List, add tasks a list, and see all tasks in a list. We've done this by creating a class List and a class Task and adding methods to these classes for our desired behavior.

Every great command line application has a nice user interface -- a way to engage with the program. The next user story is to build a menu system so that as a user, one can easily manipulate the program through the command line.

We will need the following:

* + A way to display options through a menu
  + A way to prompt the user and accept user input
  + A way to call methods in the program based on user input

Menu Guidelines

**1.**Let's start by creating a module Menu. This can go at the beginning of the program, above the class List

**Stuck?** View solution code.

Recall that modules can help organize related methods and constants. They are similar to classes in how they are declared--the key difference being that you can not create objects from a module nor can you subclass a module. Modules essentially are meant to store behavior, which can be used in classes or the program runner.

**2.**In the Menu module, create a menu method that displays a string. The string should be a message that welcomes the user and displays a menu of options.

The menu can be a simple ordered list enclosed in the string. Each option on the list should map to a number. The menu options should display options to 1) Add, 2) Show and Q) Quit the program.

**Stuck?** View solution code.

**3.**Now that the user has a method to choose an action, we will also want to give the user a way to input information into the program.

Create a method show that displays the menu you created in the previous step.

**Stuck?** View solution code.

We now have a menu as part of the program's library.

**4.**Now that we have a module to display the Menu, let's create another module that prompts the user for input. Create a module Promptable

**Stuck?** View solution code.

**5.**In the module, write a prompt method. The method should take two arguments, a default message and a default symbol for the prompt. You can use 'What would you like to do?'as a message and ':> ' as a symbol for the prompt.

The method should display the message and the symbol, and accept user input.

**Stuck?** View solution code.

**6.**We now have a module Menu to display a menu, and a module Promptable that prompts the user for input and receives input from the user.

To make the Todo List program a real command line application, we will want to change the program runner to make use of our new Menu and Promptable modules, making the program interactive.

At present the program instantiates a new list when the program is run, as such:

**Stuck?** View solution code.

We can also make it such the menu uses a case statement to solicit user input. Let's see what the structure of this looks like in pseudo code, as shown in italics. We'll then plug in the right pieces.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 my\_list = List.new  
 *Include the Menu and Promptable modules*  
 *Show a nice message to the user*  
 *Until the user quits, display the menu and prompt the user for   
 their input*  
 *case*  
 *when user enters '1'*  
 *add a new task, based on the user input*  
 *when user enters '2'*  
 *show the list*  
 *else*  
 *end*   
 end   
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

**7.**We'll build this up, piece by piece. Let's start by including the Menu and Promptablemodules in the program runner.

**Stuck?** View solution code.

This will not give us any output yet if we run the code, so let's continue.

**8.**Beneath the Menu and Promptable modules, instantiate a new list and set it equal to a list name of your choice.

Below this, display a message to the user: 'Please choose from the following list:'

**Stuck?** View solution code.

**9.**Write an until statement that sets user input equal to user\_input using the module Promptable's show method until the user presses Q or q.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 end  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

**10.**Lets now start the case statement for the user\_input variable we set.

When the user enters 1 the menu should call on the add method. Recall that add method should pass a task object into an array of tasks. To pass in a new task, we must use Task.newin conjunction with the add method.

Since this step is a little challenging, be sure to compare your answer to the code given below.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you would   
 like to accomplish?')))  
 end  
 end  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

**11.**Continue with the case statement. When the user enters 2, the runner should display theshow method.

Let's also go ahead and add an else statement before the end of the case statement that displays a message to the user that says, 'Sorry, I did not understand'. This should follow the pattern of "when 1 ... when 2 ... else ... end".

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you would   
 like to accomplish?')))  
 when '2'  
 puts my\_list.show  
 else  
 puts 'Sorry, I did not understand'  
 end  
 end  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

s

**12.**Finally, after the final end in the until block, let's create an outro that displays a message to the user 'Outro - Thanks for using the menu system!'.

This will appear when the user enters the option to quit the menu.

**Stuck?** View solution code.

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you would   
 like to accomplish?')))  
 when '2'  
 puts my\_list.show  
 else  
 puts 'Sorry, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

**13.**Run the program from the command line. Go ahead and test it out by adding two tasks to the list, and then showing the tasks. Once you have run the program you can also test the Quit option

You can view the full solution for this user story below.

**Stuck?** View solution code.

module Menu

def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 Q) Quit "  
 end

def show  
 menu  
 end  
 end

module Promptable  
 def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def show  
 all\_tasks  
 end  
 end

class Task  
 attr\_reader :description

def initialize(description)  
 @description = description  
 end

end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 ml = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 ml.add(Task.new(prompt('What is the task you would like to accomplish?')))  
 when '2'  
 puts ml.show  
 else  
 puts 'Try again, I did not understand.'  
 end  
 end  
 puts 'Outro - Thanks for using our awesome menuing system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/07.html>>

Secondary User Stories

Estimated time: 2+ hours

User Story 5: **As a user I can write a list to a text file.**

Congratulations, you finished all the core user stories. It's now time to push ourselves a little further and work on the secondary user stories. In the following sections, we focus on making it so that users can write a list to a text file, read a list from a text file, delete a task from a list, and update a task on a list.

It's important to remember that each individual program is as unique, and no one solution is right. If you would like to try a solution of your own, feel free to. The instructions here are guidelines that will take you to a given implementation, but the possibilities of how you build this program, are endless.

Write A List To A File Guidelines

**1.**Let's first update the menu so that it includes the ability to write to a file.

In the menu method, modify the options such that a user can "Write to a File". You can make this option 3.

**Stuck?** View solution code.

module Menu

def menu  
 " Welcome to the TodoList Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Write to a File   
 Q) Quit "  
 end

end

**2.**We'll need to set the program up, so that when we write a task to the file, we can easily read the description of a task.

Write a method to\_s that returns a task's description as a string.

**Stuck?** View solution code.

**3.**Create a method write\_to\_file that allows a user of the Todo List program to write tasks to a file. It should take the filename as an argument and output the description of the tasks in the all tasks array as a list.

As a hint, use a Ruby IO class that allows you to write to a file.

**Stuck?** View solution code.

**4.**Add the write\_to\_file method in the program runner as a third option in the case statement. Note that the write\_to\_file method will take an argument.

**Stuck?** View solution code.

**5.**Ah ha! Since the write\_to\_file method takes a filename as an argument, let's use theprompt method to ask the username for a filename to write to.

If you recall, the prompt method by default takes a message and a symbol as an argument. It prints both of these to the user, and receives user input, as shown here:

def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end

**Stuck?** View solution code.

**6.**Run the program from the comment line and do the following:

* + Add two tasks of your choice.
  + Tell the program which filename you would like to write to.

Nice job! You just created the feature in the Todo List program that allow you to write to a file. If we want to read the list from the file, we can do this in the next step. A solution for this user story is given below.

module Menu  
 def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Write to File  
 Q) Quit "  
 end

def show  
 menu  
 end  
 end

module Promptable  
 def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def show  
 all\_tasks  
 end

def write\_to\_file(filename)  
 IO.write(filename, @all\_tasks.map(&:to\_s).join("\n"))  
 end  
 end

class Task  
 attr\_reader :description

def initialize(description)  
 @description = description  
 end

def to\_s  
 description  
 end  
 end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you would   
 like to accomplish?')))  
 when '2'  
 puts my\_list.show  
 when '3'  
 my\_list.write\_to\_file(prompt('What is the filename   
 to write to?'))  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/08.html>>

Read A Todo List

Estimated time: 1+ hour

User Story 6: **As a user I can read a Todo List from a text file.**

We have a Todo List program that allows us to write a list of tasks to a file of our choice. Pretty cool, isn't it?

But it would not make much sense to have a program that allows us to make a Todo List on a file we can't read. Those dishes, for one, would never get done! And boy, I'm sure grandma will not be happy about that when she comes over for a game of poker. We'll need a way to read a list of tasks from a file. And since you're great at working with files, this part should be a breeze.

Read A List From A File Guidelines

**1.**Let's begin by adding theRead from a Fileoption the Menu module.

We added it as an option 4. You may feel free to add it to a different place in the menu, but you'll need to be mindful of how you number the options for the program runner.

**Stuck?** View solution code.

**2.**Create a read\_from\_file method that allows a user of the Todo List program to read tasks from a file.

As a hint, refer back to the class you used for the write\_to\_file method.

**Stuck?** View solution code.

The IO class's public class method readlines takes a file as an argument. Altogether,IO.readlines(filename) is an array of all lines in the file called filename.

We can iterate through the lines in the file, and with each line instantiate a new task object.

**3.**Let's add the read\_from\_file method the program runner.

Similar to the write\_to\_file method, the read\_from\_file method takes a filename as an argument, so we will need to prompt users for a filename to read from.

**Stuck?** View solution code.

**4.**Just like in the write\_to\_file method, the read\_from\_file method needs a file to read from.

Pass the prompt method into the read\_from\_file method as a default argument. Recall that the prompt method will need to request for the appropriate user input.

**Stuck?** View solution code.

**5.**You can also add an optional step to catch filenames that are entered incorrectly.

**Stuck?** View solution code.

**6.**Run the program from the command line. Choose the Read from File option from the menu. When asked about which filename to read from, you can choose the name of the file you just added tasks to. (Alternatively, you can choose the Write to File option and Read from File option again, from scratch.) A solution for this user story is given below.

**Stuck?** View solution code.

module Menu

def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Write to File  
 4) Read from File  
 Q) Quit "  
 end

def show  
 menu  
 end  
   
 end

def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
   
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 unless task.to\_s.chomp.empty?  
 all\_tasks << task  
 end  
 end

def show  
 all\_tasks  
 end

def write\_to\_file(filename)  
 IO.write(filename, @all\_tasks.map(&:to\_s).join("\n"))  
 end

def read\_from\_file(filename)  
 IO.readlines(filename).each do |line|  
 add(Task.new(line.chomp))  
 end  
 end  
 end

class Task  
 attr\_reader :description

def initialize(description)  
 @description = description  
 end

def to\_s  
 description  
 end  
 end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(Menu.show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you   
 would like to accomplish?')))  
 when '2'  
 puts my\_list.show  
 when '3'  
 my\_list.write\_to\_file(prompt 'What is the filename to   
 write to?')  
 when '4'  
 begin  
 my\_list.read\_from\_file(prompt('What is the filename to   
 read from?'))  
 rescue Errno::ENOENT  
 puts 'File name not found, please verify your file name   
 and path.'  
 end  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/09.html>>

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/09.html>>

Delete A Task

Estimated time: 1+ hour

User Story 7: **As a user I can delete a task from a list**

Let's continue with the secondary user stories. The next user story is to delete a task from a list. How can we do this?

We'll need to be able to identify the specific task that a user can delete. The easiest way to do this is to number the tasks, and then write a method to delete a task from the array by using the task's number.

Delete A Task From A List Guidelines

**1.**Let's first change the menu, such that we give users the option to delete a task from a Todo List.

Add the Delete option to the menu. We've added it as option 3, and shifted each option down accordingly, such as 4) Write to a File and 5) Read from a File.

**Stuck?** View solution code.

**2.**Nice work. We will need a way to identify the task by a number. We currently have a showmethod, but it does not give us a way to select a task.

Modify the show method, such that by iterating through the all\_tasks array, tasks are listed in the following format:

1) Make Breakfast

2) Wash Dishes

**Stuck?** View solution code.

**3.**Next write a delete method. The method should accept a task\_number as an argument, and delete the task that the user selected from the all\_tasks array.

If you are a little unsure how to implement this method, recall two things. First, note the appropriate Ruby methods that allow you to delete an element from an array. Second, recall that arrays start at an index of 0.

**Stuck?** View solution code.

**4.**We're almost there! Modify the program runner such that when a user selects option 4, the numbered task list is shown and users are given the option to delete a task.

In this step, be sure that the case statement corresponds to the appropriate menu option.

**Stuck?** View solution code.

**5.**Run the program from the command line. Choose the option to Delete a task from the list, and go ahead and delete the task of your choice. A solution to this user story is provided below.

**Stuck?** View solution code.

module Menu  
 def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Delete  
 4) Write to File  
 5) Read from File  
 Q) Quit "  
 end

def show  
 menu  
 end  
 end

module Promptable  
 def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def delete(task\_number)  
 all\_tasks.delete\_at(task\_number - 1)  
 end

def show  
 all\_tasks.map.with\_index { |l, i| "(#{i.next}): #{l}"}  
 end

def write\_to\_file(filename)  
 IO.write(filename, @all\_tasks.map(&:to\_s).join("\n"))  
 end

def read\_from\_file(filename)  
 IO.readlines(filename).each do |line|  
 add(Task.new(line.chomp))  
 end  
 end  
 end

class Task  
 attr\_reader :description

def initialize(description)  
 @description = description  
 end

def to\_s  
 description  
 end  
 end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 ml = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 ml.add(Task.new(prompt('What is the task you would like to accomplish?')))  
 when '2'  
 puts ml.show  
 when '3'  
 puts ml.show  
 ml.delete(prompt('Which task to delete?').to\_i)  
 when '4'  
 ml.write\_to\_file(prompt 'What is the filename to write to?')  
 when '5'  
 begin  
 ml.read\_from\_file(prompt('What is the filename to read from?'))  
 rescue Errno::ENOENT  
 puts 'File name not found, please verify your file name and path.'  
 end  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the awesome menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/10.html>>

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/10.html>>

Update A Task

Estimated time: 1+ hours

User Story 8: **As a user I can update a task on a list.**

The next secondary user story is to update a task from a list.You'll notice that this will not be too different from the last step. As a bonus, we already have the tasks numbered, so we'll simply need to build in the update functionality.

Update A Task On A List Guidelines

**1.**Change the menu, such that we give users the option to update a task on a Todo List.

Add the Update option to the menu. We've added it as option 3, and shifted each task down appropriately.

**Stuck?** View solution code.

**2.**Add an update method. The method should take a task\_number and a task as an argument.

The method should set the appropriate task in the all\_tasks array equal to the new task.

**Stuck?** View solution code.

**3.**You're on a roll! Modify the program runner such that it includes the update statement.

Recall that the update statement will take two arguments, a task number (which you'll have to prompt from the user) and a task (which will get saved as the task description.

Be sure that the case statement corresponds to the appropriate menu option.

**Stuck?** View solution code.

**4.**Run the program from the command line. Choose the Update option from the menu, and update a task of your chouce. A solution to this user story is provided below.

**Stuck?** View solution code.

module Menu  
 def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Update  
 4) Delete  
 5) Write to File  
 6) Read from File  
 Q) Quit "  
 end

def show  
 menu  
 end  
 end

module Promptable  
 def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def delete(task\_number)  
 all\_tasks.delete\_at(task\_number - 1)  
 end

def update(task\_number, task)  
 all\_tasks[task\_number - 1] = task  
 end

def show  
 all\_tasks.map.with\_index { |l, i| "(#{i.next}): #{l}"}  
 end

def write\_to\_file(filename)  
 IO.write(filename, @all\_tasks.map(&:to\_s).join("\n"))  
 end

def read\_from\_file(filename)  
 IO.readlines(filename).each do |line|  
 add(Task.new(line.chomp))  
 end  
 end  
 end

class Task  
 attr\_reader :description

def initialize(description)  
 @description = description  
 end

def to\_s  
 description  
 end  
 end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Menu  
 include Promptable  
 ml = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 ml.add(Task.new(prompt('What is the task you would like   
 to accomplish?')))  
 when '2'  
 puts ml.show  
 when '3'  
 ml.update(prompt('Which task to update?').to\_i,   
 Task.new(prompt('Task Description?')))  
 when '4'  
 puts ml.show  
 ml.delete(prompt('Which task to delete?').to\_i)  
 when '5'  
 ml.write\_to\_file(prompt 'What is the filename to   
 write to?')  
 when '6'  
 begin  
 ml.read\_from\_file(prompt('What is the filename to   
 read from?'))  
 rescue Errno::ENOENT  
 puts 'File name not found, please verify your file   
 name and path.'  
 end  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the awesome menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/11.html>>

Enable A Task Status

Estimated time: 2+ hours

User Story 8: **As a user I can enable a status for a task.**

Congratulations! You've already made it through the primary and secondary user stories. If you are new to Ruby, you deserve a pat on the back. You have a fully functioning Todo List application.

We've added some additional reach user stories for those thirsty for a challenge. If you have made it this far, you are certainly capable of doing the reach user stories. If you feel confused, give yourself time to think about the best solution, and you'll surely create it.

For the reach user stories, we'll be enabling a task to have a status marking it complete or incomplete.

Enable A Task Status Guidelines

**1.**In the class Task, take a look at the initialize method. Give the class Task the ability to initialize a new task with a status.

The default status should be set to false. Note that the status is a boolean value.

**Stuck?** View solution code.

**2.**Go ahead and give the class Task an attribute accessor for the status.

**Stuck?** View solution code.

**3.**Write a method that will instantly let us know if a task is completed.

You may call this method completed? and it displays the status, which is again, a boolean value.

**Stuck?** View solution code.

**4.**Nice! We now have a way to call the completed? method on a Task object to check its completed status. We'll still need a way to represent this visually in the program.

In the class Task, create a private method called represent\_status. The method should display an '[X]' if the task is completed and a '[ ]' if the task is not completed.

You can do this with an if..else statement or a ternary operators.

s

**Stuck?** View solution code.

Nice work! We have most of the pieces in place, but since we'll be using the read and write methods for the task's status.

**5.**Write a method to\_machine. This method display both the represent\_status method and the description, separated by a colon. When called, the output of this method should look like:

[X] : Make Breakfast  
 [X] : Wash Dishes  
 [ ] : Dry Dishes

**Stuck?** View solution code.

**6.**You're doing great! Modify the write\_to\_file method, such that it outputs each of the tasks on a new line. You can use the write method in the IO class. This method can take a both filename and a string as arguments.

The string should output each task in the all\_tasks array on a new line. The catch is that we will need to use the to\_mashine method on each task we output.

It is also helpful to save the string as a variable, before passing it into IO.write.

**Stuck?** View solution code.

**7.**We're almost there! Modify the read\_to\_file method. The read from file method should read each line from the file, displaying both the description and the status.

**Stuck?** View solution code.

Nice work! We won't run our program just yet. But we have all the pieces we need to enable a task status. A solution to this user story is provided below.

**Stuck?** View solution code.

module Menu  
 def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Update  
 4) Delete  
 5) Write to File  
 6) Read from File  
 Q) Quit "  
 end

def show  
 menu  
 end  
 end

module Promptable  
 def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end  
 end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def delete(task\_number)  
 all\_tasks.delete\_at(task\_number - 1)  
 end

def update(task\_number, task)  
 all\_tasks[task\_number - 1] = task  
 end

def show  
 @all\_tasks.map.with\_index { |l, i| "(#{i.next}): #{l}" }  
 end

def write\_to\_file(filename)  
 machinified = @all\_tasks.map(&:to\_machine).join("\n")  
 IO.write(filename, machinified)  
 end

def read\_from\_file(filename)  
 IO.readlines(filename).each do |line|  
 status, \*description = line.split(':')  
 status = status.include?('X')  
 add(Task.new(description.join(':').strip, status))  
 end  
 end  
 end

class Task  
 attr\_reader :description  
 attr\_accessor :completed\_status

def initialize(description, completed\_status = false)  
 @description = description  
 @completed\_status = completed\_status  
 end

def to\_s  
 "#{represent\_status} : #{description}"  
 end

def completed?  
 completed\_status  
 end

def to\_machine  
 "#{represent\_status}:#{description}"  
 end

private

def represent\_status  
 "#{completed? ? '[X]' : '[ ]'}"  
 end

end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Promptable  
 include Menu  
 ml = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 ml.add(Task.new(prompt('What is the task you would like to accomplish?')))  
 when '2'  
 puts ml.show  
 when '3'  
 ml.update(prompt('Which task to update?').to\_i, Task.new(prompt('Task Description?')))  
 when '4'  
 puts ml.show  
 ml.delete(prompt('Which task to delete?').to\_i)  
 when '5'  
 ml.write\_to\_file(prompt 'What is the filename to write to?')  
 when '6'  
 begin  
 ml.read\_from\_file(prompt('What is the filename to read from?'))  
 rescue Errno::ENOENT  
 puts 'File name not found, please verify your filename and path.'  
 end  
 when '7'  
 puts ml.show  
 ml.toggle(prompt('Which would you like to toggle the status for?').to\_i)  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the awesome menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/12.html>>

Toggle A Task Status

Estimated time: 1+ hour

User Story 10: **As a user I can toggle a status as complete or incomplete.**

We're now at the final reach user story, the ability to toggle a status as complete or incomplete.

Toggle A Task Status Guidelines

**1.**Let's first add the toggle option to the menu in the module Menu. We've added it as option 7, before the Q option.

**Stuck?** View solution code.

def menu  
 " Welcome to the TodoList Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Update  
 4) Delete  
 5) Write to File  
 6) Read from File  
 7) Toggle Status  
 Q) Quit "  
 end

**2.**In class Task create a method toggle\_status. The method should negate thecompleted? method, and set it to the new instance variable status.

**Stuck?** View solution code.

**3.**Nice. In the class List create a method toggle. The method should `task\_number` as an argument.

The method will call the toggle\_status method on the appropriate task from the all\_tasksarray.

**Stuck?** View solution code.

An alternative implementation of this method is below.

**Stuck?** View solution code.

        def toggle(task\_number)  
          puts "In List we change a task of   
 #{all\_tasks[task\_number - 1].inspect}"  
          all\_tasks[task\_number - 1].toggle\_status  
          puts "Now our task has this information   
 #{all\_tasks[task\_number - 1].inspect}"  
         end

**4.**Nice, let's go ahead and add the toggle option to the menu. Be sure that it aligns with the menu option you created in the Menu Module. Also be sure to prompt for user input.

**Stuck?** View solution code.

**5.**Run the program from the command line. Feel free to add tasks to the list, show the tasks on the list, toggle the task status of tasks from the command line. A full solution to this user story - and the entire program we build - is provided below.

**Stuck?** View solution code.

module Menu

def menu  
 " Welcome to the TodoLister Program!  
 This menu will help you use the Task List System  
 1) Add  
 2) Show  
 3) Update  
 4) Delete  
 5) Write to File  
 6) Read from File  
 7) Toggle Status  
 Q) Quit "  
 end

def show  
 menu  
 end

end

module Promptable

def prompt(message = "Just the facts, ma'am.", symbol = ':> ')  
 print message  
 print symbol  
 gets.chomp  
 end

end

class List  
 attr\_reader :all\_tasks

def initialize  
 @all\_tasks = []  
 end

def add(task)  
 all\_tasks << task  
 end

def delete(task\_number)  
 all\_tasks.delete\_at(task\_number - 1)  
 end

def update(task\_number, task)  
 all\_tasks[task\_number - 1] = task  
 end

def show  
 all\_tasks.map.with\_index { |l, i| "(#{i.next}): #{l}" }  
 end

def write\_to\_file(filename)  
 machinified = @all\_tasks.map(&:to\_machine).join("\n")  
 IO.write(filename, machinified)  
 end

def read\_from\_file(filename)  
 IO.readlines(filename).each do |line|  
 status, \*description = line.split(':')  
 status = status.downcase.include?('x')  
 add(Task.new(description.join(':').strip, status))  
 end  
 end

def toggle(task\_number)  
 all\_tasks[task\_number - 1].toggle\_status  
 end  
   
end

class Task  
 attr\_reader :description  
 attr\_accessor :completed\_status

def initialize(description, completed\_status = false)  
 @description = description  
 @completed\_status = completed\_status  
 end

def to\_s  
 "#{represent\_status} : #{description}"  
 end

def completed?  
 completed\_status  
 end

def toggle\_status  
 @completed\_status = !completed?  
 end

def to\_machine  
 "#{represent\_status}:#{description}"  
 end

private

def represent\_status  
 completed? ? '[X]' : '[ ]'  
 end

end

if \_\_FILE\_\_ == $PROGRAM\_NAME  
 include Promptable  
 include Menu  
 my\_list = List.new  
 puts 'Please choose from the following list'  
 until ['q'].include?(user\_input = prompt(show).downcase)  
 case user\_input  
 when '1'  
 my\_list.add(Task.new(prompt('What is the task you   
 would like to accomplish?')))  
 when '2'  
 puts my\_list.show  
 when '3'  
 my\_list.update(prompt('Which task to update?').to\_i,   
 Task.new(prompt('Task Description?')))  
 when '4'  
 puts my\_list.show  
 my\_list.delete(prompt('Which task to delete?').to\_i)  
 when '5'  
 my\_list.write\_to\_file(prompt 'What is the filename to   
 write to?')  
 when '6'  
 begin  
 my\_list.read\_from\_file(prompt('What is the filename to   
 read from?'))  
 rescue Errno::ENOENT  
 puts 'File name not found, please verify your file name   
 and path.'  
 end  
 when '7'  
 puts my\_list.show  
 my\_list.toggle(prompt('Which would you like to toggle the   
 status for?').to\_i)  
 else  
 puts 'Try again, I did not understand.'  
 end  
 prompt('Press enter to continue', '')  
 end  
 puts 'Outro - Thanks for using the menu system!'  
 end

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/13.html>>

Congratulations! You just completed the Todo List program, an object oriented program that uses Ruby classes, modules, and methods, and some of the core features of the Ruby language.

We created this program first, by building up our primary user stories to create a list, add a task to the list, see all tasks in the list, and to be able to manipulate the list through a menu system. We then built out the secondary user stories, which include the ability to write a list to a file, read a list from a file, delete a task from a list, and update a task in a list. Finally, we implemented the reach user stories, which include the ability to enable a task status and toggle a task as complete or incomplete. We now have a functioning Todo List application. Nice work!

From <<https://s3.amazonaws.com/codecademy-content/projects/FinalProjects/ruby/13.html>>