$ lister

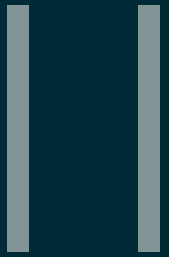
- It makes lists

$ lister

- Lister makes lists
- adds items to lists
- Removes items from lists
- Saves lists
- Loads lists that have been saved

$ lister

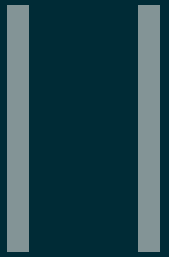
- You're working in a terminal, focused
  - You have a great idea
    - You fire up a text editor
      - Your file is now lost in the abyss of "My Documents"
      - …or the desktop



$ lister

- You switch to your note-making app
  - You fiddle with other ideas
  - You play with formatting
    - You start researching on the internet

      …You're now scrolling social media



$ lister

- You type:

```
$ lister -a “App Idea” “It makes lists”
```

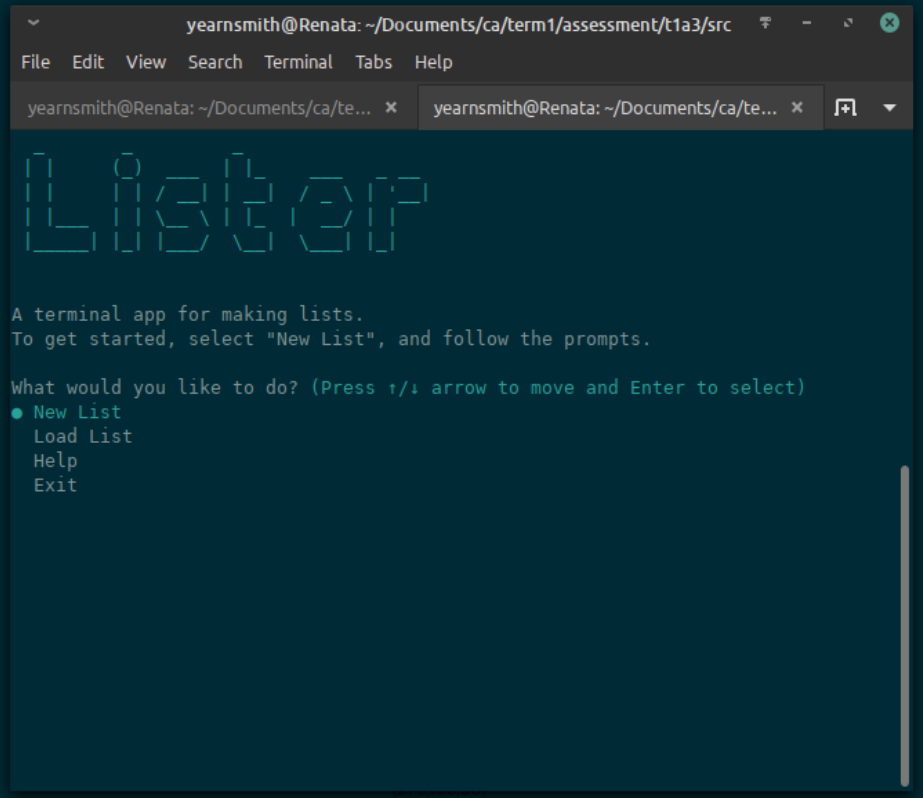
$ lister

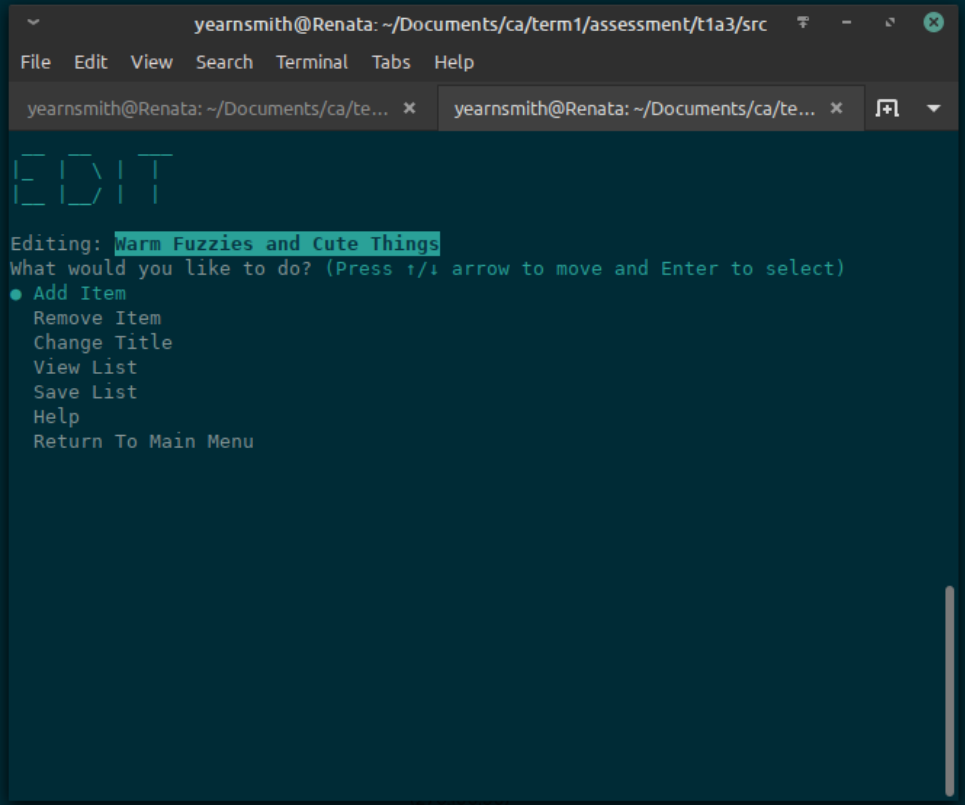
- files   = [lister.rb, List.rb, State.rb]
- classes = [List, State]
- require = [psych, pastel, tty-prompt,
               tty-font, bundler, rspec]

$ lister

- Two Modes:
  - Interactive
  - Linemode
- Optional Third Mode:
  - It speaks to you like Dave Lister

$ lister


yearnsmith@Renata: ~/Documents/ca/term1/assessment/t1a3/src
File Edit View Search Terminal Tabs Help
yearnsmith@Renata: ~/Documents/ca/te...
yearnsmith@Renata: ~/Documents/ca/te...
A terminal app for making lists.
To get started, select "New List", and follow the prompts.
What would you like to do? (Press ↑/↓ arrow to move and Enter to select)
● New List
Load List
Help
Exit


yearnsmith@Renata: ~/Documents/ca/term1/assessment/t1a3/src
File Edit View Search Terminal Tabs Help
yearnsmith@Renata: ~/Documents/ca/te...
yearnsmith@Renata: ~/Documents/ca/te...
Editing: Warm Fuzzies and Cute Things
What would you like to do? (Press ↑/↓ arrow to move and Enter to select)
● Add Item
Remove Item
Change Title
View List
Save List
Help
Return To Main Menu

```
30
31      system "clear"
32
33      case opt
34
35      when choices[0]
36        editing_titles(opt)
37        item_to_add = State.ask "What item would you like to add?"
38
39        puts list.add_item(item_to_add)
40      when choices[1]
41        editing_titles(opt)
42        if list.list_items.length == 0
43          puts "There are no items in this list."
44        else
45
46          puts list.list_title
47          puts "="*list.list_title.length
48          puts list.list_items
49
50
51          item_to_remove = State.ask "What item would you like to remove?"
52
53          puts list.remove_item(item_to_remove)
54        end
55      when choices[2]
56        editing_titles(opt)
57        puts "Current Title:\n\"#{list.list_title}\""
58        new_title = State.ask "What would you like to change the title to?"
59
60        new_title = State.check_if_nil(new_title)
61
62        puts list.change_title(new_title)
63      when choices[3]

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Close        ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Linter    ^_ Go To Line
```

```ruby
# List Class

# Lists:
# [!] .create_list(State method) != .init_list(State method) != initializing/instantiatir
# [X] Are objects of List class.
# [X] Have a title
# [X] Contain items
# [X] Store items in an array
# [X] It's items may be added
# [X] It's items may be removed
# [X] It's title may be changed
# [X] Store title and items together in a hash & YAML format
# [X] Are saved to disc in YAML format

require 'psych'
require 'tty-prompt'
require 'tty-font'
require 'pastel'
require_relative 'state'

class List

  attr_reader :list_hash, :list_yaml #, :removed_items
  attr_accessor :list_title, :list_items, :removed_items, :added_items

  def initialize(title)
    #Neccessary for displaying, storing, and accessing list data
    @list_title = title
    @list_items = []
    @list_hash = { @list_title => {items: @list_items, last_five_removed: @removed_items,
    @list_yaml = Psych.dump @list_hash

    #Features for future
    #store last 3 removed items for this list
    @removed_items = []
```

```ruby
      array.shift( (array.length + 1) - n_items )
    end
  end


  def add_item(item_to_add)

    @list_items << item_to_add
    return "\"#{item_to_add}\" has been added to \"#{@list_title}\""
  end

  def remove_item(item_to_remove)
    # Ensure there are items to remove
    if @list_items.length == 0
      puts "There are no items in this list."
      return
    end
    # Ensure sure item is in list
    until @list_items.include?(item_to_remove)
      puts "\"#{item_to_remove}\" is not in the list.\n"
      item_to_remove = State.ask "What item would you like to remove?"
    end
    # Confirm Deletion
    puts "This will delete the last occurance of \"#{item_to_remove}\".\nIt will be gone forever..."
    confirm = State.menu.select("Are you sure you want to delete?",%w(Yes No))

    #remove item and add it to an array of removed items, along with it's index in the list.
    #This will be handy for any debugging, log files, and a future "undo" action.
    if confirm == "Yes"
      # ensure @removed_items never exceeds 5 items.
      limit_items(@removed_items, 5)
      #push the removed item and it's index to @removed_items
      # 1. access @list_items, get the index of the last instance item_to_remove.
      # 2. access @list_items, get the index of the last instance of item_to_remove. Then delete the it
```

```ruby
# [x] Loads List files from disc
# [ ] Stores recent actions
# [ ] Enables Undo actions
# [ ] Enables Redo actions
# [x] Stores metadata in a hash & YAML format
# [x] Metadata is saved to disc in YAML format

class State

  attr_reader :state_dir, :state_name, :state_file, :state_hash, :state_yaml,:titles
  attr_accessor :files, :linemode


  #Where these files are stored
  @@main_options = ["New List", "Load List", "Help", "Exit"]
  @@edit_options = [ "Add Item", "Remove Item", "Change Title", "View List", "Save List", "Help", "Ret
  # Naming
  @@invalid_title_chars = "\"\'\\\/\:\:\*\<\>\|\&"
  @@state_dir = "./states/"
  @@list_dir = "./lists/"

  #State File:
  @@state_files = Dir.children(@@state_dir)

  # Use this for all prompts
  @@menu = TTY::Prompt.new(symbols: {marker: "●"}, help_color: :cyan, active_color: :cyan)
  # use this for all coloured text
  @@pastel = Pastel.new


  def initialize(name="default")

    @state_name = name
    @state_file = "#{@state_name}.lstr"

```

```ruby
  #Where these files are stored
  @@main_options = ["New List", "Load List", "Help", "Exit"]
  @@edit_options = [ "Add Item", "Remove Item", "Change Title", "View List", "Save List", "Help", "Ret
  # Naming
  @@invalid_title_chars = "\"\'\\\/\:\:\*\<\>\|\&"
  @@state_dir = "./states/"
  @@list_dir = "./lists/"

  #State File:
  @@state_files = Dir.children(@@state_dir)

  # Use this for all prompts
  @@menu = TTY::Prompt.new(symbols: {marker: "●"}, help_color: :cyan, active_color: :cyan)
  # use this for all coloured text
  @@pastel = Pastel.new


  def initialize(name="default")

    @state_name = name
    @state_file = "#{@state_name}.lstr"

    @files = Dir.children( @@list_dir ).select { |f| f.end_with? "yml"}
    @@titles = @files.map(&:clone).each do |i|
      i.delete_suffix!(".yml")
      i.gsub!("-"," ")
    end
    @linemode = false

    @state_hash = { @state_name.to_sym => { linemode: @linemode, files: @files } }
    @state_yaml = Psych.dump(@state_hash)
  end

  #### Getters ####

  def self.dir
```

```ruby
    #instantiate blank list with file name
    list = List.new(file_to_load)

    #print hash
      # open file with IO class -- this automatically closes the file for me. :)
      # Use Psych to convert yaml file into a hash. Using safe_load to de-serialize for safety.
      # https://ruby-doc.org/core-2.7.2/IO.html#method-c-read
      # https://ruby-doc.org/stdlib-2.7.2/libdoc/psych/rdoc/Psych.html#method-c-safe_load
      file_to_load = Psych.safe_load( IO.read( "#{@@list_dir}#{file_to_load}.yml" ),permitted_classes
      # Map hash to instance variables
      list.list_title = a = file_to_load.to_a[0][0]
      list.list_items = file_to_load[a][:items]
      list.removed_items = file_to_load[a][:last_five_removed]
      list.added_items = file_to_load[a][:last_five_added]
      # Update the hash and yaml for the List instance (These aren't publically writable)
      list.update_yaml
      # Put confirmation
      puts "Editing #{list.list_title}..."
      State.press_any_key
      # return list object to lister
      return list
  end
  #Save Lists
  def self.save_list(title,yaml)
    
    puts "Saving to disc..."
    IO.write("#{@@list_dir}#{title}.yml",yaml)

    if !@@titles.include?(title)
      @@titles << title
    end

    return "\"#{title}\" has been saved. :)"
  end

  def update_list_titles
    @files = Dir.children( @@list_dir ).select {|f| f.end_with? "yml"}
```

```ruby
      list.added_items = file_to_load[a][:last_five_added]
      # Update the hash and yaml for the List instance (These aren't publically writable)
      list.update_yaml
      # Put confirmation
      puts "Editing #{list.list_title}..."
      State.press_any_key
      # return list object to lister
      return list
  end
  #Save Lists
  def self.save_list(title,yaml)

    puts "Saving to disc..."
    IO.write("#{@@list_dir}#{title}.yml",yaml)

    if !@@titles.include?(title)
      @@titles << title
    end

    return "\"#{title}\" has been saved. :)"
  end

  def update_list_titles
    @files = Dir.children( @@list_dir ).select { |f| f.end_with? "yml"}

    @@titles = @files.map(&:clone).each do |i|
                i.delete_suffix!(".yml")
               end
  end

end# Class
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo M
^X Close ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo M

$ lister

```
$ lister “App Idea” “It makes lists”
```

- 1 arguments
  1. open list
- 2 arguments
  1. open list
  2. add item

$ lister

- [x] Let’s make a list

```
#!/usr/bin/bash

Cp files usr/bin/lister

# modify to if/elsif/else statement to find user's .bashrc
Echo "alias lister='usr/bin/lister/ruby lister.rb'" >> ~.bashrc
#  will run as lister (but will it accept variables?)
```

$ lister --ho

or

```
#!/usr/bin/ruby

# ruby code goes here


#to run, type: ./ruby.rb
```

$ lister

```
$ lister --add “App Idea” “It makes lists”
```

```
$ lister --remove “App Idea” “It makes lists”
```

```
$ lister --echo “App Idea”
```

[[-a , --add], [-r, --remove], [-e, --echo]]

$ lister

- Validation
  - Loops!
    - Multiple validation arguments don’t work
      - Solving: Use TTY-prompt’s built in validation
- Scope
  - I underestimated the length of some tasks

$ lister

- Validating file name… I learned some RegExp
- Is my code modular?
  - List class is accessing State class
  - State class is accessing List class
- Should I be modifying someone’s .bashrc?
  - Should I just rename “lister.rb" to “lister”?

$ lister

- Finish Triage
- Convert loops to error handling
- Add colours in validation
- Fancify list titles etc.