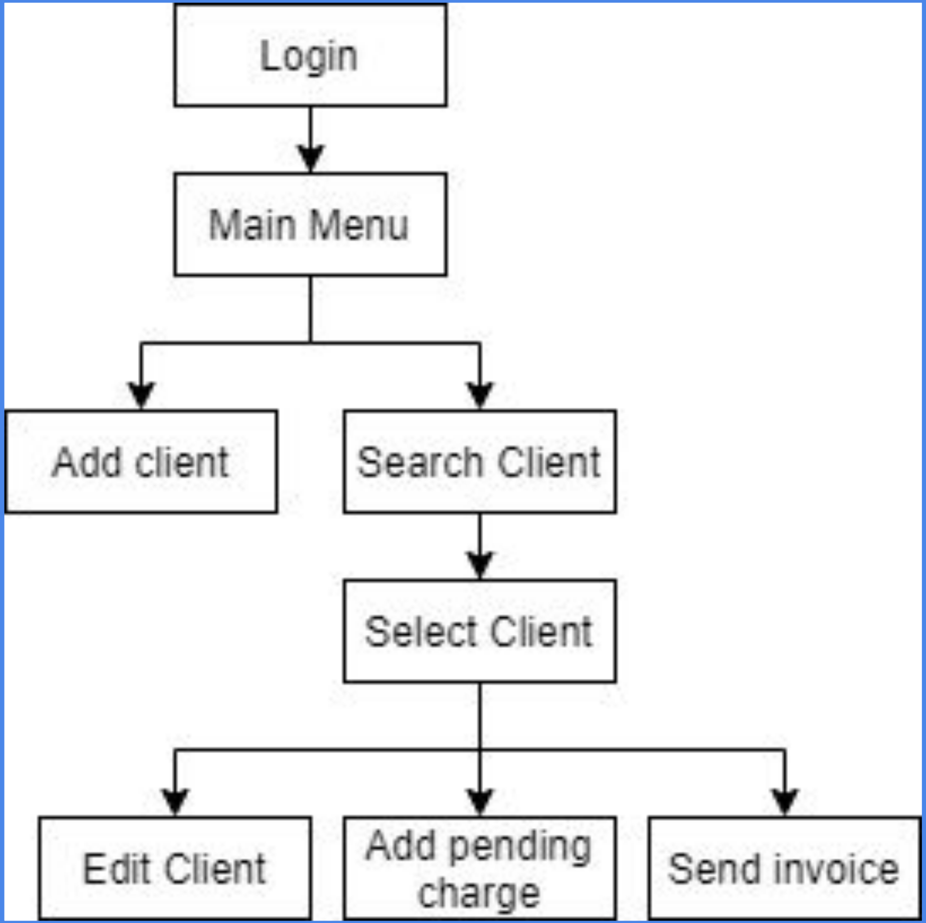
T1A3 Ruby Terminal App

BashBooks - Invoicing app for the terminal worker

Features

- Separate login accounts for write and read access
- Create, Read, Update, and Delete client information saved to a file
- Export client list to a JSON file for reading in other apps
- Create charges to an account for work done
- Send invoice emails to clients with charges on their account

Menu tree


Login
Main Menu
Add client
Search Client
Select Client
Edit Client
Add pending charge
Send invoice

Code overview

Code is structured into methods for each stage of the menu:

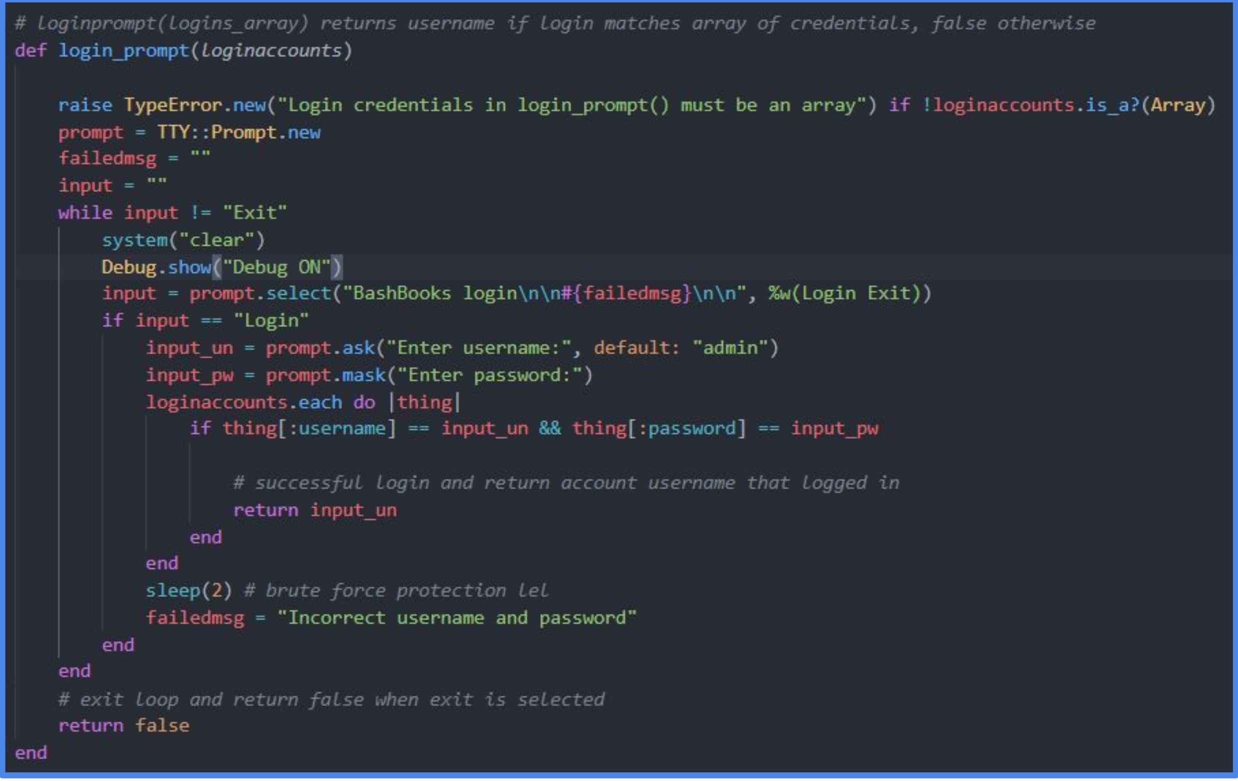
login_prompt()

main_menu()

add_new_client()

clientsearch()

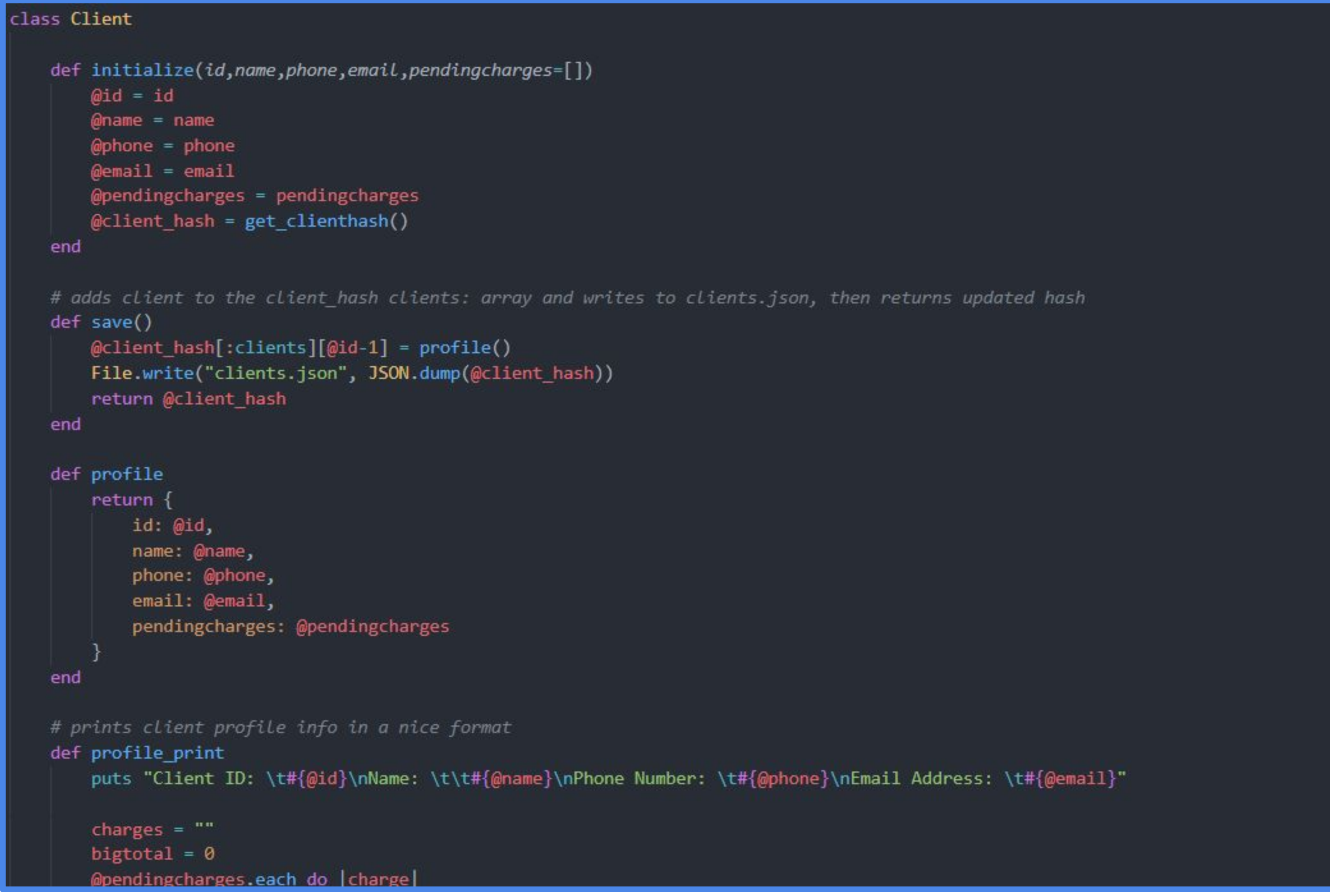
selectclient()



```ruby
# loginprompt(logins_array) returns username if login matches array of credentials, false otherwise
def login_prompt(loginaccounts)

    raise TypeError.new("Login credentials in login_prompt() must be an array") if !loginaccounts.is_a?(Array)
    prompt = TTY::Prompt.new
    failedmsg = ""
    input = ""
    while input != "Exit"
        system("clear")
        Debug.show("Debug ON")
        input = prompt.select("BashBooks login\n\n#{failedmsg}\n\n", %w(Login Exit))
        if input == "Login"
            input_un = prompt.ask("Enter username:", default: "admin")
            input_pw = prompt.mask("Enter password:")
            loginaccounts.each do |thing|
                if thing[:username] == input_un && thing[:password] == input_pw

                    # successful login and return account username that logged in
                    return input_un
                end
            end
            sleep(2) # brute force protection lel
            failedmsg = "Incorrect username and password"
        end
    end
    # exit loop and return false when exit is selected
    return false
end
```

Code overview

Whenever editing or adding a new client, the class `Client` contains all methods for storing a client's information, writing their information to the JSON file, and sending invoices using the mailjet ruby gem

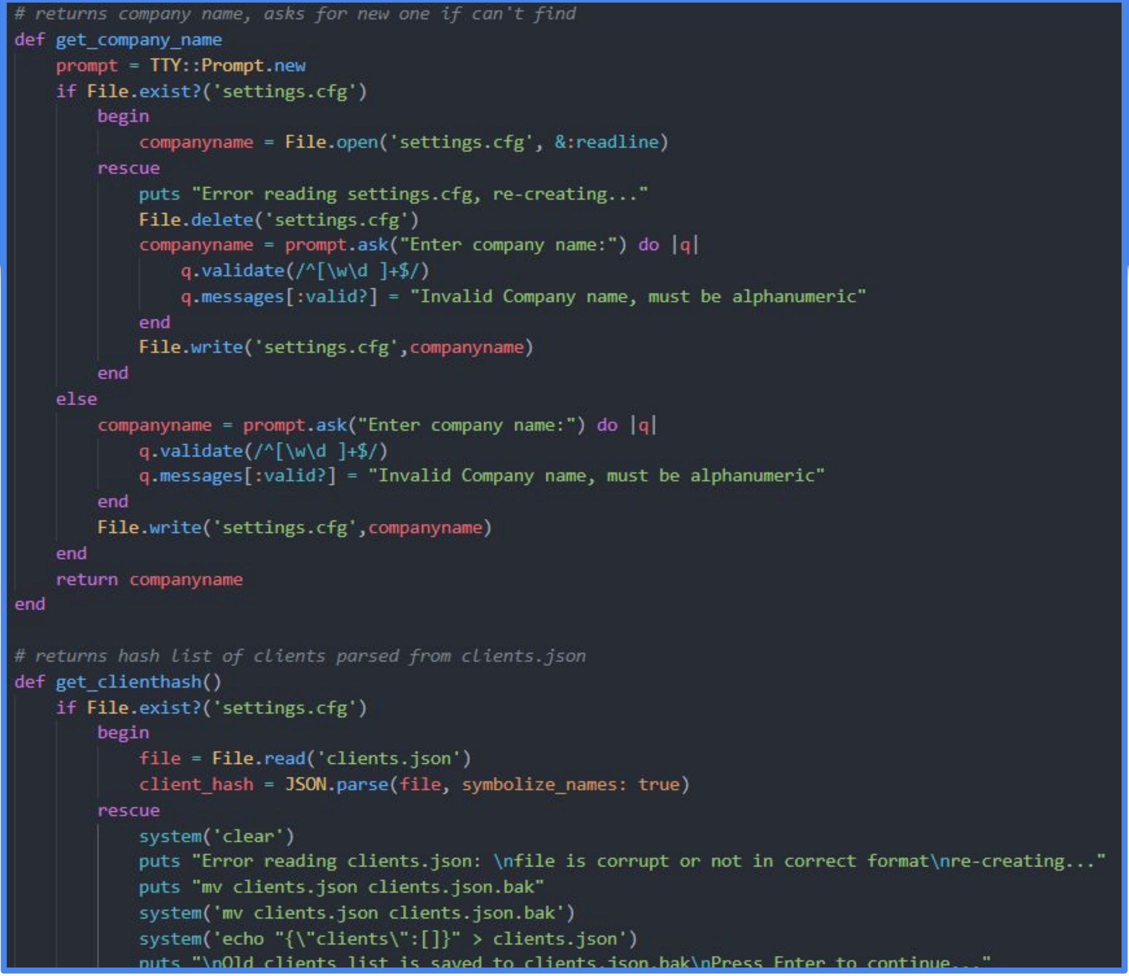


```ruby
class Client

    def initialize(id,name,phone,email,pendingcharges=[])
        @id = id
        @name = name
        @phone = phone
        @email = email
        @pendingcharges = pendingcharges
        @client_hash = get_clienthash()
    end

    # adds client to the client_hash clients: array and writes to clients.json, then returns updated hash
    def save()
        @client_hash[:clients][@id-1] = profile()
        File.write("clients.json", JSON.dump(@client_hash))
        return @client_hash
    end

    def profile
        return {
            id: @id,
            name: @name,
            phone: @phone,
            email: @email,
            pendingcharges: @pendingcharges
        }
    end

    # prints client profile info in a nice format
    def profile_print
        puts "Client ID: \t#{@id}\nName: \t\t#{@name}\nPhone Number: \t#{@phone}\nEmail Address: \t#{@email}"

        charges = ""
        bigtotal = 0
        @pendingcharges.each do |charge|
```

Code overview

File handling methods are stored in files.rb

This contains methods for reading the JSON file to return a hash, and opening the settings file to get the company name

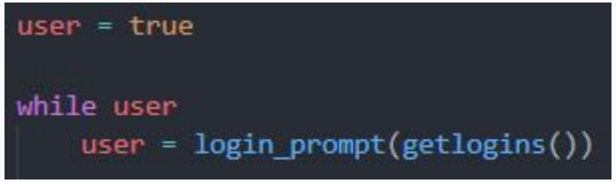


```ruby
# returns company name, asks for new one if can't find
def get_company_name
    prompt = TTY::Prompt.new
    if File.exist?('settings.cfg')
        begin
            companyname = File.open('settings.cfg', &:readline)
        rescue
            puts "Error reading settings.cfg, re-creating..."
            File.delete('settings.cfg')
            companyname = prompt.ask("Enter company name:") do |q|
                q.validate(/^[\w\d ]+$/)
                q.messages[:valid?] = "Invalid Company name, must be alphanumeric"
            end
            File.write('settings.cfg',companyname)
        end
    else
        companyname = prompt.ask("Enter company name:") do |q|
            q.validate(/^[\w\d ]+$/)
            q.messages[:valid?] = "Invalid Company name, must be alphanumeric"
        end
        File.write('settings.cfg',companyname)
    end
    return companyname
end

# returns hash list of clients parsed from clients.json
def get_clienthash()
    if File.exist?('settings.cfg')
        begin
            file = File.read('clients.json')
            client_hash = JSON.parse(file, symbolize_names: true)
        rescue
            system('clear')
            puts "Error reading clients.json: \nfile is corrupt or not in correct format\nre-creating..."
            puts "mv clients.json clients.json.bak"
            system('mv clients.json clients.json.bak')
            system('echo "{\"clients\":[]}" > clients.json')
            puts "\nOld clients list is saved to clients.json.bak\nPress Enter to continue..."
```

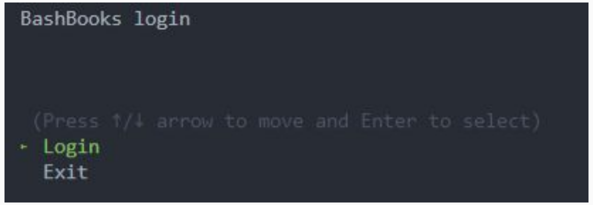
Feature 1 - Login accounts

The initial main.rb script will run the login_prompt() method, using a hash of logins from the files.rb method getlogins().

Until the user selects "Exit" in the prompt, they will continue to get the login prompt.



```
user = true

while user
    user = login_prompt(getlogins())
```

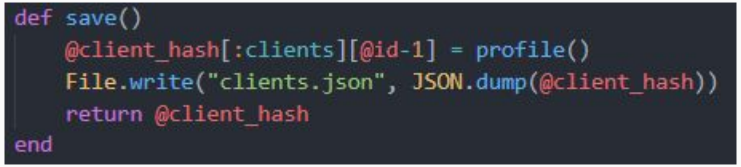


```
BashBooks login


 (Press ↑/↓ arrow to move and Enter to select)
‣ Login
  Exit
```

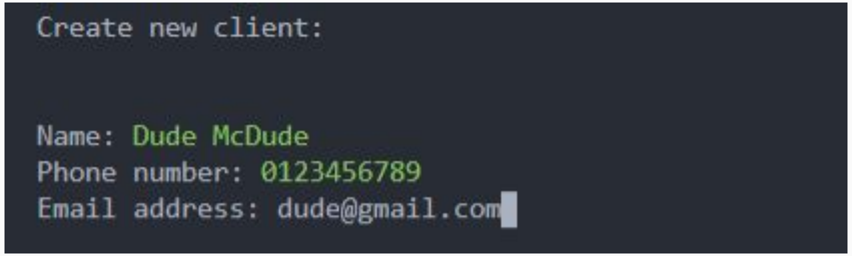
In the main menu, the logged in user will be stored in `username` to determine if the user has write or read privileges. The first release will just have 2 accounts, admin/admin and guest/guest for write and read respectively.

Feature 2 - CRUD client information

Within the main_menu() method, there will be the option to “add new client” - this will create a new `Client` object with instance variables filled in by the tty-prompts for user input.



```ruby
def save()
    @client_hash[:clients][@id-1] = profile()
    File.write("clients.json", JSON.dump(@client_hash))
    return @client_hash
end
```

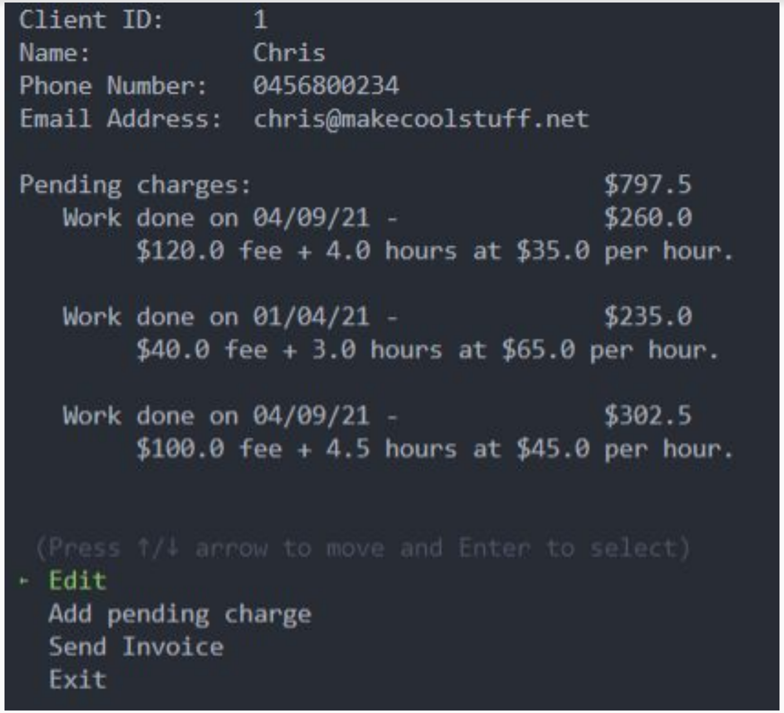


```
Create new client:


Name: Dude McDude
Phone number: 0123456789
Email address: dude@gmail.com
```

This will then call a Client.save() method which will save their information to the JSON file

Feature 3 - Add charges to account

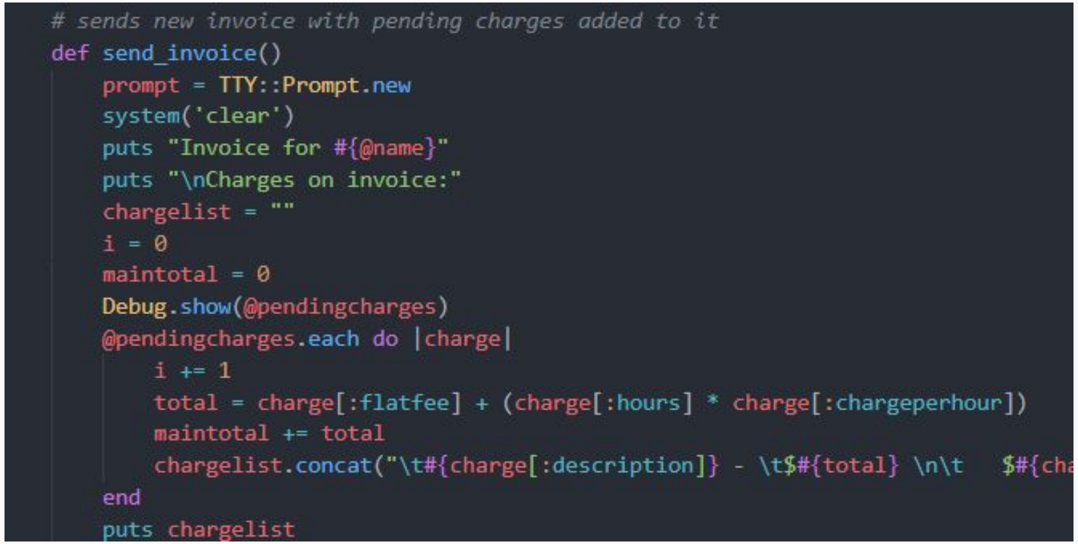
The `Client` object will have a method add_charge() which will add a pending charge hash to the client's account with a description of the charge, flat fees, billable hours, and charge per hour.



```
Client ID:       1
Name:            Chris
Phone Number:    0456800234
Email Address:   chris@makecoolstuff.net

Pending charges:                         $797.5
   Work done on 04/09/21 -               $260.0
        $120.0 fee + 4.0 hours at $35.0 per hour.

   Work done on 01/04/21 -               $235.0
        $40.0 fee + 3.0 hours at $65.0 per hour.

   Work done on 04/09/21 -               $302.5
        $100.0 fee + 4.5 hours at $45.0 per hour.


 (Press ↑/↓ arrow to move and Enter to select)
‣ Edit
  Add pending charge
  Send Invoice
  Exit
```

Feature 4 - Send invoice via email

The `Client` object will have a send_invoice method that will put all pending charges onto an email template and send it through the mailjet ruby gem



```
# sends new invoice with pending charges added to it
def send_invoice()
    prompt = TTY::Prompt.new
    system('clear')
    puts "Invoice for #{@name}"
    puts "\nCharges on invoice:"
    chargelist = ""
    i = 0
    maintotal = 0
    Debug.show(@pendingcharges)
    @pendingcharges.each do |charge|
        i += 1
        total = charge[:flatfee] + (charge[:hours] * charge[:chargeperhour])
        maintotal += total
        chargelist.concat("\t#{charge[:description]} - \t$#{total} \n\t   $#{cha
    end
    puts chargelist
```