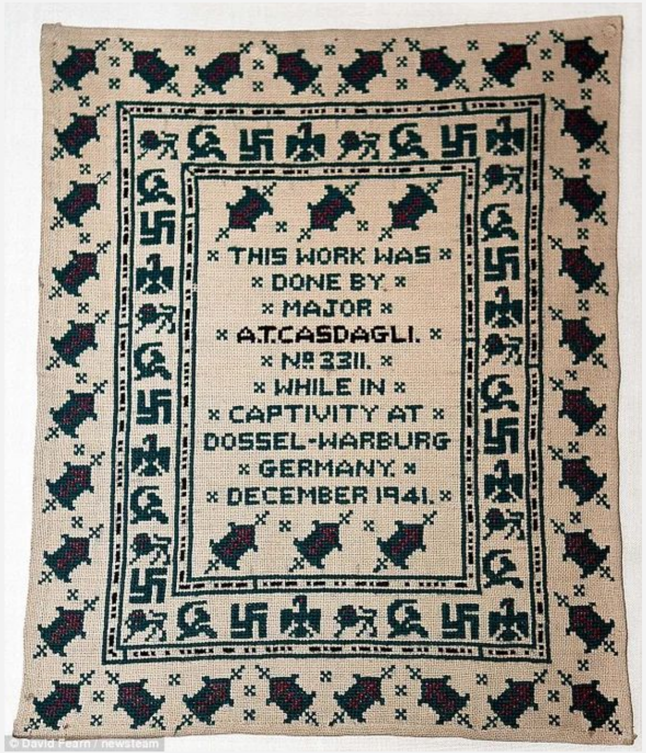
SteggyHide

Jeremy Bell

What is Steganography

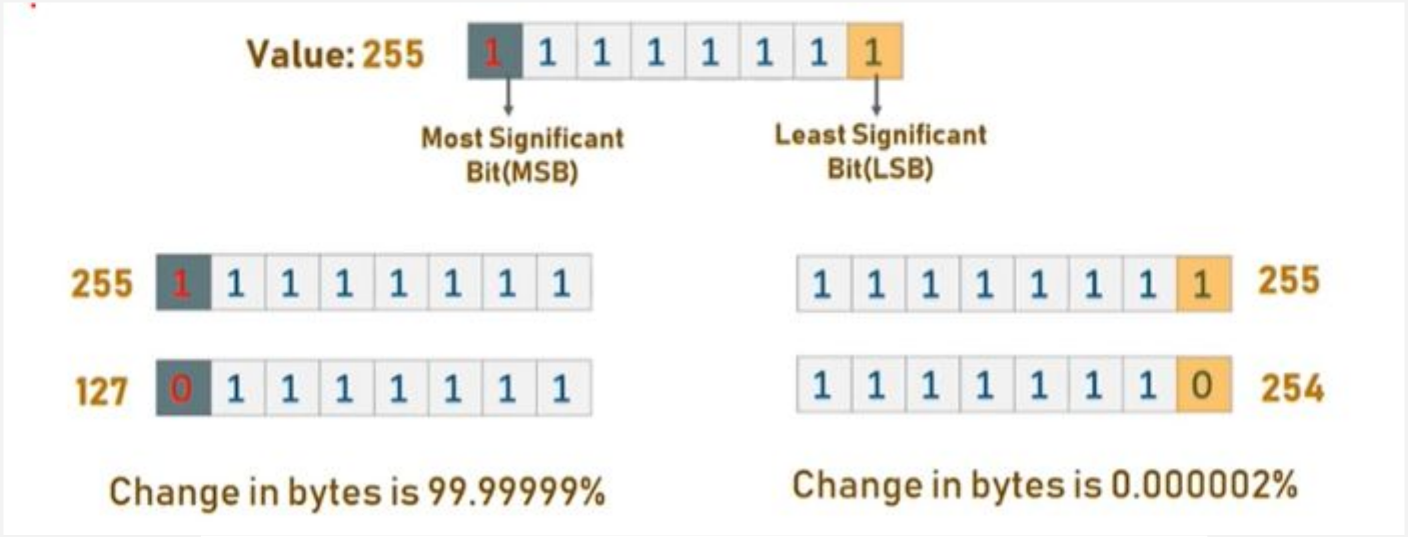
- Hiding things in a container in plain sight
- Physical examples include:
  - Invisible ink
  - Sewing morse code in clothes/tapestry


THIS WORK WAS DONE BY MAJOR A.T.CASDAGLI. Nº 3311. WHILE IN CAPTIVITY AT DOSSEL-WARBURG GERMANY. DECEMBER 1941.
© David Fearn / newsteam

In WWII British POW Major Alexis Casdagli made this tapestry. The morse code messages decode to “God Save the King” and “F*ck Hitler”
Source: https://www.wired.com/2012/01/british-pow-uses-morse-code-to-stitch-hidden-message-during-wwii/

Steganography in Digital Images

- Data is stored in images in the form of colour information
- Most common form is done by manipulating the Least Significant Bit (LSB)
- RGB colours are represented by 1 byte (8 bits) for each colour channel
- If we change the LSB we can encode data but the human eye can't tell the difference



Value: 255
1 1 1 1 1 1 1 1
Most Significant Bit(MSB)
Least Significant Bit(LSB)
255 1 1 1 1 1 1 1 1
127 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 255
1 1 1 1 1 1 1 0 254
Change in bytes is 99.99999%
Change in bytes is 0.000002%

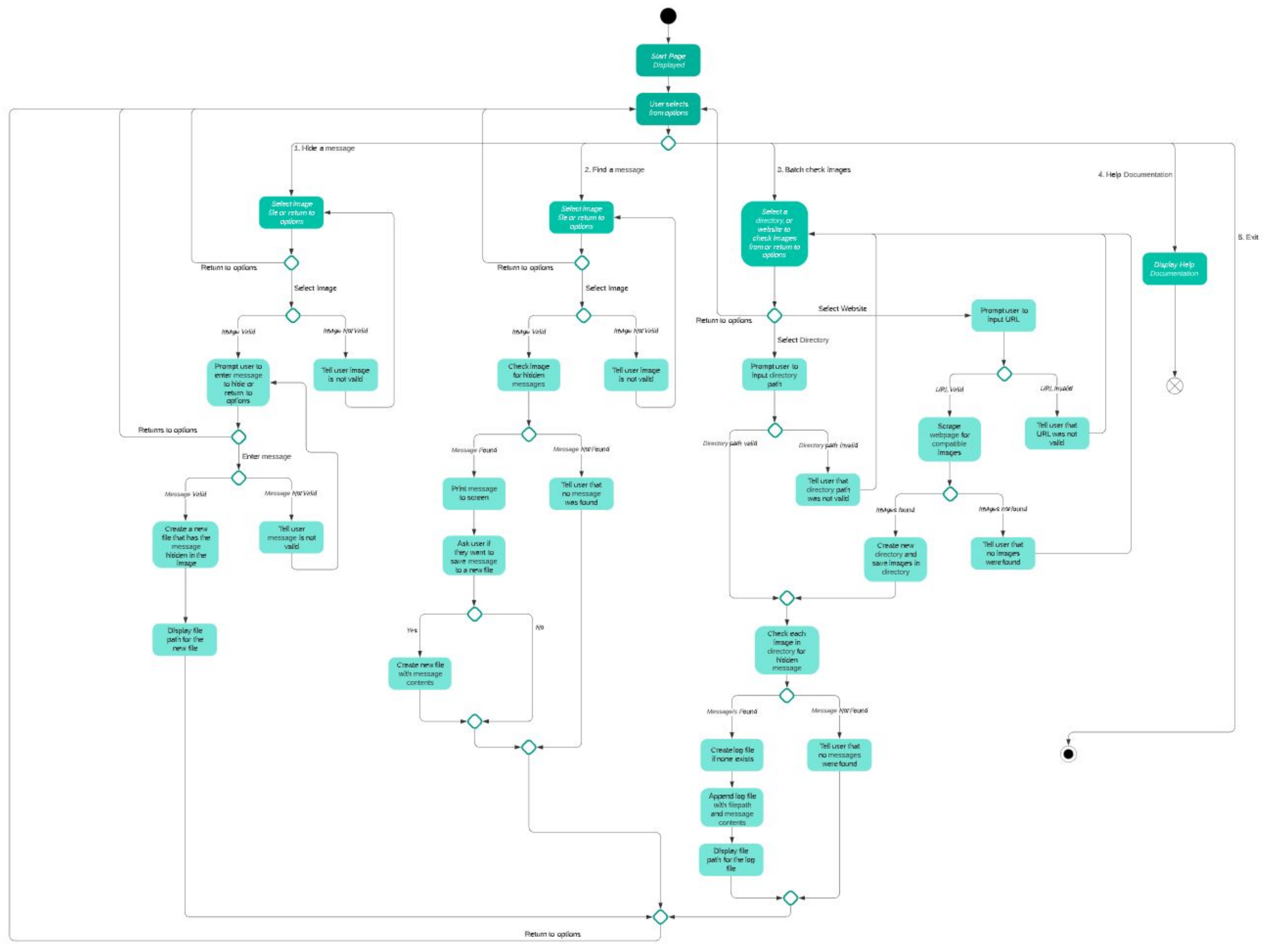
Photo by Edureka Steganography tutorial - https://www.edureka.co/blog/steganography-tutorial

Features

1. User can encode a hidden message within a PNG image
2. User can decode a hidden message within a PNG image
3. User can batch check PNG images from a directory or a webpage

SteggyHide Activity Diagram

Jeremy Bel | September 20, 2020

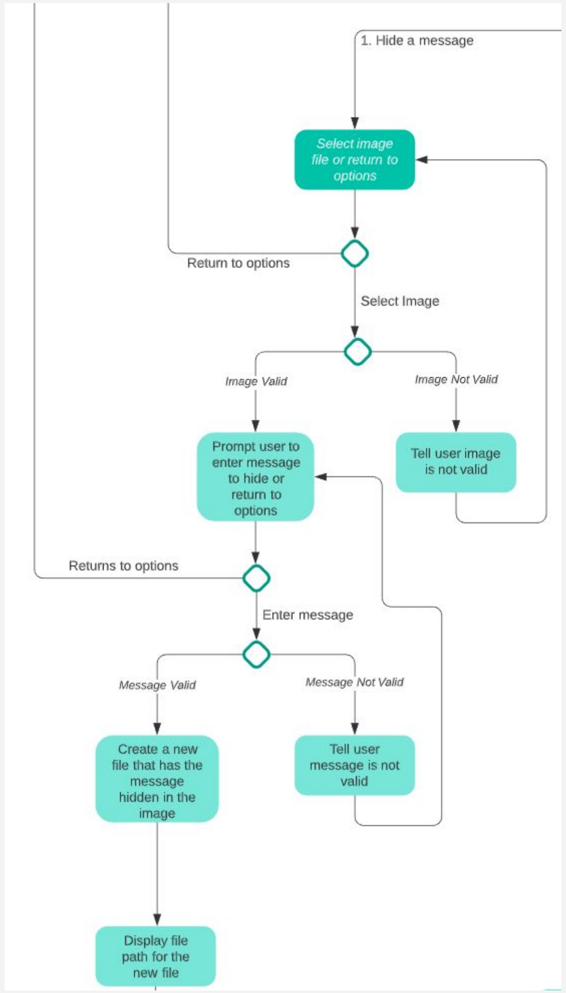

Start Page Displayed
User selects from options
1. Hide a message
2. Find a message
3. Batch check images
4. Help Documentation
5. Exit
Select image file or return to options
Return to options
Select Image
Image Valid
Image Not Valid
Prompt user to enter message to hide or return to options
Tell user image is not valid
Returns to options
Enter message
Message Valid
Message Not Valid
Create a new file that has the message hidden in the image
Tell user message is not valid
Display file path for the new file
Select image file or return to options
Return to options
Select Image
Image Valid
Image Not Valid
Check image for hidden messages
Tell user image is not valid
Message Found
Message Not Found
Print message to screen
Tell user that no message was found
Ask user if they want to save message to a new file
Yes
No
Create new file with message contents
Select a directory or website to check images from or return to options
Return to options
Select Website
Select Directory
Prompt user to input URL
Prompt user to input directory path
URL Valid
URL Invalid
Scrape webpage for compatible images
Tell user that URL was not valid
Directory path valid
Directory path invalid
Tell user that directory path was not valid
Images found
Images not found
Create new directory and save images in directory
Tell user that no images were found
Check each image in directory for hidden message
Message/s Found
Message Not Found
Create log file if none exists
Tell user that no messages were found
Append log file with filepath and message contents
Display file path for the log file
Display Help Documentation
Return to options

Feature 1 - Hide Message

- User inputs a file path to select a PNG file
- User inputs a message string to be hidden
- A new PNG file is created with the hidden message encoded

Error Handling

- File given is not a PNG
- Message invalid
    - Image is too small to hide that particular message

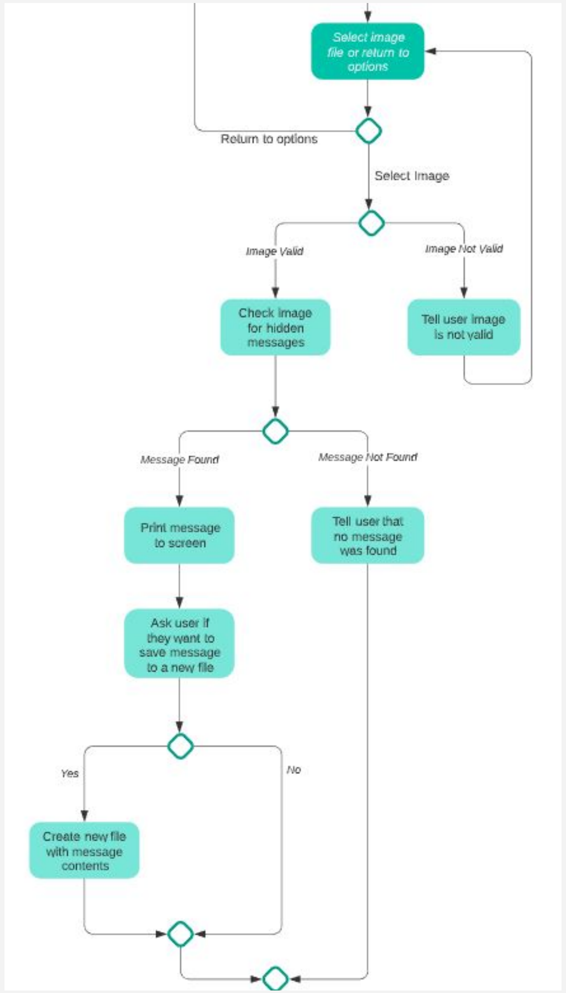

1. Hide a message
Select image file or return to options
Return to options
Select Image
Image Valid
Image Not Valid
Prompt user to enter message to hide or return to options
Tell user image is not valid
Returns to options
Enter message
Message Valid
Message Not Valid
Create a new file that has the message hidden in the image
Tell user message is not valid
Display file path for the new file

Feature 2 - Find Message

- User inputs a file path to select a PNG file
- SteggyHide checks to find a message
- Message is printed to the screen and user has option to save it in a text file

Error Handling

- File given is not a PNG
- Message not found

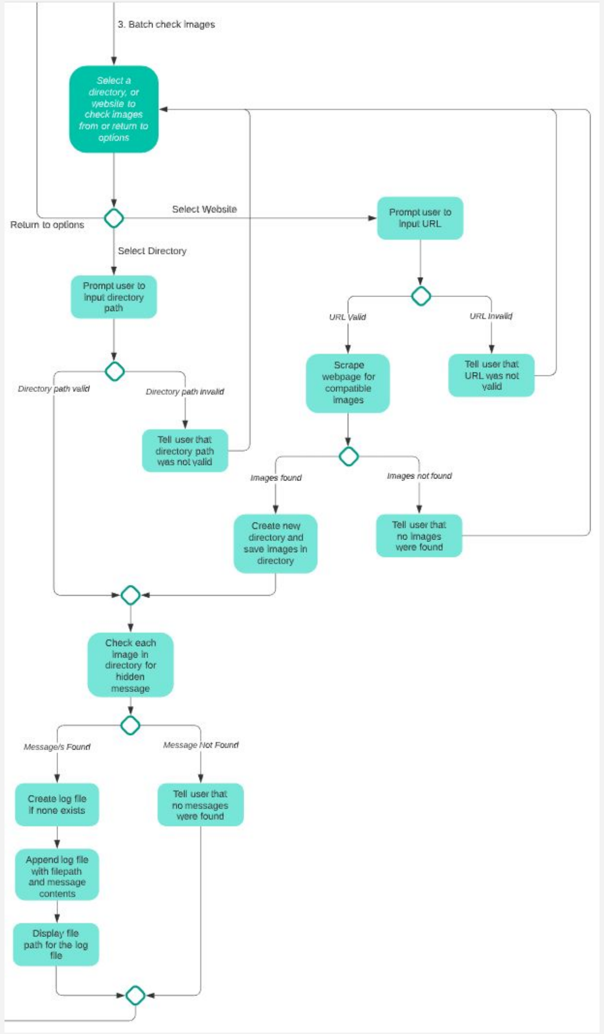

Select image file or return to options
Return to options
Select Image
Image Valid
Image Not Valid
Check image for hidden messages
Tell user image is not valid
Message Found
Message Not Found
Print message to screen
Tell user that no message was found
Ask user if they want to save message to a new file
Yes
No
Create new file with message contents

Feature 3 - Batch Find

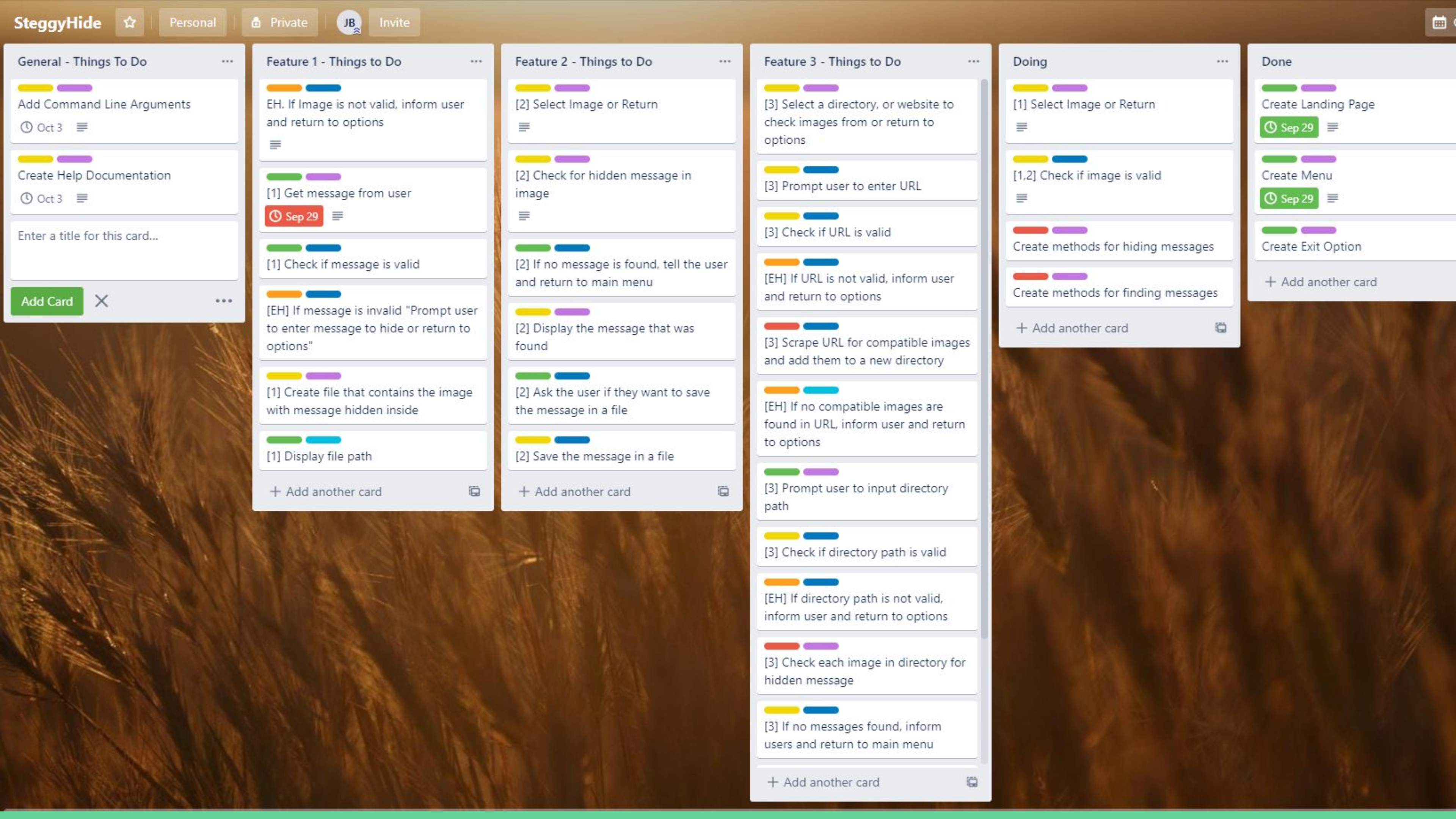
- User inputs a file directory or a webpage URL
- SteggyHide downloads all PNG images from webpage into a new directory
- SteggyHide checks all the images in the directory
- For each found message a log file is appended with the filepath and message contents

Error Handling

- Directory Invalid
- URL Invalid
- No Images in directory or website
- No Messages Found



3. Batch check images
Select a directory, or website to check images from or return to options
Return to options
Select Website
Prompt user to input URL
Select Directory
Prompt user to input directory path
URL Valid
URL Invalid
Scrape webpage for compatible images
Tell user that URL was not valid
Directory path valid
Directory path invalid
Tell user that directory path was not valid
Images found
Images not found
Create new directory and save images in directory
Tell user that no images were found
Check each image in directory for hidden message
Message/s Found
Message Not Found
Create log file if none exists
Tell user that no messages were found
Append log file with filepath and message contents
Display file path for the log file



SteggyHide

Personal · Private · JB · Invite

General - Things To Do

- Add Command Line Arguments — Oct 3
- Create Help Documentation — Oct 3
- Enter a title for this card…

Add Card

Feature 1 - Things to Do

- EH. If Image is not valid, inform user and return to options
- [1] Get message from user — Sep 29
- [1] Check if message is valid
- [EH] If message is invalid "Prompt user to enter message to hide or return to options"
- [1] Create file that contains the image with message hidden inside
- [1] Display file path

+ Add another card

Feature 2 - Things to Do

- [2] Select Image or Return
- [2] Check for hidden message in image
- [2] If no message is found, tell the user and return to main menu
- [2] Display the message that was found
- [2] Ask the user if they want to save the message in a file
- [2] Save the message in a file

+ Add another card

Feature 3 - Things to Do

- [3] Select a directory, or website to check images from or return to options
- [3] Prompt user to enter URL
- [3] Check if URL is valid
- [EH] If URL is not valid, inform user and return to options
- [3] Scrape URL for compatible images and add them to a new directory
- [EH] If no compatible images are found in URL, inform user and return to options
- [3] Prompt user to input directory path
- [3] Check if directory path is valid
- [EH] If directory path is not valid, inform user and return to options
- [3] Check each image in directory for hidden message
- [3] If no messages found, inform users and return to main menu

+ Add another card

Doing

- [1] Select Image or Return
- [1,2] Check if image is valid
- Create methods for hiding messages
- Create methods for finding messages

+ Add another card

Done

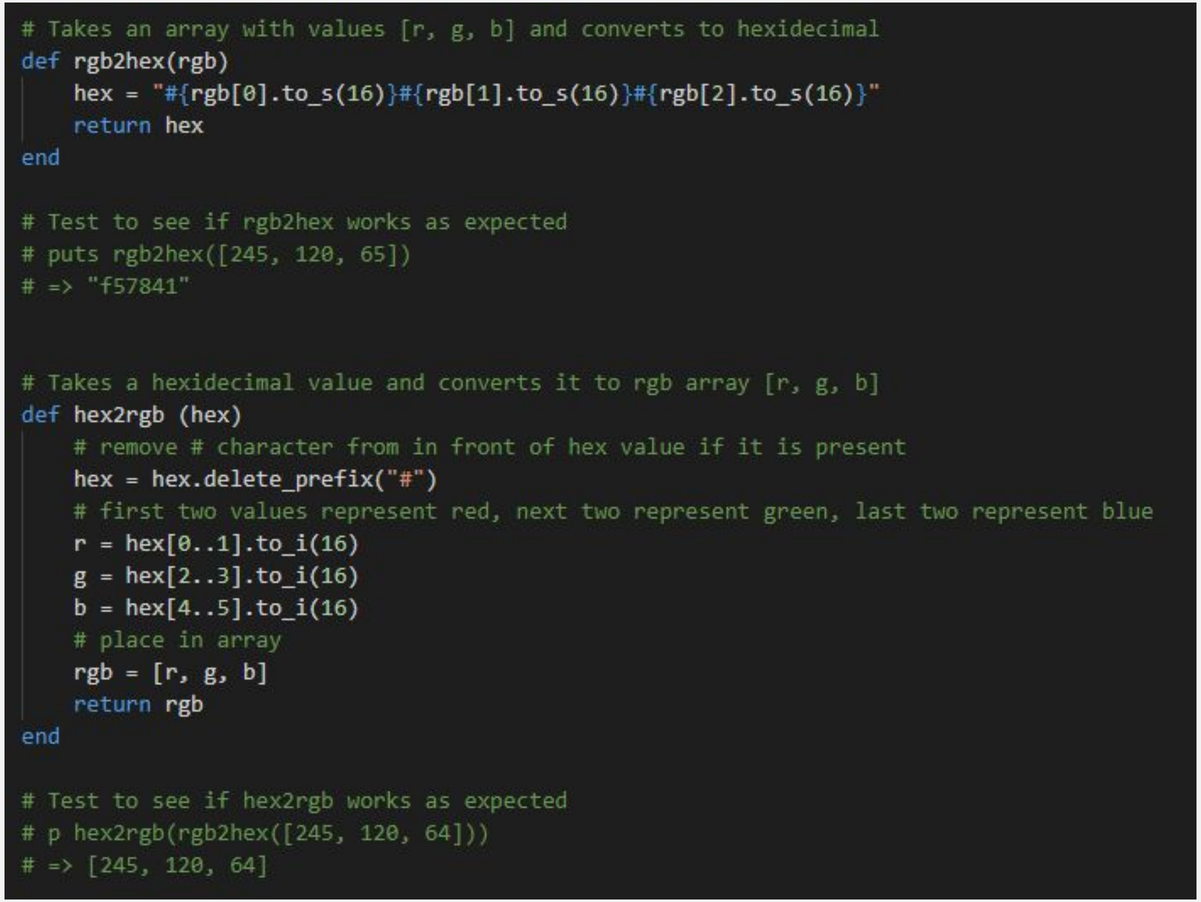
- Create Landing Page — Sep 29
- Create Menu — Sep 29
- Create Exit Option

+ Add another card

Challenges

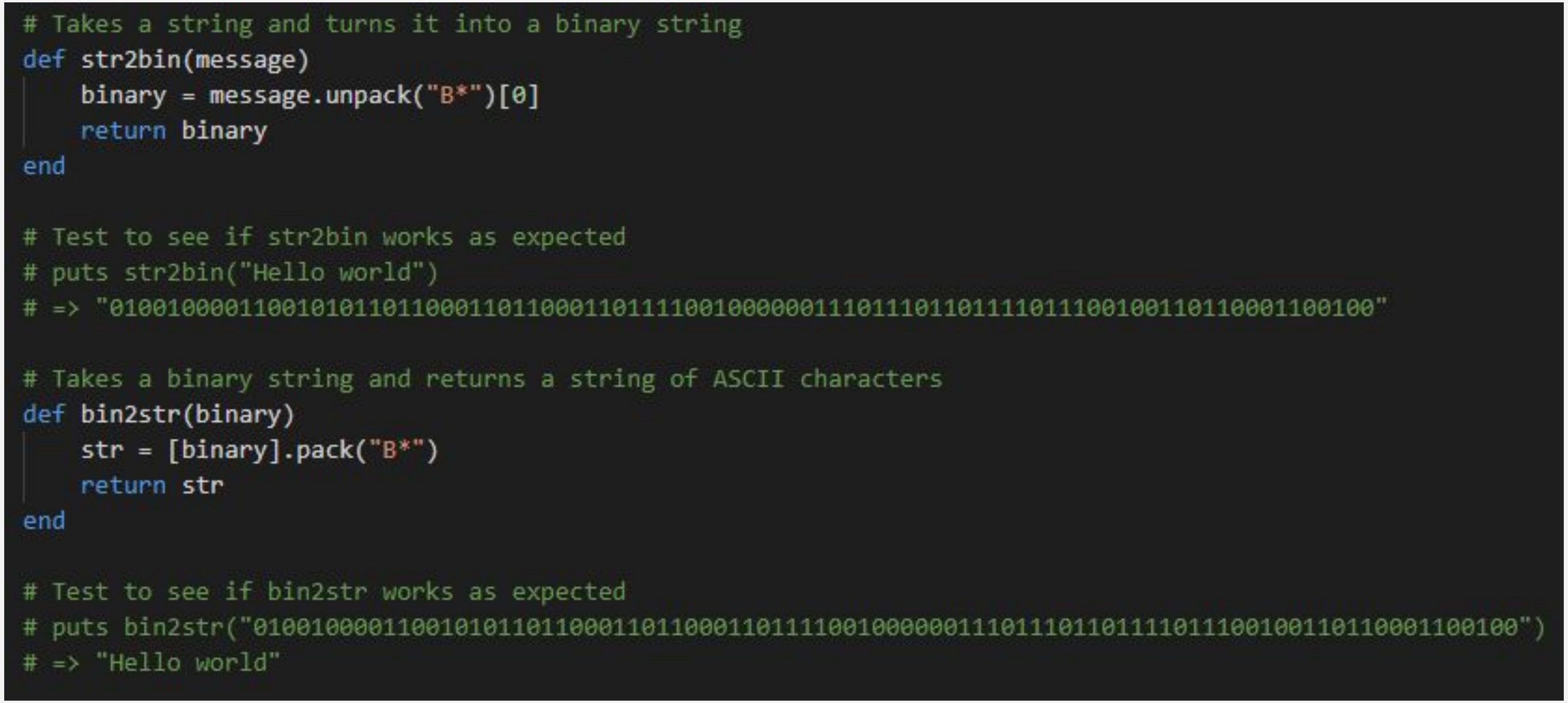
- The gem I planned on using didn’t do what I thought it did
  - Needed to create methods:
    - Hex to RGB, and RGB to Hex
    - String to Binary, and Binary to String
    - Encode, and Decode
    - Convert Image to Array of Pixel Data
    - Convert Array of Pixel Data to Image
    - Hide Message
  - Still need to create:
    - Find Message

Code



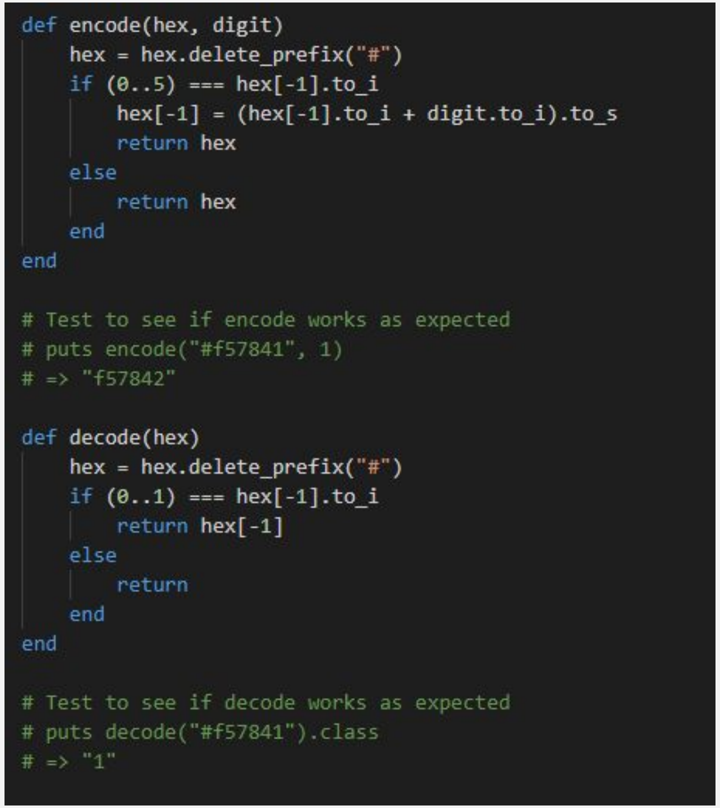
```ruby
# Takes an array with values [r, g, b] and converts to hexidecimal
def rgb2hex(rgb)
    hex = "#{rgb[0].to_s(16)}#{rgb[1].to_s(16)}#{rgb[2].to_s(16)}"
    return hex
end

# Test to see if rgb2hex works as expected
# puts rgb2hex([245, 120, 65])
# => "f57841"


# Takes a hexidecimal value and converts it to rgb array [r, g, b]
def hex2rgb (hex)
    # remove # character from in front of hex value if it is present
    hex = hex.delete_prefix("#")
    # first two values represent red, next two represent green, last two represent blue
    r = hex[0..1].to_i(16)
    g = hex[2..3].to_i(16)
    b = hex[4..5].to_i(16)
    # place in array
    rgb = [r, g, b]
    return rgb
end

# Test to see if hex2rgb works as expected
# p hex2rgb(rgb2hex([245, 120, 64]))
# => [245, 120, 64]
```

Code



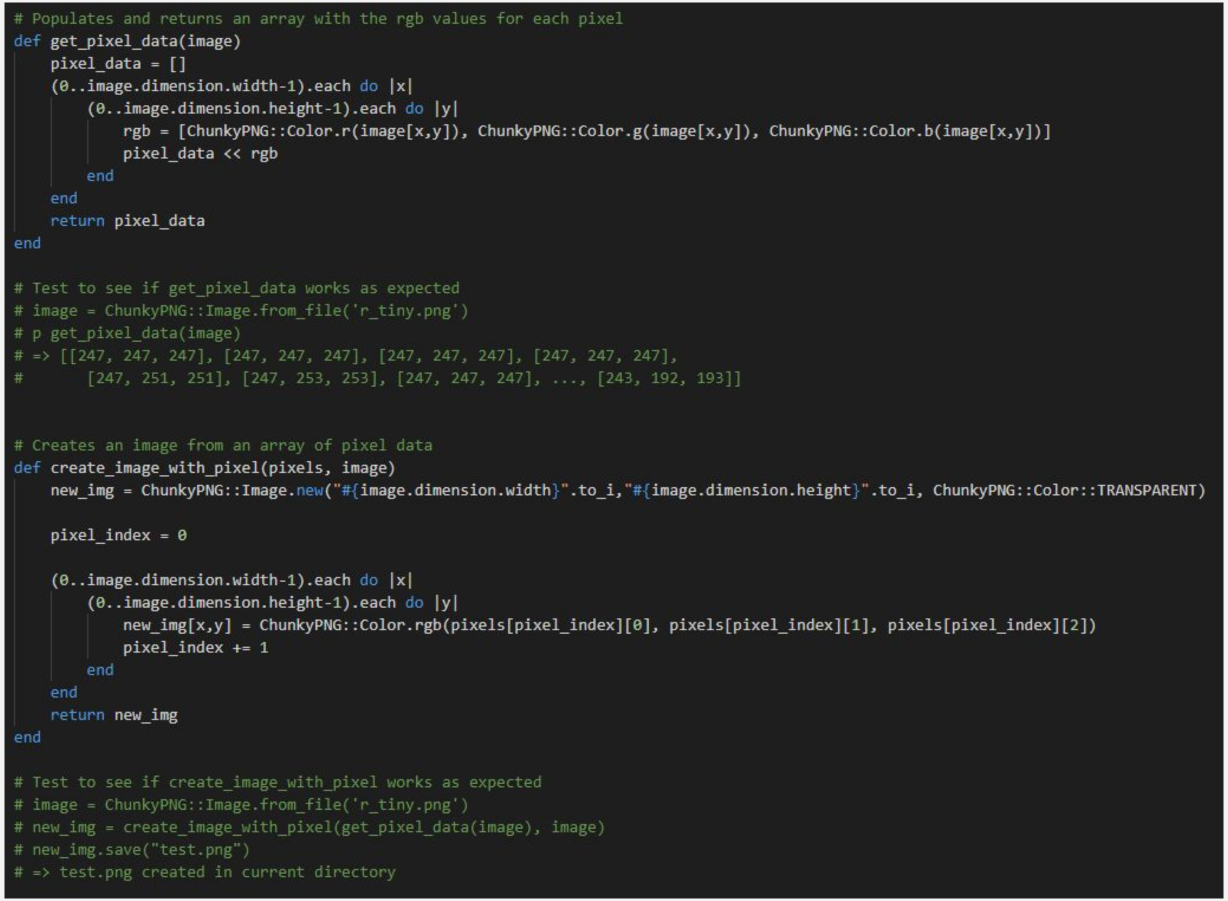
```ruby
# Takes a string and turns it into a binary string
def str2bin(message)
    binary = message.unpack("B*")[0]
    return binary
end

# Test to see if str2bin works as expected
# puts str2bin("Hello world")
# => "0100100001100101011011000110110001101111001000000111011101101111011100100110110001100100"

# Takes a binary string and returns a string of ASCII characters
def bin2str(binary)
    str = [binary].pack("B*")
    return str
end

# Test to see if bin2str works as expected
# puts bin2str("0100100001100101011011000110110001101111001000000111011101101111011100100110110001100100")
# => "Hello world"
```

Code



```ruby
def encode(hex, digit)
    hex = hex.delete_prefix("#")
    if (0..5) === hex[-1].to_i
        hex[-1] = (hex[-1].to_i + digit.to_i).to_s
        return hex
    else
        return hex
    end
end

# Test to see if encode works as expected
# puts encode("#f57841", 1)
# => "f57842"

def decode(hex)
    hex = hex.delete_prefix("#")
    if (0..1) === hex[-1].to_i
        return hex[-1]
    else
        return
    end
end

# Test to see if decode works as expected
# puts decode("#f57841").class
# => "1"
```

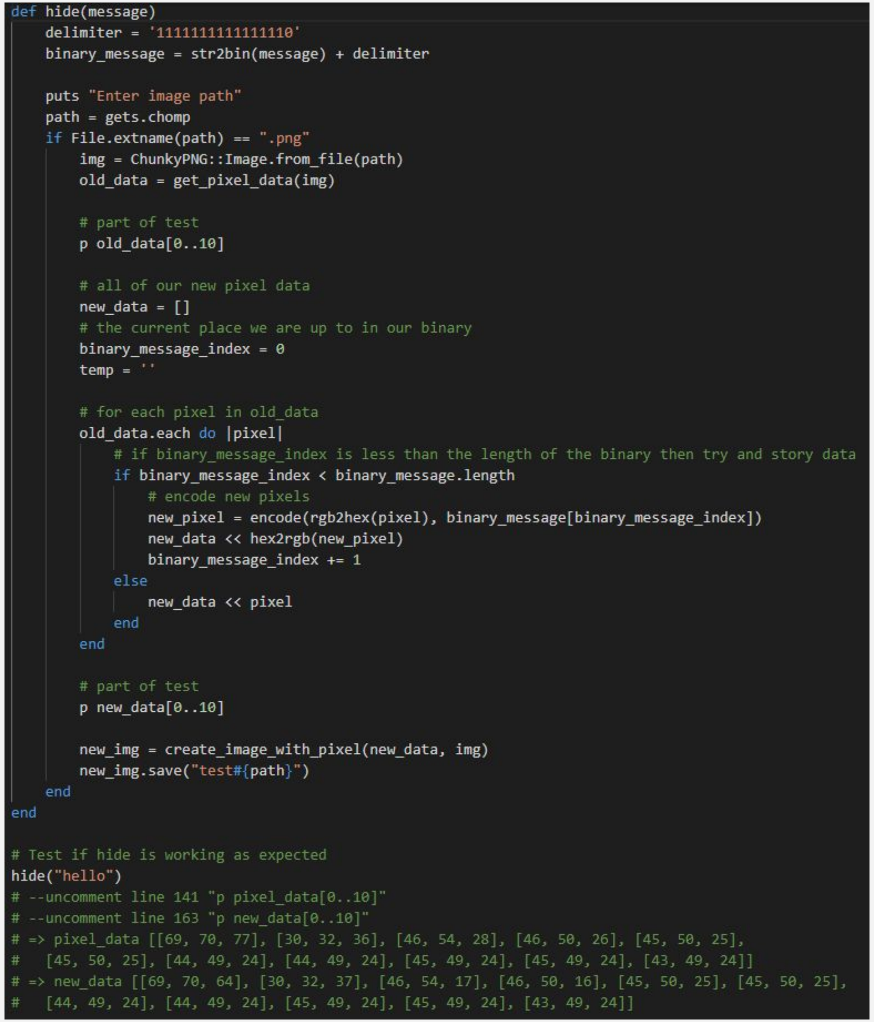
Code



```ruby
# Populates and returns an array with the rgb values for each pixel
def get_pixel_data(image)
    pixel_data = []
    (0..image.dimension.width-1).each do |x|
        (0..image.dimension.height-1).each do |y|
            rgb = [ChunkyPNG::Color.r(image[x,y]), ChunkyPNG::Color.g(image[x,y]), ChunkyPNG::Color.b(image[x,y])]
            pixel_data << rgb
        end
    end
    return pixel_data
end

# Test to see if get_pixel_data works as expected
# image = ChunkyPNG::Image.from_file('r_tiny.png')
# p get_pixel_data(image)
# => [[247, 247, 247], [247, 247, 247], [247, 247, 247], [247, 247, 247],
#       [247, 251, 251], [247, 253, 253], [247, 247, 247], ..., [243, 192, 193]]


# Creates an image from an array of pixel data
def create_image_with_pixel(pixels, image)
    new_img = ChunkyPNG::Image.new("#{image.dimension.width}".to_i,"#{image.dimension.height}".to_i, ChunkyPNG::Color::TRANSPARENT)

    pixel_index = 0

    (0..image.dimension.width-1).each do |x|
        (0..image.dimension.height-1).each do |y|
            new_img[x,y] = ChunkyPNG::Color.rgb(pixels[pixel_index][0], pixels[pixel_index][1], pixels[pixel_index][2])
            pixel_index += 1
        end
    end
    return new_img
end

# Test to see if create_image_with_pixel works as expected
# image = ChunkyPNG::Image.from_file('r_tiny.png')
# new_img = create_image_with_pixel(get_pixel_data(image), image)
# new_img.save("test.png")
# => test.png created in current directory
```

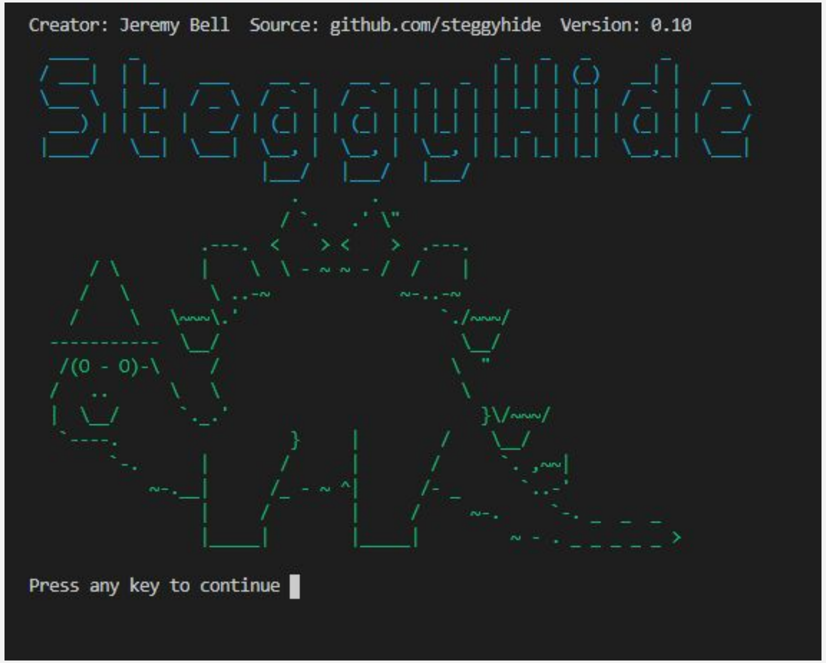
Code

- User inputs file path for image
- Convert message to binary
- Turn image into array of pixel data
- Check if the message has been hidden already
- Generate the new pixel array by calling the encode method which adds the binary bit to the hex value and then converting back to rgb pixel values and pushing into the new pixel array
- Use the new pixel array to create image and save image to new file

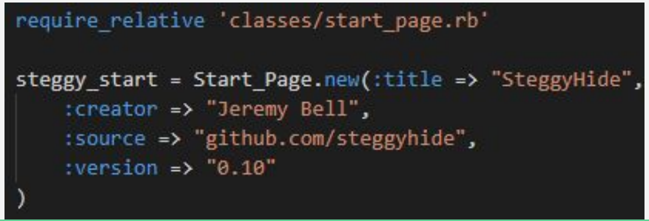


```ruby
def hide(message)
    delimiter = '1111111111111110'
    binary_message = str2bin(message) + delimiter

    puts "Enter image path"
    path = gets.chomp
    if File.extname(path) == ".png"
        img = ChunkyPNG::Image.from_file(path)
        old_data = get_pixel_data(img)

        # part of test
        p old_data[0..10]

        # all of our new pixel data
        new_data = []
        # the current place we are up to in our binary
        binary_message_index = 0
        temp = ''

        # for each pixel in old_data
        old_data.each do |pixel|
            # if binary_message_index is less than the length of the binary then try and story data
            if binary_message_index < binary_message.length
                # encode new pixels
                new_pixel = encode(rgb2hex(pixel), binary_message[binary_message_index])
                new_data << hex2rgb(new_pixel)
                binary_message_index += 1
            else
                new_data << pixel
            end
        end

        # part of test
        p new_data[0..10]

        new_img = create_image_with_pixel(new_data, img)
        new_img.save("test#{path}")
    end
end

# Test if hide is working as expected
hide("hello")
# --uncomment line 141 "p pixel_data[0..10]"
# --uncomment line 163 "p new_data[0..10]"
# => pixel_data [[69, 70, 77], [30, 32, 36], [46, 54, 28], [46, 50, 26], [45, 50, 25],
#   [45, 50, 25], [44, 49, 24], [44, 49, 24], [45, 49, 24], [45, 49, 24], [43, 49, 24]]
# => new_data [[69, 70, 64], [30, 32, 37], [46, 54, 17], [46, 50, 16], [45, 50, 25], [45, 50, 25],
#   [44, 49, 24], [44, 49, 24], [45, 49, 24], [45, 49, 24], [43, 49, 24]]
```

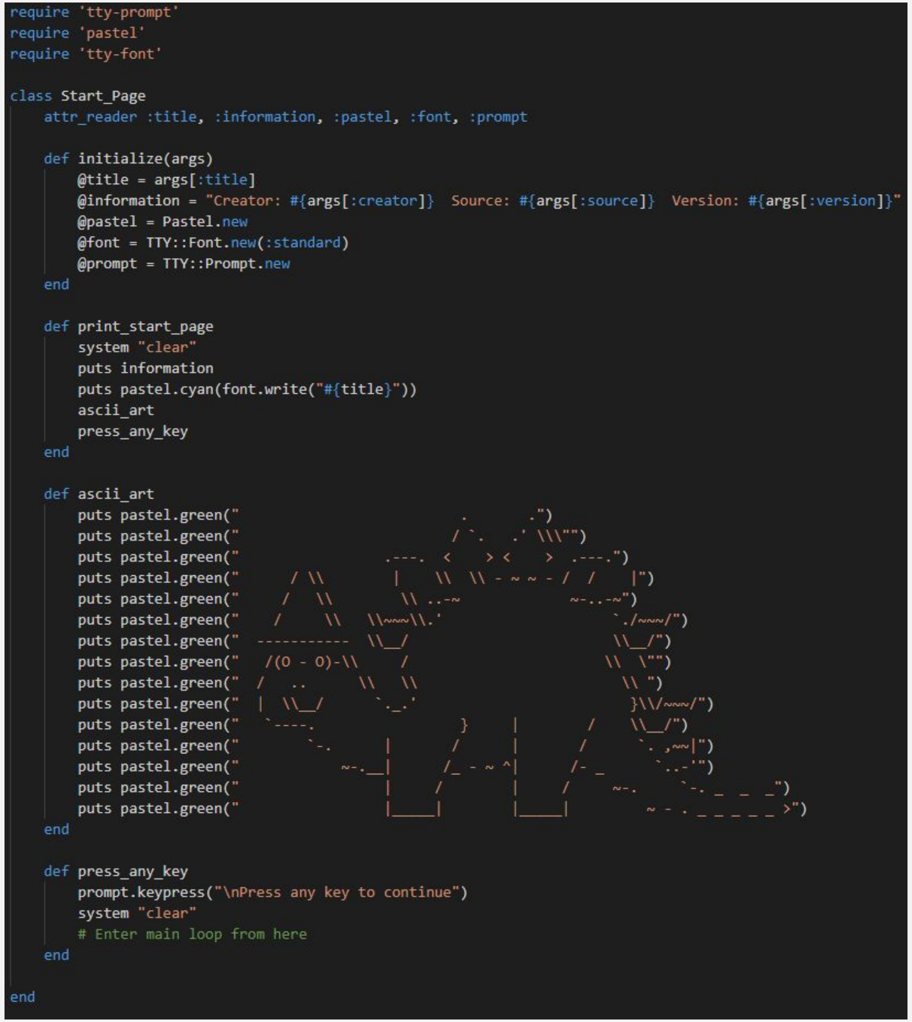
Start Screen



```
Creator: Jeremy Bell  Source: github.com/steggyhide  Version: 0.10
  ____  _                         _   _ _     _
 / ___|| |_ ___  __ _  __ _ _   _| | | (_) __| | ___
 \___ \| __/ _ \/ _` |/ _` | | | | |_| | |/ _` |/ _ \
  ___) | ||  __/ (_| | (_| | |_| |  _  | | (_| |  __/
 |____/ \__\___|\__, |\__, |\__, |_| |_|_|\__,_|\___|
                |___/ |___/ |___/
                          .       .
                         / `.   .' \"
                 .---.  <    > <    >  .---.
      / \         |    \  \ - ~ ~ - /  /    |
     /   \         \ ..-~                ~-..-~
    /     \   \~~~\.'                    `./~~~/
  -----------  \__/                        \__/
   /(O - O)-\     /                       \  \"
  /   ..      \   \                        \
  |  \__/     `._.'                         }\/~~~/
   `----.                 }     |        /    \__/
        `-.      |       /      |       /      `. ,~~|
            ~-.__|      /_ - ~ ^|      /- _      `..-'
                 |     /        |     /     ~-.     `-. _  _  _
                 |_____|        |_____|         ~ - . _ _ _ _ _ >

Press any key to continue
```

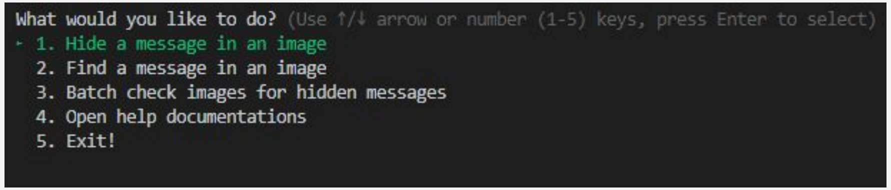


```ruby
require_relative 'classes/start_page.rb'

steggy_start = Start_Page.new(:title => "SteggyHide",
    :creator => "Jeremy Bell",
    :source => "github.com/steggyhide",
    :version => "0.10"
)
```

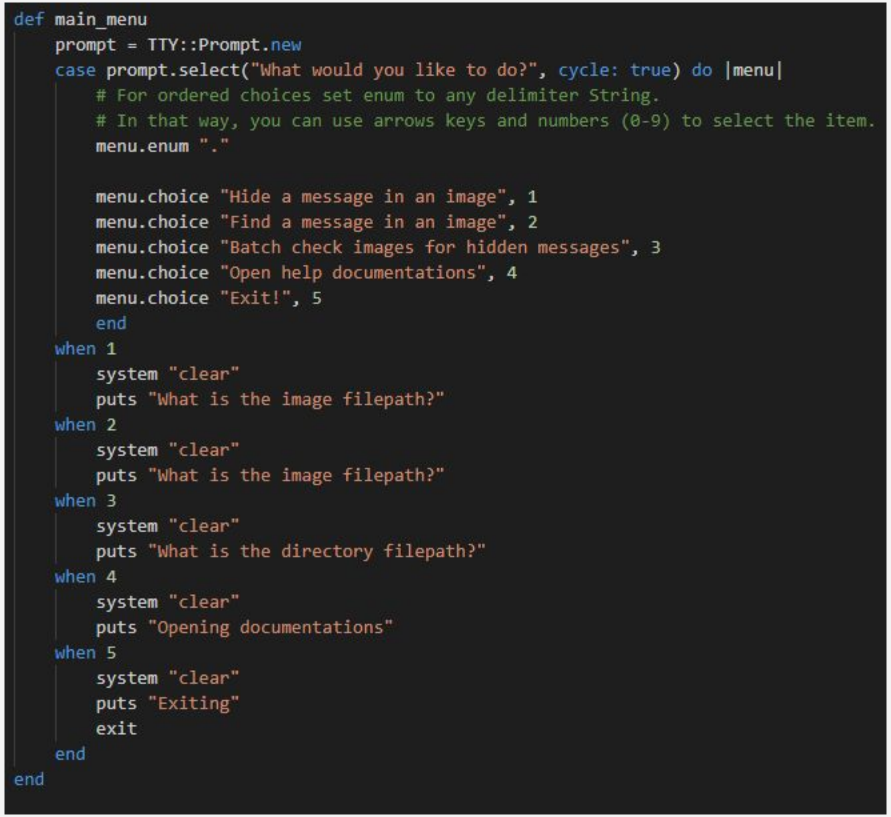


```ruby
require 'tty-prompt'
require 'pastel'
require 'tty-font'

class Start_Page
    attr_reader :title, :information, :pastel, :font, :prompt

    def initialize(args)
        @title = args[:title]
        @information = "Creator: #{args[:creator]}  Source: #{args[:source]}  Version: #{args[:version]}"
        @pastel = Pastel.new
        @font = TTY::Font.new(:standard)
        @prompt = TTY::Prompt.new
    end

    def print_start_page
        system "clear"
        puts information
        puts pastel.cyan(font.write("#{title}"))
        ascii_art
        press_any_key
    end

    def ascii_art
        puts pastel.green("                          .       .")
        puts pastel.green("                         / `.   .' \\\"")
        puts pastel.green("                 .---.  <    > <    >  .---.")
        puts pastel.green("      / \\         |    \\  \\ - ~ ~ - /  /    |")
        puts pastel.green("     /   \\         \\ ..-~                ~-..-~")
        puts pastel.green("    /     \\   \\~~~\\.'                    `./~~~/")
        puts pastel.green("  -----------  \\__/                        \\__/")
        puts pastel.green("   /(O - O)-\\     /                       \\  \"")
        puts pastel.green("  /   ..      \\   \\                        \\ ")
        puts pastel.green("  |  \\__/     `._.'                         }\\/~~~/")
        puts pastel.green("   `----.                 }     |        /    \\__/")
        puts pastel.green("        `-.      |       /      |       /      `. ,~~|")
        puts pastel.green("            ~-.__|      /_ - ~ ^|      /- _      `..-'")
        puts pastel.green("                 |     /        |     /     ~-.     `-. _  _  _")
        puts pastel.green("                 |_____|        |_____|         ~ - . _ _ _ _ _ >")
    end

    def press_any_key
        prompt.keypress("\nPress any key to continue")
        system "clear"
        # Enter main loop from here
    end

end
```

Main Menu



```
What would you like to do? (Use ↑/↓ arrow or number (1-5) keys, press Enter to select)
‣ 1. Hide a message in an image
  2. Find a message in an image
  3. Batch check images for hidden messages
  4. Open help documentations
  5. Exit!
```



```ruby
def main_menu
    prompt = TTY::Prompt.new
    case prompt.select("What would you like to do?", cycle: true) do |menu|
        # For ordered choices set enum to any delimiter String.
        # In that way, you can use arrows keys and numbers (0-9) to select the item.
        menu.enum "."

        menu.choice "Hide a message in an image", 1
        menu.choice "Find a message in an image", 2
        menu.choice "Batch check images for hidden messages", 3
        menu.choice "Open help documentations", 4
        menu.choice "Exit!", 5
        end
    when 1
        system "clear"
        puts "What is the image filepath?"
    when 2
        system "clear"
        puts "What is the image filepath?"
    when 3
        system "clear"
        puts "What is the directory filepath?"
    when 4
        system "clear"
        puts "Opening documentations"
    when 5
        system "clear"
        puts "Exiting"
        exit
    end
end
```