SIP: Simple Image Processing Language
Final Report

ICOM 4036: Prof. Wilson Rivera
Team Members:
Graciany A. Lebrón
Yomar Ruiz
Samuel Gonzalez
Luis Rivera

Introduction

Currently, there are a variety of image processing features available across multiple programming languages. Many of them require a lot of lines of code or are hard to use due to the technical jargon that is associated with digital signal processing; which also makes the documentation difficult to understand. Our focus in this project is to develop a language that is straightforward for users that are not necessarily knowledgeable of image processing or digital signal processing in general. We expect that the learning curve for our proposed programming language is far from steep so that users can swiftly modify images as they want to. Users can do this by importing an image and applying a variety of operations to an image such as scaling, sharpening, and feature extraction via our programming language. Our proposed language will be implemented using PLY, the python scanner/parser tool that will allow us to create the straightforward syntax of SIP. The python libraries of Scipy and Numpy will be used since they will allow us to manipulate the vectors/matrices associated with the images while also having some toolboxes for image/signal processing. Finally, matplotlib will be used to render the images and display the effects that the user executed.

SIP Grammar

<statement> -> <method> | <assignment>

<assignment> -> <img_assignment> | <method_assignment>

<method> -> <method_np> | <method_1p> | <method_2p> | <method_no>

<method_no> -> **METHOD_NO LP STRING RP**

<method_np> -> **ID DOT METHOD_NP LP RP**

<method_1p> -> **ID DOT METHOD_1P LP DIRECTION RP**

| **ID DOT METHOD_1P LP LEVEL RP**

| **ID DOT METHOD_1P LP STRING RP**

<method_2p> -> **ID DOT METHOD_2P LP INT COMMA INT RP**

<img_assignment> -> **ID EQUALS ID**

<method_assignment> -> **ID EQUALS** <method_no>

Grammar Legend

- Method_1P: Method with one parameter
- Method_2P: Method with two parameters
- Method_NP: Method with no parameters
- Method_NO: Method with no object

Language Tutorial

Steps of how to use our programming language:

1. Download SIP from our GitHub repository in the following link: https://sammy5430.github.io/SIP-Simple-Image-Processing-Language/
2. Make sure to have Python 3 (3.6.5) installed in your computer.
3. Install the following python libraries: PLY, Scipy, Numpy, Matplotlib, and scikit-image.
4. Run the sip.py file on your favourite IDE or terminal/command prompt.
5. Start using SIP Language!

Language Reference Manual

Image Types

- 2D Images
    - Also known as binary images, these types are only capable of performing tasks that do not require the image being modified to have 3 dimensions to represent them. Performing a wrong command will result in a warning explaining so.
- 3D Images
    - All types of images represented with 3 dimensions, each representing their respective colors (RGB).

Variables

All variables will only be images represented as arrays of numbers representing pixels and can only be initialized by assigning an image or assigning it to a command reading an image file.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img2 = img
```

read(“FILENAME)

- Reads an image from a file, used for variable initialization.
- **Parameters:**
  - **FILENAME:** String type parameter which represents the name of the file to be read.

Example:

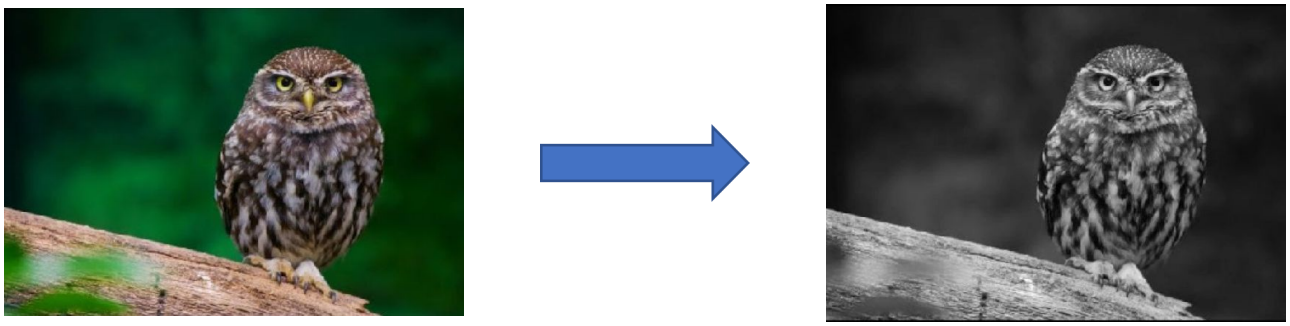
```
SIP >>img = read("owl.jpg")
```

grayscale()

- Converts a 3D image to a 2D image composed exclusively of different shades of gray.

Example:

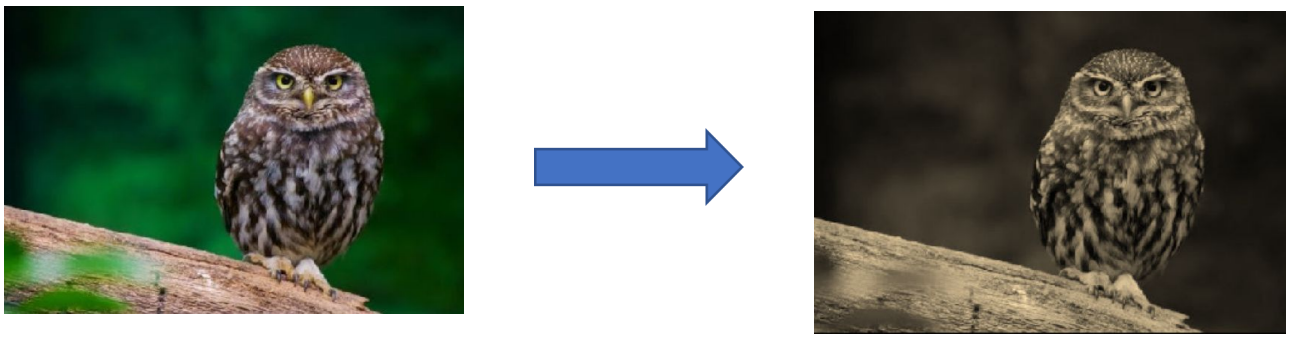
```
SIP >>img = read("owl.jpg")
SIP >>img.grayscale()
```



sepia()

- Converts an image to sepia tones, mostly used to represent aging of old images.

Example:

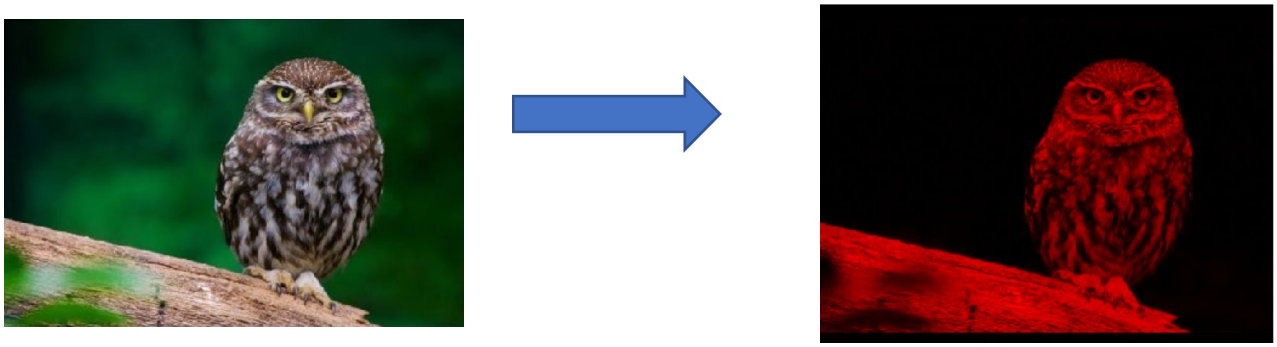


red()

- Extracts only the red component of an RGB image.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.red()
```

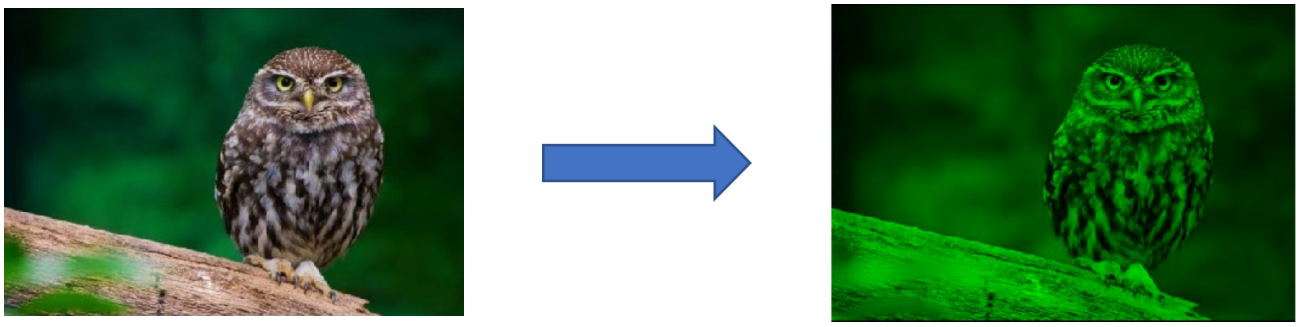


green()

- Extracts only the green component of an RGB image.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.green()
```

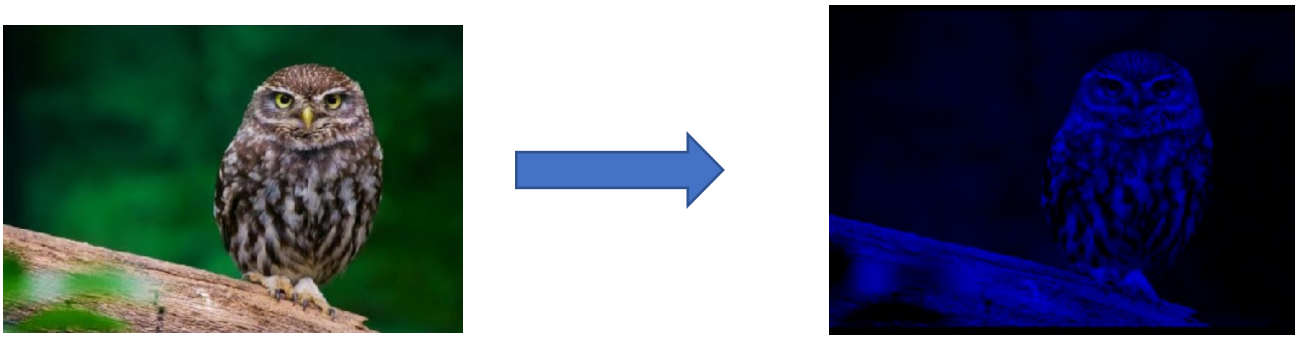


blue()

- Extracts only the blue component of an RGB image.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.blue()
```

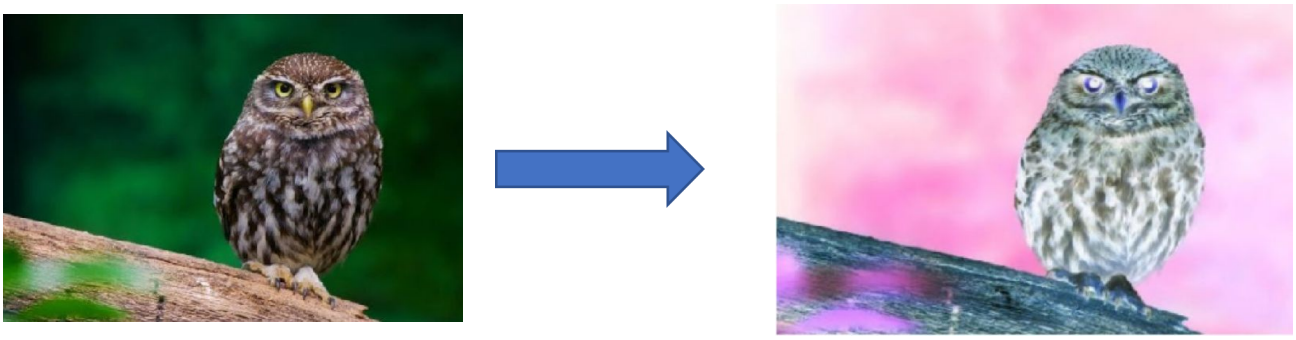


invert()

- Inverts the color of each individual pixel, creating the effect of a *negative* image.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.invert()
```

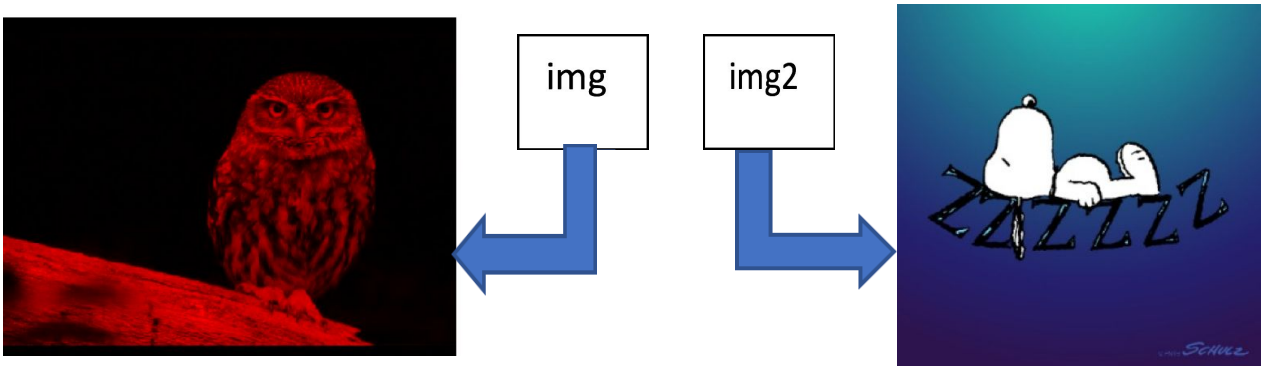


show()

- Shows the current state of the image stored in a given variable.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.show()
SIP >>img.red()
Keep Changes? (y/n):y
Changes to 'img' were saved successfully.
SIP >>img.show()
SIP >>img2 = read("test.jpg")
SIP >>img2.show()
```

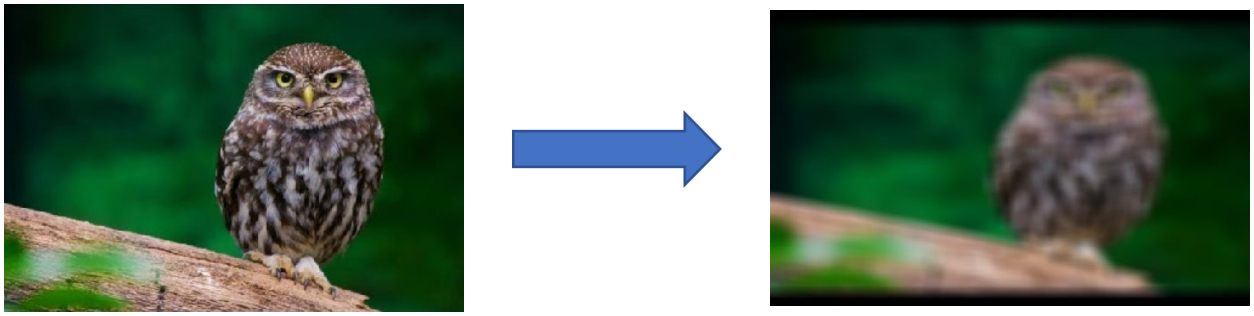


blur(LEVEL)

- It will reduce image noise and reduce detail to the image this function is being used on
- **Parameters:**
  - **LEVEL:** Will require the user to input intensity of the blur in the form of **medium**, **low** or **high**.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.blur(high)
```



rotate(DIRECTION)

- Will rotate the image 90 degrees to the side that was input.
- **Parameters:**
  - **DIRECTION:** Will require the user to input which direction would he like it to rotate to, **right** or **left**.

Example:

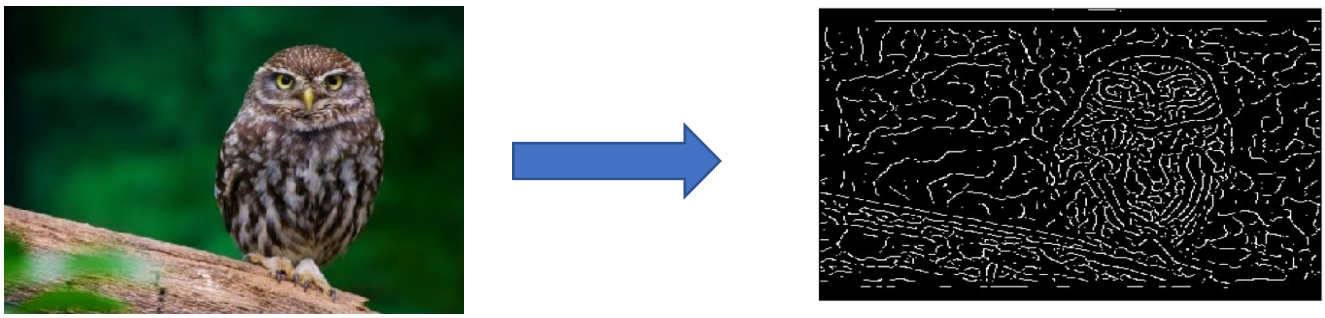
```
SIP >>img = read("owl.jpg")
SIP >>img.rotate(left)
```

edges(LEVEL)

- Finds the boundaries of objects within the images. Works by detecting discontinuities in brightness. **This method can only be called onto 2D images.**
- **Parameters:**
  - **LEVEL:** Will require the user to input intensity of the edges in the form of **medium**, **low** or **high**.

Example:

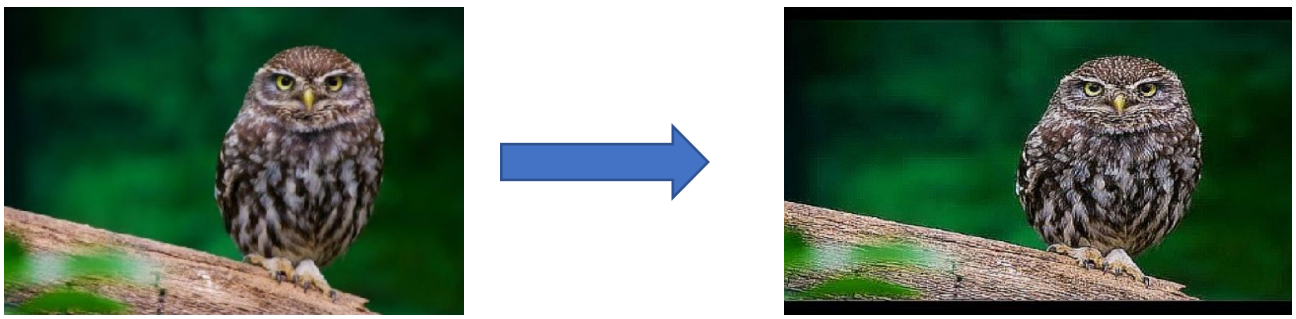
```
SIP >>img = read("ovl.jpg")
SIP >>img.grayscale()
Keep Changes? (y/n):y
Changes to 'img' were saved successfully.
SIP >>img.edges(medium)
```



sharpen(LEVEL)

- Emphasizes the differences of value between neighboring pixels within the image.
- **Parameters:**
  - **LEVEL:** Will require the user to input intensity of the sharpen in the form of **medium**, **low** or **high**.

Example:



save("FILENAME")

- Will save the copy of the image being used onto the project folder with the name given.
- **Parameters:**
  - **FILENAME:** Will require the user to input the name of the new file to be saved as well as its image format between quotation marks.

Example:

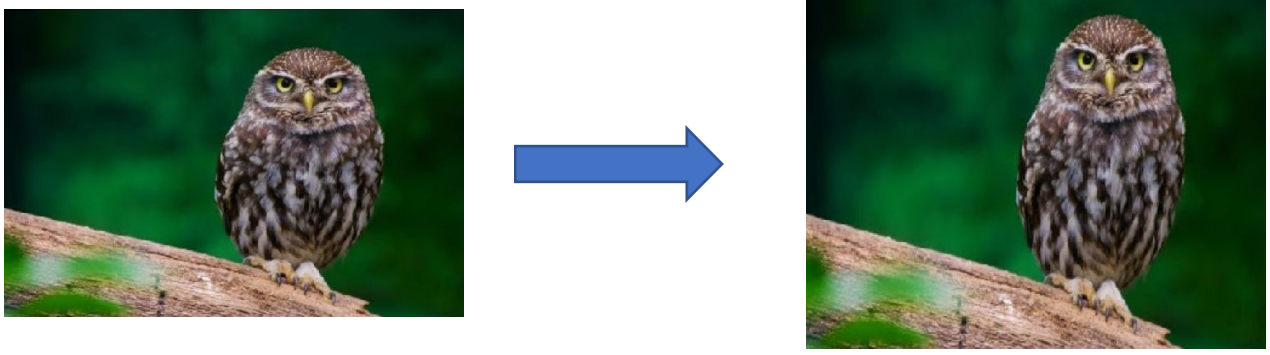
```
SIP >>img.save("testing.png")
Changes were successfully saved.
```

resize(height, width)

- Will resize the image to fit the dimensions used as parameters.
- **Parameters:**
  - **width:** the new width of the image,width must be positive integer.
  - **height:** the new height of the image, height must be positive integer.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.resize(500, 500)
```

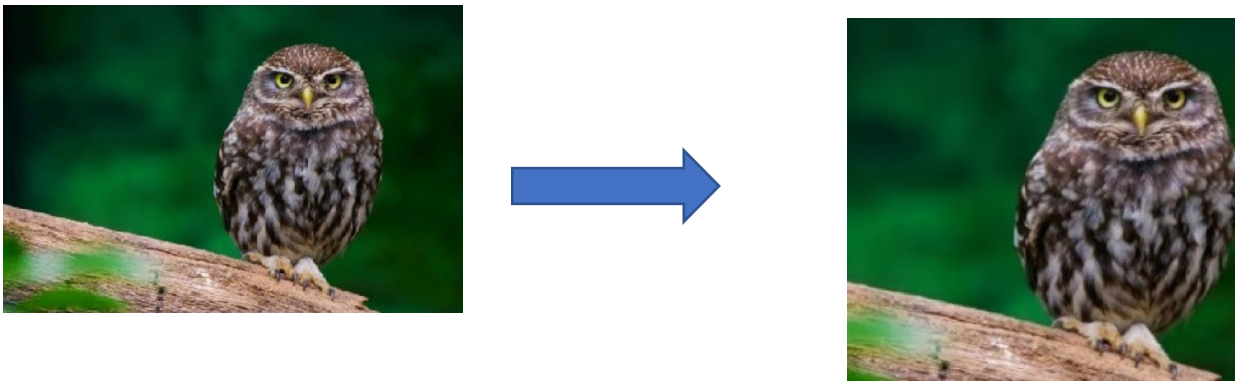


crop(height, width)

- Will crop the image using the dimensions given as parameters, the cropping is with respect to the center of the image.
- **Parameters:**
  - **width:** the width of the crop, width must be positive integer.
  - **height:** the height of the crop, height must be positive integer.

Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.crop(300,300)
```

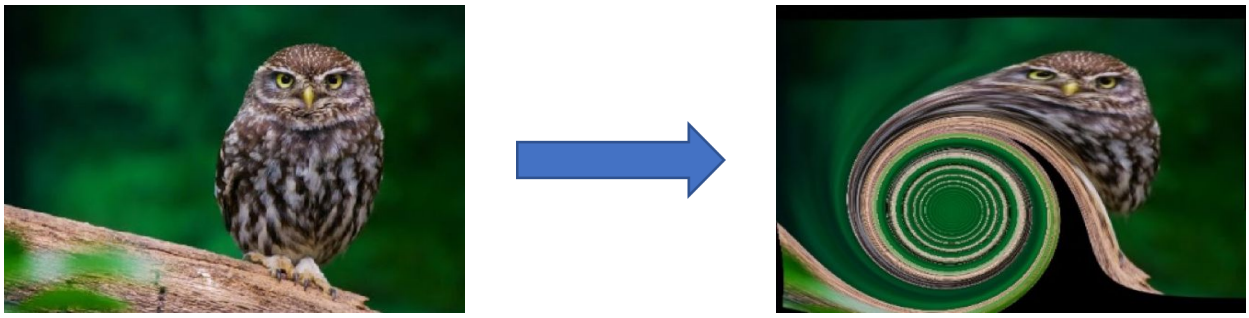


spiral(strength, radius)

- Will apply a spiral effect onto the image
- **Parameters:**
  - **strength:** the strength of the spiral, must be positive integer.
  - **radius:** the radius of the spiral, must be positive integer.

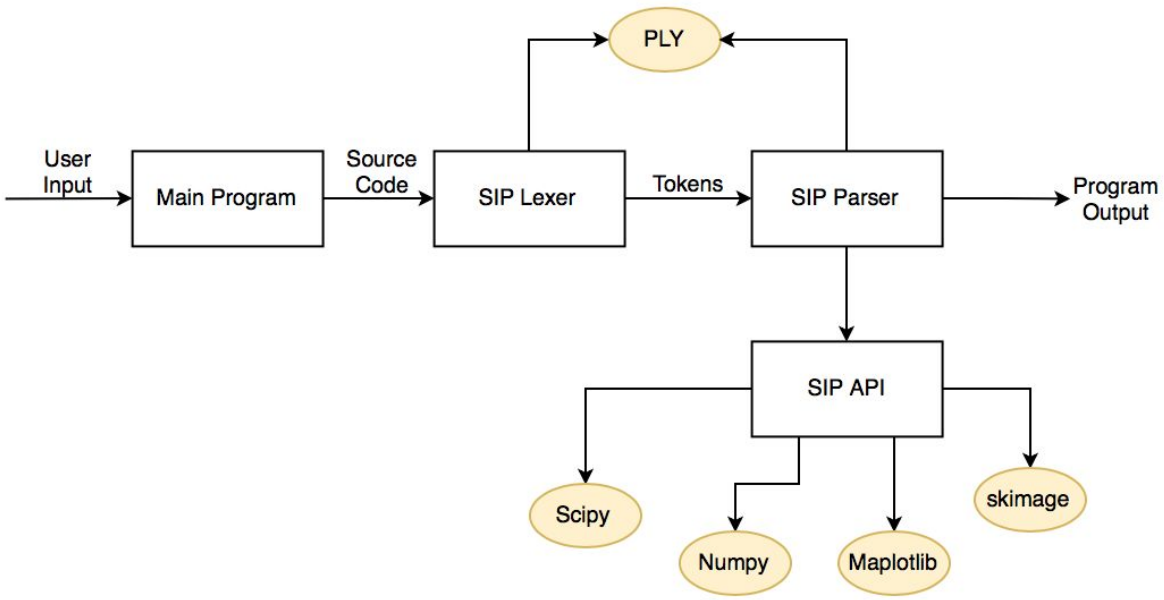
Example:

```
SIP >>img = read("owl.jpg")
SIP >>img.spiral(50, 200)
```



Language Development

I. Translator Architecture



II. Description: Interfaces between Modules

Main Program: Is the module that ties the SIP Language altogether, it waits for user input to the invoke the sip Lexer and Parser to then finally start the code generation process.

SIP Lexer: SIP Lexical analyzer was designed using the Library PLY. Here the token specifications are declared and the regular expression that is necessary to match the tokens given by the user when they input SIP code.

SIP Parser: The SIP Parser was designed using PLY as well. The tokens received from the lexer to the parser, we need to determine if they follow the grammar rules of our language. This module is in charge of specifying the grammar rules using a Backus-Naur Form (BNF) Automata. After the parser deemed the user

input as a valid grammar expression, then we can execute the corresponding intermediate code within SIP's API.

SIP API: This module was developed using the following python libraries: Scipy, Numpy, Matplotlib, and skimage. Here is where are intermediate code resides and it is executed depending on which grammar rule the parser identifies in the tokens that the user inputs. This is where the image transformations and operations occurs.

Python Libraries/Dependencies: The ovals in yellow in the figure above are all the libraries used in SIP Lang, these are: scipy, numpy, matplotlib, skimage, and ply.

III. Software Development Environment

The following Programs were used in the development process of SIP Language:

PyCharm: Is a Python IDE were we could write all develop, implement, and test, to make the SIP Language translator functional. In addition, the version of Python used in our code was Python 3.6.5.

Virtualenv: Is a tool to create isolated python environments and it is also package manager that helps you install the dependencies in your project.

GitHub Desktop: Is a version control software to share and monitor the code among our team of developers.

IV. Test Methodology

The modules in within SIP Lang where tested using Blackbox Testing, where testers provide the inputs and observe the outputs. If the output was incorrect with respect to their corresponding input, then further testing was done.

V. Programs to Test Translator

Some Examples of the Programs to Test the Lexer

The objective of the programs to test the lexer was to see if the lexer applied correctly the regular expression and identified the token related to that regular expression.

Prog 1:

```
Img = read("new.jpg")
Img
Sepia
;;
spiral(100,200)
red()
Owl = img
```

Prog 2:

```
Img = read("new.jpg")
Img.blue()
invert()
Sepia()
;;
spiral(100,200)
blur(medium)
red()
Owl = img
Img = Owl
img.save("edited.png")
```

Some Examples of the Programs to the Test Parser

The objective of the programs to test the parser was to see if the parser could find a grammar rule that applied to the tokens that the lexer identified.

Prog 3:

```
Img = read("new.jpg")
Img.blue()
img.invert()
Img = img.spiral(100,200)
Owl = img
Img = Owl
img.save()
img.blur("high")
img..red(100,100)
```

Prog 4:

```
Img = read("new.jpg")
Img.blue()
img.invert()
Img = newImg
img.sharpen("high")
img.resize(100,1000)
img..crop(100,100)
```

Examples of Programs to Test the Complete SIP Lang Translator

The objective of the programs to test the SIP translator was to see if our translator could identify which syntax that the user inputs is correct and that the corresponding piece of Intermediate code in SIP Language API executed successfully.

Prog 5:

```
Img = read(“new.jpg”)
Img.blue()
img.invert()
Img = img.spiral(--100,200)
Owl = img
Img = Owl
img.save()
img.blur(“high”)
img..red(100,100)
img.save(“edited.png”)
Copy = img
copy.grayscale()
copy.save(“gray.jpeg”)
```

Prog 6:

```
Img = read(“new.jpg”)
Img.blue(100)
img**.invert()
Img = newImg
Copy = img
img.sharpen(“high”)
img.resize(100,1000)
```

```
img..crop(100,100)
copy.crop(500,500)
copy.save("cropped.jpg")
```

CONCLUSION

The completed language implementation provides some of the more common image processing methods with a simpler syntax, appropriate error handling, and parameters that are easier to understand. Methods which previously required complex user input have been implemented on SIP to work with levels ('low', 'medium', and 'high'). Furthermore, the rotate method which normally requires the user to implement the number of degrees to rotate and the origin point for the rotation, has been simplified to work with 90-degree rotations with the origin set to the center of the image. After comparing code snippets between other implementations of these methods and the implementations done in this project, we can see a reduction in line of codes, which was one of the project's goals. In addition, we can perceive the simplicity of the parameters used in SIP, when comparing it with other implementations. Overall, this language provides higher most image processing methods with better accessibility, since users can understand it even with little or no previous knowledge of image processing languages.