Fixel
COMS W4115 Final Report

Kavita Jain-Cocks
kj2264@columbia.edu

Amrita Mazumdar
am3210@columbia.edu

Darien Nurse
don2102@columbia.edu

Nilkanth Patel
njp2110@columbia.edu

Matthew Patey
mep2167@columbia.edu

May 12, 2014

Contents

1 White Paper **3**
- 1.1 Introduction . . . 3
- 1.2 Related Work . . . 3
- 1.3 Goals of Fixel . . . 4
- 1.4 Architecture . . . 5
- 1.5 Features . . . 5

2 Language Reference Manual **7**
- 2.1 Introduction . . . 7
- 2.2 Lexical Conventions . . . 7
- 2.3 Types . . . 9
- 2.4 Scope . . . 10
- 2.5 Expressions . . . 10
- 2.6 Grammar . . . 15

3 Tutorial **18**
- 3.1 Introduction . . . 18
- 3.2 Basic Functionality . . . 18
- 3.3 Built-in Functions . . . 21
- 3.4 Primitives and Colors . . . 22
- 3.5 Variables and Expressions . . . 23
- 3.6 Building Custom Functions . . . 24
- 3.7 Control Flow . . . 24
- 3.8 Pixels and Colors . . . 26
- 3.9 Conclusion . . . 27

4 Project Plan **28**
- 4.1 Initial Concept Development . . . 28
- 4.2 Project Management Tools . . . 28
- 4.3 Planning . . . 28
- 4.4 Development . . . 29
- 4.5 Testing . . . 29
- 4.6 Responsibilities . . . 29
- 4.7 Timeline . . . 30

5 Language Evolution **30**
- 5.1 Compiler Tools . . . 31
- 5.2 Libraries . . . 31
- 5.3 Consistency . . . 31

6 Translator Architecture **32**
- 6.1 Lexer . . . 32
- 6.2 Parser . . . 32
- 6.3 Code Generator . . . 32

6.4 Integration 34

7 Development and Runtime Environment **34**

8 Test Plan **35**
8.1 Tools 35
8.2 Challenges 36
8.3 Relevant Files 36
8.4 Continuous Integration Server 37

9 Conclusions **37**
9.1 Lessons Learned 37
9.2 Advice for Future Teams 41
9.3 Suggestions 41

A Git Log **42**

B Source Code **58**
B.1 Scripts 58
B.2 Lexer 60
B.3 Parser 65
B.4 Code Generation 73
B.5 Testing 79

1 White Paper

1.1 Introduction

Over the last few years, businesses and technology centered on social media have experienced extraordinary growth. Sharing articles, videos, and personal information has never been easier. Photo-sharing, in particular, has become popular due to the proliferation of smartphones, tablets, and other smart devices on the market. This function is typically provided through both websites and applications that facilitate the upload and display of images. Social networking services like Instagram, Twitter, Pinterest, and Snapchat have capitalized on this trend by provide their users with new and interesting ways to share photos and other media.

With the growth of photo sharing and social media in mind, our group has decided to create a language that would facilitate and enhance the online photo sharing experience. Fixel is an easy-to-learn, simple to use programming language for editing images. It is targeted towards individuals who often share information through social media and offers them more control over their photo editing experience than standard image sharing applications. The syntax is structured to resemble that of posts on social networking services, leveraging the use of hashtags (`#`) and at symbol (`@`). This syntax will help those who are familiar with social networking services easily adapt to Fixel.

Fixel offers various tools for image editing including, but not limited to, cropping, filtering, scaling, and blurring. These tools are typically complex and may require high-level mathematics to implement. Fixel's built in functions abstracts these tools such that individuals with absolutely no programming experience can uses them. To maintain simplicity and optimum portability, Fixel uses Python as its runtime environment. Python can run on most modern operating systems and provides a wide variety of image editing tools from the Python Imaging Library (PIL).

The remainder of this whitepaper is organized as followed: Related Work is a summary of the Python programming language and its role in developing Fixel. Goals enumerates a list of goals Fixel hopes to achieve. Architecture outlines the technical, backend features of Fixel. Finally, Features describes the various image editing features that Fixel will implement.

1.2 Related Work

Most modern programming languages already have external libraries to simplify image processing functionality, focusing on image filtering and manipulation. While these external libraries and frameworks facilitate similar image filter construction, the resulting implementations are exceedingly complex and cater more to seasoned programmers with a significant image processing background.

Python specifically has a robust and powerful Python Imaging Library (PIL), which provides functionality from image canvas cropping, aspect-ratio fitting, and file format conversion to complex filter constructions. These libraries can be

very useful for users seeking to have very fine control over how their images are edited, however a background in signal processing is required to fully harness the capabilities of these libraries.

Similarly, Processing, and its sister language Processing.js, are languages built on top of Java and JavaScript, respectively, to facilitate visually-based programming projects for non-programmers or novice programmers. While Processing is a more user-friendly environment for constructing simple image-processing filters, it has grown to become a more robust and powerful development environment which is not as conducive to an introductory understanding of programming. It also requires a background in signal processing to construct image filters.

1.3 Goals of Fixel

Fixel will be a user friendly language that offers its users the opportunity to create a customizable experience in an environment that mimics one they are already familiar and comfortable with.

User Friendly

One of the most important goals to be pursued in the construction of this language is that it will be intuitive for the user. It is assumed that a large portion of the target audience may have limited programming experience and the aim of Fixel is that they will be able to fully utilize it with a minimal learning curve. Since the goal of Fixel is to bring complex photo-editing abilities to people who do not necessarily have a programming background, Fixel will have an easy-to-learn syntax.

Mimic Syntax of Social Media

The syntax of Fixel will mimic that of social media and currently existing photo modification tools, such as Instagram and Twitter. This will serve multiple purposes. The first is that the user will hopefully feel more at ease using this new tool. Secondly, this will provide continuity between Fixel and currently existing options. Finally, it will allow users inexperienced with programming a more entertaining and accessible experience since their programs will not look like conventional code.

Customizable Experience

By offering users increased photo alteration options, Fixel aims to offer increased autonomy over the picture sharing experience. Fixel allows for increased originality since users will not be hindered by predetermined filters and effects. As a result, it will be possible to create attractive and increasingly complex images to be shared. Fixel gives the user increased freedom without requiring extensive knowledge of image editing libraries thereby increasing the opportunity for creativity and artistry.

Dynamic Typing

Another important user-friendly feature of Fixel is that it is dynamically typed. Dynamic typing will allow users to easily create powerful and compact programs without writing the tedious boilerplate required of statically typed languages. Beginner programmers can focus on what they want their program to do without having to worry about defining complex types to do it. A dynamic language will simplify the process for both new and experienced programmers. Beginners can avoid lower level details of reference type, casting and polymorphism. Experts can obtain the powerful effects of typing features such as polymorphism without having to dig into contracts and interfaces set up in static languages.

1.4 Architecture

Portable

An easy-to-use language is still not accessible if it is limited to a specific environment and requires a complex and tedious setup. That is why we chose to use Python as our runtime environment. Python was designed to be highly extensible, support a variety of programming paradigms, and be able to run on any major operating system. Additionally, a variety of Python implementations exist in addition to the standard compiler and VM that allow Python to be run on different platforms such as the JVM and .NET. This versatility and variety makes Python not only accessible on a number of platforms, but also highly integratable in many different types of projects. We will be taking advantage of this by running Fixel programs in Python and writing our translator in Python. This will allow deployment and/or development anywhere that Python can be run.

Robust

In order to increase ease of use without decreasing capability, our language will abstract away as much of the challenging/tedious parts of programming as possible. This will include simplification and/or internal handling of errors, stringing together function calls and determining parameters from programmer's input and assisting with control flow such as loops and switches. This abstraction requires our translator to intelligently handle situations that which we wish to hide from the programmer.

1.5 Features

Fixel is constructed to offer maximum customization for those interested in image editing and enhancement. Using a simple syntax, programmers are able to automate a number of edits to their photos using built-in functions. This process is intuitive, and users are awarded the flexibility to define their own set of inputs to customize the result of each transformation. The functions available to Fixel programmers are:

Blur

Using an existing library's set of Fourier transforms, Fixel allows for the blurring of edges. Alternatively, if the user desires, they may select a center point (using its x and y pixel coordinates) and have the system blur out the areas surrounding it.

Grayscale

A simple bit-by-bit translation can convert an image with RGB hex values to its corresponding grayscale version. The user can also define the level of "tint" applied to the pixels (from a range zero to one), if they wish to move the image towards being grayscale, but not remove color entirely.

Invert

This feature has no inputs. It simply takes the user's test image and returns a negative version of the image.

Overlay

The overlay feature is predicated on the user having some degree of familiarity with RGB color schemes and the basic principles behind color hex values. They are able to use the overlay command to manipulate the hue and saturation of the image by manually entering the offset for these two values.

Brightness and Contrast

Users may specify how they'd like change the brightness and contrast of their images with percentages (for example, changing the brightness to be 125% would increase it by 25%).

Border

The user can select from a collection of pre-configured border styles for their images and define the thickness of these borders. Some borders are more ornate than others.

Crop

By providing a set of coordinates the user would like to retain, they can crop off the remainder of the image and keep only the part that falls within the predefined space. The user has the option of cropping to a rectangle, a square, a circle, or an ellipsoid, all of which are configured using four coordinates.

Scale

The scale function allows users to change the size of the image. They may do so either by providing a new final size for the image (and have the pixels stretch to fit this new size), or they can retain the current images proportions and specify a percentage increase or decrease in size.

Collage

Fixel is built to support a series of pre-defined collage "types" that the user can select and then customize using their images. This is done by understanding the collage configurations and assembling arrays filled with images in the order in which they expect them to appear in the configuration.

Rotate

This command allows the user to rotate the image any number of degrees they wish. If the resulting image is not a rectangle, white space will be added to ensure that it is.

Caption

By defining the text of a caption, the user can use this function to insert text below their image or collage. They may also specify the font size, type, style, and weight of the caption using properties borrowed from CSS syntax.

2 Language Reference Manual

2.1 Introduction

This Language Reference manual provides the syntax details of the Fixel language which facilitates image editing and makes writing the code for image modification easier. This manual begins with a description of lexical conventions, gives information about the syntax used in Fixel programs, and then provides a grammar.

2.2 Lexical Conventions

This section describes the lexical conventions used in Fixel. This includes detailing the identifiers, keywords, and reserved characters.

Comments

Fixel uses a double forward slash `//` to denote a single line comment. A single line comment refers to the area from the end of the double slash until a newline character is seen. Fixel does not support multiline comments. Therefore, each line of a comment must begin with the double forward slash.

Identifiers

Identifiers are used to represent a primitive, object, or a function. All identifiers must start with either a letter or an underscore. However, in the rest of the name, it may contain letters, numbers, and underscores. On the other hand, reserved characters as well as spaces, and any other whitespace are prohibited. All identifiers are case sensitive.

Keywords

Certain identifiers are reserved and can only be used as keywords. Their usage is prohibited in other contexts. A list of keywords is shown below:

for	while
if	else
return	in
true	false
and	forp
not	

Reserved Characters

Certain characters are reserved due to their necessity to the grammar. The following is a list of these characters:

+	-	*	/
(	)	[	]
:	//	,	"
#	@	<	>
=	!=	>=	<=

Forbidden Words

Certain identifiers are forbidden since they correspond to keywords in python. A list of forbidden keywords is shown below:

del	from	as	elif
global	with	assert	pass
True	False	yield	break
execpt	import	print	class
exec	raise	continue	finally
is	def	lambda	try

2.3 Types

Basic Types

Fixel accounts for several different types of primitives. They are described below.

- **Integer**: Integer constants follow the same rules as they would in Python. Therefore they are implemented using a long in C resulting in at least 32 bits of precision.
- **Boolean**: A boolean refers to a value that can take either true or false.
- **String**: A string in Fixel, like a string in Python, refers to an ordered list of characters. Additionally, this sequence of characters that are strung together is preceded and followed by double quotes.

Derived Types

- **Image**: The Image type is used to refer to the images passed as program arguments when the program is run. Fixel built in functions operate on these image types. Images have height and width properties, and allow individual pixels to be both read and written.
- **Color**: Color is a wrapper for a 3 integer tuple. It contains exactly 3 integers each of which correspond to the red, green, and blue (RGB) values of a color respectively. Colors support basic arithmatic operations between with numbers and other colors.
- **Pixel**: Accessible only from a forp loop, a pixel is a wrapper around a color, an x and a y coordinate. Assigning a color to a pixel variable updates the pixel of the Image that the pixel is associated with.

Constants

- **Character Constants**: This refers to a sequence of characters of length 1 or more with an unchangeable value. Certain character constants are necessarily represented with a corresponding escape sequence. The list is shown below.

newline	`\n`
horizontal tab	`\t`
carriage return	`\r`
backslash	`\\`
not equal	`!=`

- **Boolean Constants**: Boolean constants will be either true or false.

2.4 Scope

Variables that share scope are required to have unique names. A variable's identifier will have scope beginning from where it is declared and persist until the end of the function it is declared in.

Global Scope

All functions have global scope, and can be called from anywhere in the program.

Function Scope

Variables that are declared inside functions, will have a scope that lasts for the duration of the function and will expire when the function is exited. The implicit image and list variables created from the program arguments have scope throughout the entirety of the main function.

2.5 Expressions

Intermediate Expression

Intermediate expressions can be a primary expression or a function expression.

```
intermediate-expression:
    primary-expression
    function-expression
```

Primary Expressions

Primary expressions can be variables, string literals, boolean and number primitive literals, list creation, and parenthesized expressions. All variable names are preceded by the `@` symbol. Variable properties can be accessed with the `.` operator and list or image sequenced values can be accessed with brackets.

```
primary-expression:
    variable-access-expression
    STRING
    NUMBER
    TRUE
    FALSE
    '[' parameters ']â
    '(' expression ')â

variable-access-expression:
    variable-expression
    variable-access-expression '.' ID

variable-expression:
```

```
    variable
    variable_expression '[' parameters ']'

variable:
    '@' ID
```

Function Calls

A function call starts with the # symbol, followed by the function name and an optional comma separated argument list.

```
function-expression:
    '#' ID parameters

parameters:
    primary-expression
    parameters ',' primary-expression
    epsilon
```

Logical NOT operator

The logical NOT operator is left-associative and includes the keyword `not`. This operator and the multiplicative operators have the highest priority.

```
logical-NOT-expression:
    primary-expression
    'not' logical-NOT-expression
```

Unlike other operators, the logical NOT operator is unary. If the operand evaluates to true or a boolean equivalent, the logical NOT expression yields a boolean with value false, or vice-versa.

Multiplicative Operators

Multiplicative operators are left-associative and include the symbols * and /. These are the operators and logical NOT operator have the highest priority.

```
multiplicative-expression:
    logical-NOT-expression
    multiplicative-expression '*' logical-NOT-expression
    multiplicative-expression '/' logical-NOT-expression
```

The * (multiplication) operator yields the product of its arguments. If one operator is a number and the other a sequence, sequence repetition is performed; a negative repetition factor yields an empty sequence. If both are colors then a new color is formed where each value is the product of the two corresponding values of the operands. If one is a color and the other a number, then a new color is formed by multiplying each field of the color by the number.

The / (division) operator yield the quotient of its arguments. Division by zero raises the ZeroDivisionError exception. If both are colors then a new color is formed where each value is the quotient of the two corresponding values of the operands. If one is a color and the other a number, then a new color is formed by dividing each field of the color by the number, or the number by each field, depending on order.

Additive Operators

Additive operators are left-associative and include the symbols + and -. These operators have a lower priority than multiplicative operators.

```
additive-expression:
    multiplicative-expression
    additive-expression '+' multiplicative-expression
    additive-expression '-' multiplicative expression
```

The + (addition) operator yields the sum of its arguments. If one operator is a number and the other a sequence, the sequences are concatenated. If both are colors then a new color is formed where each value is the sum of the two corresponding values of the operands. If one is a color and the other a number, then a new color is formed by adding each field of the color to the number.

The - (subtraction) operator yields the difference of its arguments. If both are colors then a new color is formed where each value is the difference of the two corresponding values of the operands. If one is a color and the other a number, then a new color is formed by subtracting each field of the color from the number, or the number from each field, depending on order.

Relational Operators

Relational operators are left-associative and include the symbols <, >, <=, >=. These operators have a lower priority than additive operators.

```
relational-expression:
    additive-expression
    relational-expression '<' additive-expression
    relational-expression '>' additive-expression
    relational-expression '<=' additive-expression
    relational-expression '>=' additive-expression
```

The < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to) operators evaluate to a boolean value of true if the respective relation is true and false if it is false.

If the operands are numbers, then the operands are compared arithmetically. If the operands are Strings, then the strings are compared lexicographically using the numeric equivalents of their characters.

Equality Operators

Equality operators are left-associative and include the symbols == and !=. These operators have a lower priority than relational operators.

```
equality-expression:
    relational-expression
    equality-expression '==' relational-expression
    equality-expression '!=' relational-expression
```

The == (equals) and != (not equals) operators evaluate to a boolean value of true if the respective relation is true and false if it is false.

If the operands are numbers, then the operands are compared arithmetically. If the operands are Strings, then the strings are compared lexicographically using the numeric equivalents of their characters. If the operands are of different types that cannot be converted to a common type, then the result is always unequal.

Logical AND operator

The logical AND operator is left-associative and includes the keyword 'and'. This operator has a lower priority than equality operators.

```
logical-AND-expression:
    equality-expression
    logical-AND-expression 'and' equality-expression
```

Consider the expression: x and y. The operator first evaluates x; if x is false, the expression evaluate to false; otherwise, the expression is evaluated as y.

Logical OR operator

The logical OR operator is left-associative and includes the keyword 'or'. This operator has a lower priority than the logical AND operator.

```
logical-OR-expression:
    logical-AND-expression
    logical-OR-expression 'or' logical-AND-expression
```

Consider the expression: x or y. The operator first evaluates x; if x is true, the expression evaluate to true; otherwise, the expression is evaluated as y.

Assignment Expressions

Assignment expressions are either the assignment of an assignment expression to an identifier, or a logical or expression.

```
assignment-expression:
    ID '=' assignment-expression
    logical-OR-expression
```

Assignment is evaluated after all other operations in an expression.

Declarations

A function declaration starts with the function name and is followed by an optional comma separated argument list and a mandatory statement block.

```
function-definition:
    ID parameter-declaration block

parameter-declaration:
    epsilon
    variable
    parameter-declaration ',' variable
```

Statements

Statements are units of code executed to change the program state, but are not associated with a value. Statements include control flow, loops and expression statements.

```
statement:
    expression-statement
    selection-statement
    iteration-statement
    return-statement
```

Statement Blocks

Statements can be grouped into a logical block that determines a unit of scope and execution, so that all statements it contains can be executed when a function is called, a loop body is entered, or control otherwise selects the block for execution. The block begins with a colon, followed immediately by a newline. All lines in the block after the line containing the colon have an indentation greater than or equal to the previous line. A line not conforming to this rule marks the end of the block.

```
block:
    ':' NEWLINE INDENT statement-list DEDENT
```

A line with greater indentation than the previous line is interpreted as being preceded by an INDENT token. A line with indentation less than the proceeding line is interpreted as being preceded by a DEDENT token.

Expression Statements

An expression statement is simply an expression followed by a new line that separates the statement from the following statement.

```
expression-statement:
    expression NEWLINE
```

Selection Statements

Selection statements allow the programmer to specify control flow.

```
selection-statement:
    'if' expression block
    'if' expression block 'else' block
```

In both if and if else statements the if block is executed if the boolean expression evaluates to true. In if else statements, the else block is executed if the boolean expression evaluates to false.

Iteration Statements

Iteration statements allow the user to write loops. While loops execute until a boolean expression evaluates to false. for loops allow iteration over each element in a sequence, and forp, or for pixel, loops allow iteration over every pixel in an image.

```
iteration-statement:
    'for' variable 'in' variable block
    'forp' variable 'in' variable block
    'while' expression block
```

The for loop must be given a variable with type list. In each iteration of the loop, the first identifier is assigned the next item in the list. The forp loop is similar to this, except that the second variable must be an Image. Additionally, if the pixel variable is assigned a color, this change is reflected in the image. While loops evaluate a boolean expression at the start of each iteration, and abort if it evaluates to false.

2.6 Grammar

```
program:
    statement-list
    statement-list translation-unit

block:
    ':' NEWLINE INDENT statement-list DEDENT

translation-unit:
    function-definition
    translation-unit function-definition

function-definition:
    ID parameter-declaration block

parameter-declaration:
```

```
    epsilon
    variable
    parameter-declaration ',' variable

statement-list:
    statement
    statement-list statement

statement:
    expression-statement
    selection-statement
    iteration-statement
    return-statement

return-statement:
    'return' expression-statement

expression-statement:
    expression NEWLINE

selection-statement:
    'if' expression block
    'if' expression block 'else' block

iteration-statement:
    'for' variable 'in' variable block
    'forp' variable 'in' variable block
    'while' expression block

expression:
    assignment-expression

assignment-expression:
    variable-expression '=' assignment-expression
    logical-OR-expression

logical-OR-expression:
    logical-AND-expression
    logical-OR-expression 'or' logical-AND-expression

logical-AND-expression:
    equality-expression
    logical-AND-expression 'and' equality-expression

equality-expression:
    relational-expression
```

```
    equality-expression '==' relational-expression
    equality-expression '!=' relational-expression

relational-expression:
    additive-expression
    relational-expression '<' additive-expression
    relational-expression '>' additive-expression
    relational-expression '<=' additive-expression
    relational-expression '>=' additive-expression

additive-expression:
    multiplicative-expression
    additive-expression '+' multiplicative-expression
    additive-expression '-' multiplicative expression

multiplicative-expression:
    logical-NOT-expression
    multiplicative-expression '*' logical-NOT-expression
    multiplicative-expression '/' logical-NOT-expression

logical-NOT-expression:
    intermediate-expression
    'not' logical-NOT-expression

intermediate-expression:
    primary-expression
    function-expression

primary-expression:
    variable-access-expression
    STRING
    NUMBER
    TRUE
    FALSE
    '[' parameters ']'
    '(' expression ')'

function-expression:
    '#' ID parameters

parameters:
    primary-expression
    parameters ',' primary-expression
    epsilon

variable-access-expression:
```

```
    variable-expression
    variable-access-expression '.' ID

variable-expression:
    variable
    variable_expression '[' parameters ']'

variable:
    '@' ID
```

3 Tutorial

3.1 Introduction

The following section is a brief tutorial of Fixel. Here, we highlight key aspects of the language through example programs that demonstrate how users can build up functionality in their Fixel code. At its core, Fixel is a language for image editing. But given the right tools, it can be a powerful tool in automating the process of touching up and fixing images.

This tutorial will focus on the basics of the language—from built-in functions to primitive and derived types—and expands on how to use these tools in Fixel to construct powerful image filters. The basis for this language is the ability to customize and combine common image filters to generate unique image treatments that can then be reused as needed. The given examples demonstrate how to couple filters with input images to produce more visually interesting output images, and then to perform the same transformation on multiple images with relative ease.

The Fixel language facilitates image editing and makes writing code for image processing more user-friendly. It aims to be human-readable, modular, and easy to learn, regardless of one's familiarity with other programming modalities. Its syntax is based on that of common social media services, and can easily be understood as sample conversations in these programs.

3.2 Basic Functionality

Fixel is, at its core, an image processing language. We therefore begin by introducing how it deals with images, as illustrated by a few, very basic Fixel program.s

Using Images

Before getting into the basic construction of Fixel programs, it is first necessary to address the use of images in Fixel. They serve as the foundation for any sort of image processing one might wish to test.

The purpose of Fixel is to create filters to edit images easily and efficiently. The language is used to write executable filters that can then be applied to an

input image to produce an output image. Consequently, all Fixel programs are written with the purpose of modifying a given input image, or images.

Memory for an image is dynamically allocated by the system at run time.

So how does one work with images? When the program is run, Fixel creates an image variable for each image of the form "image0", "image1", etc. accessible in the main fuction. Fixel also creates a variable called images which is a list of all the images created by Fixel. Variables are referenced in Fixel by preceding the variable name with the "@" symbol. Using the variable name image0, we can now access certain attributes of the image, namely width, height, and color space. Let's look at an example of this using one of Fixel's built-in functions, described in detail in later sections:

```
#grayscale @image0
------------
gray.fxl
```

To run this program, the user executes the command:

```
> ./fixel gray.fxl input-image.jpg
```

The result of this program is a new image file, input-image-fixel.jpg, which is a grayscale version of the image edited using one of Fixel's built-in functions, `#grayscale`. As seen in the preceding example, functions are called using the `#` to indicate a function name.

Part of Fixel's appeal is its ability to work with multiple images within a single instance of a filter program. To perform an action on multiple images, simply stack them on top of on another and run functions on them:

```
#grayscale @image0
#grayscale @image1
#grayscale @image2
------------
gray-multi.fxl
```

To run this program, the user executes the command:

```
> ./fixel gray-multi.fxl image0.jpg image1.gif image2.png
```

This program will result in the three input images passed as arguments through the execution command being reproduced in grayscale, in the same directory as their original name with `-fixel` appended to the file name.

As can be seen from this execution command, for every image defined in a Fixel program, a separate argument needs to be appended to the execution command with the path of the input image.

Additionally, the images need not have the same file extension within a single Fixel program—so long as the files are of valid image format, they can be used with Fixel.

Variables can also be used to declare non-image types, like integers, strings, lists and so on. Variables may be of the types listed later in the Variables section of the Tutorial and in the accompanying Language Reference Manual.

Hello World

Fixel's Hello World program takes an image as input and overlays the text "Hello, World!" over the photo. This program demonstrates using Fixel to manipulate an input image using one of Fixel's built-in functions, `#caption`.

```
#caption @image0, "Hello, World!"
------------
hello-world.fxl
```

To run this program, the user executes the command:

```
> ./fixel hello-world.fxl image0.jpg
```

The input and resulting output photos of this program are shown below:

image1.jpg image1-fixel.jpg

Figure 1: Single image being captioned by Fixel.

How does this program work? Let's deconstruct the single code statement made in `hello-world.fxl`.

The first word is the function call, denoted by the # symbol. Because `#caption` is a built-in function (which are described in detail in the next section) it needs no function definition.

The next word, `@image0` refers to the input image, passed to the program during execution. The interpreter will throw an error during runtime if a program refers to `@image1` without passing in two input images.

The final aspect of the code statement is the string in quotes, "Hello, World!" The built-in function `#caption` takes in an argument, the string to be overlaid on the image. Arguments to a function are passed as comma-separated statements. If we wanted to overlay a different text with the `#caption` function, we could write:

```
#caption @image0, "Hey!"
```

or

```
#caption @image0, "?!?!!@()#$*#@"
```

to display different text on the image.

To further demonstrate the functionality presented in `hello-world.fxl`, let's caption multiple images in one Fixel program:

```
#caption @image0, "Hello, World!"
#caption @image1, "Hello, again."
#caption @image2, "Hi!"
------------
hello-world-multi.fxl
```

We run this program by executing:

```
> ./fixel hello-world-multi.fxl image0.jpg image2.jpg image3.jpg
```

The resulting images appear as follows:

image1.jpg, image2.jpg, image3.jpg

image1-fixel.jpg, image2-fixel.jpg, image3-fixel.jpg

Figure 2: Multiple images being captioned by Fixel.

3.3 Built-in Functions

One goal of Fixel is to allow the user to construct varied and complex image filters without the need for involved programming. This is partly enabled through the implementation of Fixel's built-in functions. Fixel comes with a number of built-in functions to assist the user in creating human-readable, accessible image filters, and allows the user to stack them to produce more interesting effects.

As seen in `hello-world.fxl`, functions are invoked using the `#` symbol.

Grayscale (a function without arguments)

```
#grayscale @image0
------------
gray.fxl
```

Grayscale is a simple built-in function that converts an image to a grayscale image while still outputting a conventional RGB image.

Overlay (a function with multiple arguments)

```
#overlay @image0, (#color "red"), 10
------------
red-overlay.fxl
```

Overlay allows the user to overlay an image with a color fill. The intensity of the fill can be modulated by the percentage argument, which specifies what opacity the fill is to be set at. An example of `red-overlay.fxl` in action can be seen below:

Figure 3: `red-overlay.fxl` illustrates how to use the built-in overlay function.

Full List of Built-in Functions

Presented here are basic instructions on how to call the functions built into Fixel on inputted images. For an explanation of what each of these functions does, please consult the white paper.

- Grayscale — `#grayscale @image0`
- Overlay — `#overlay @image0 color,opacity`
- Brightness — `#brighten @image0 percentage`
- Contrast — `#contrast @image0 percentage`
- Border — `#border @image0 size, color`
- Crop — `#cropit @image0 left, top, right, bottom`
- Scale — `#scale @image0 percentage`
- Collage — `#collage outputImage, listOfImages, width, height`
- Rotate — `#rotate @image0 angle`
- Caption — `#caption @image0 text`

3.4 Primitives and Colors

Fixel supports the following primitive types: `int`, `boolean`, `string`. These are useful for using data (like dimensions) from one image to make changes to another.

Fixel also provides support for colors as a 3-integer tuple representation of an RGB value created either with those RGB values, or a hex value (example: `#f0f0f0`), or a string identifying the color (example: `red`), pixels which contain a color and x and y positions, and images which incude height, width and a pixel collection.

3.5 Variables and Expressions

Fixel can also be used to read certain attributes of images and then use those values to modify other images passed into the program. This is done by reading these values into program variables and then manipulating them using using common arithmetic operators: `+`, `-`, `*`, and `/`.

Say, for instance, we'd like to write a program that takes two images with different dimensions and stretch or compress them to be double the height and width of a third image. In that case, our program would look like this:

```
@desiredWidth = @image0.width*2
@desiredHeight = @image0.height*2
@imagesToStandardize = [@image1,@image2]

for @currentImage in @imagesToStandardize:
        #stretch @desiredWidth, @desiredHeight, @currentImage
------------
standardizing-dimensions.fxl
```

`standardizing-dimensions.fxl` first instantiates variables for the three input images. It then reads the height and width of `image0` into variables called `desiredWidth` and `desiredHeight`, doubling the value immediately using the `*` operator. Next, it loads the images to standardize into a list variable and cycles through them using a `for` loop to stretch them to double the dimensions of `image0`. This could also be done without the `for` loop, but it is used here to illustrate a sample syntax for this kind of operation. `for` and `if` statements will be explored further in the "Control Flow" section.

We run this program by executing:

```
> ./fixel standardizing-dimensions.fxl image0.jpg image1.jpg image2.jpg
```

The resulting images appear as follows:

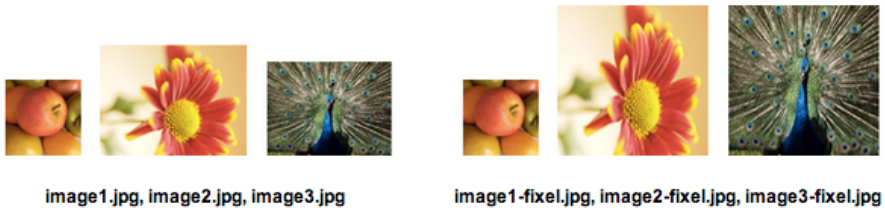

Figure 4: Output of the standardizing-dimensions.fxl program.

This process can, of course, be extended to any of the image's attributes. It can also be used to dynamically determine input arguments for Fixel's colorization functions. We can, for instance, use a step function to output sequential images with incremental increases in opacity or redness.

3.6 Building Custom Functions

Like most other programming languages, Fixel also allows users to group together and manipulate built-in functions to create custom ones. This is where the true power of Fixel lies—by empowering users with the ability to customize the filters they wish to apply to their images, they are no longer limited to the pre-determined combination of image transformations most image sharing applications offer them.

Custom functions are easy to design and implement. They are defined by beginning a line with the name of the function, followed by the variable names for the function's arguments (separated by a comma and an optional space), and finally a colon. This sequence of characters indicates to Fixel that we are about to define a function, rather than just call it on an image variable. We then add a line break, and stack the functions or statements we'd like the function to include in a tab-indented block.

Here's a sample program that first establishes a function that stretches an image to be the size of indicated by its input arguments, then converts it to grayscale and adds a caption that says `"Welcome to My Computer!"`:

```
#generateWallpaper @image0, 1600, 1200

generateWallpaper @imageName, @desktopWidth, @desktopHeight:
        #stretch @imageName, @desktopWidth, @desktopHeight
        #grayscale @imageName
        #caption @imageName, "Welcome to My Computer!"
------------
creating-wallpapers.fxl
```

We are first defining the function generateWallpaper and establishing that it has three input arguments: imageName, desktopWidth and desktopHeight. We then stack three built-in functions within `#generateWallpaper`. The first, `stretch`, passes the three input arguments as its own. The other two, `grayscale` and `caption` have static input arguments. Once we've defined our custom function, we then can apply it to `@image0` using the desired parameters (in this case, a width of `1600` and a height of `1200`).

We run this program by executing:

```
> ./fixel creating-wallpapers.fxl image0.jpg
```

The resulting images appear as follows:

3.7 Control Flow

The Fixel language allows for two types of loops: `for` and `while` loops. In addition, users may employ `if` or `if...else` statements. The syntax for these loops is demonstrated in the following sections.

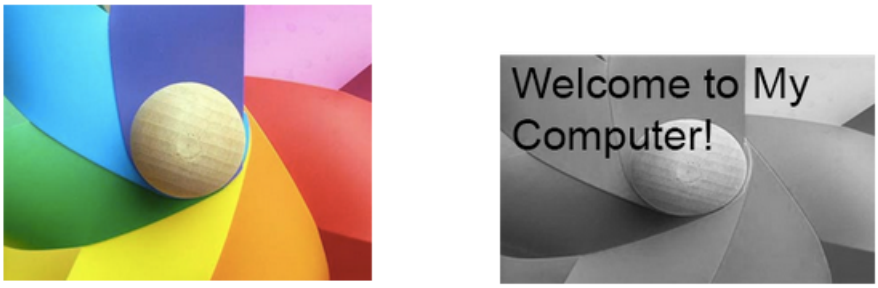

Figure 5: Output of the generateWallpaper Fixel program.

`for` Loops

`for` loops in Fixel are written in the same way as in most other programming languages. It is especially useful for using the same set of transformation functions on a multitude of images at once. An example of how to use the for loop can be seen in `standardizing-dimensions.fxl` above.

`for` loops are formatted as such: `for`, followed by a variable, followed by the keyword "in," followed by another variable, followed by a colon, followed by a line break, followed by a tab-indented list of commands to execute.

`while` Loops

Similarly, `while` loops can help users make incremental changes to photos as it processes them one by one. In this case, as long as the condition in the initial line holds true, the functions within the statement continue to be executed. For instance, say if we have a list of six images in the variable `@imagesList`, and we'd like their opacities to range from `40` to `100`, with each one's being ten percent higher than the previous one. In this case, we can create a `while` loop in our program like so:

```
@imageCount = 0
@currentOpacity = 1

while @imageCount<6:
    #opacity @images[@imageCount], @currentOpacity
    @currentOpacity = @currentOpacity + 1
    @imageCount = @imageCount + 1
------------
incremental-opacity.fxl
```

`while` loops are structured in almost the exact same way as `for` loops.

`if` And `if...else` Statements

Fixel also supports the use of `if` and `if...else` statements to direct control flow. This can be used to apply special transformations to certain images based on their properties. For example, if we'd like to compress all images with a pixel width less than `200` to `200`, we can do so with the `minimum-width.fxl` program:

```
for @currentImage in @images:
    if @currentImage.width<200:
        @proportionalHeight = @currentImage.height*(200/@currentImage.width)
        #stretch @currentImage, 200, @proportionalHeight
------------
minimum-width.fxl
```

Here, the program cycles through the images in `@images` (created automatically at program start) and checks to see if their width is less than `200`. If it is, then the program stretches them to have a width of `200` and retain a proportional height. In the same way, if we wanted the program to also standardize every image with a width above `200` to a width of `300`, we could use an `if...else` statement:

```
for @currentImage in @images:
    if @currentImage.width<200:
        @proportionalHeight = @currentImage.height*(200/@currentImage.width)
        #stretch @currentImage, 200, @proportionalHeight
    else:
        @proportionalHeight = @currentImage.height*(300/@currentImage.width)
        #stretch @currentImage, 300, @proportionalHeight
------------
minimum-maximum-width.fxl
```

Tab-indentation and line breaks hold the same amount of importance in `if` and `if...else` statements as in `while` and `for` loops.

3.8 Pixels and Colors

While there is plenty of fuctionality provided by the built in fuctions, more advanced users may find that they would prefer to use tools that provide a lower level of control. This is provided in Fixel with the use of pixels and colors. The colors at individual pixels of an image can be accessed by and set using the syntax "@image [@x, @y]". This can be useful for setting and obtaining individual pixels. The for pixel loop can be used to easily access all pixels in an image.

Forp Loop

The syntax for a pixel loop is "forp", pixel variable name, "in", image variable name. This will loop over every pixel in the image variable and assign it to the pixel variable. pixel types have color, x and y properties. Additionally, if the pixel variable is assigned to in the body of the loop, the corresponging pixel of the image is updated. Below is a simple forp example that inverts an image.

```
forp @pixel in @image0:
@pixel = 255 - @pixel.color
------------
invert.fxl
```

We run this program by executing:

```
> ./fixel creating-wallpapers.fxl image0.jpg
```

The resulting images appear as follows:

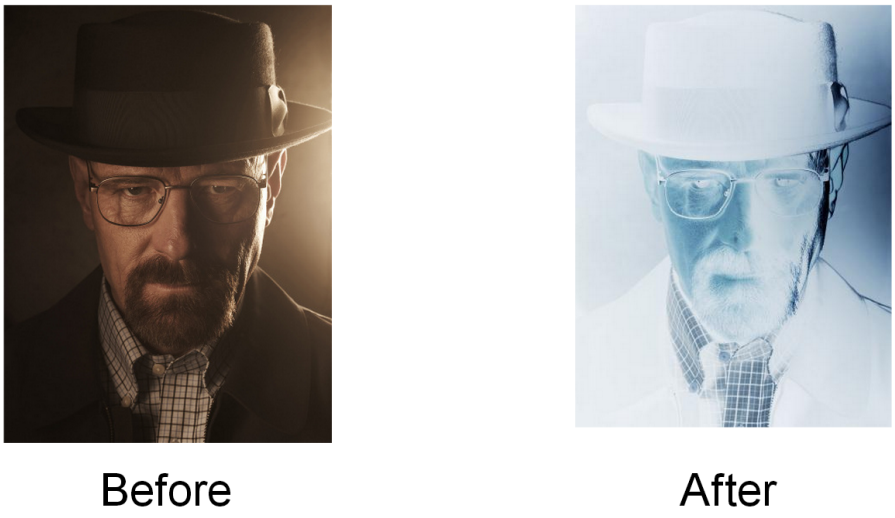

Figure 6: Output of the invert Fixel program.

This loops through the whole image and sets each pixel to its inverse. Note that colors support arithmatic with both numbers and other colors.

3.9 Conclusion

This language tutorial for Fixel covers to core concepts for programming image processing filters, and will hopefully serve as a tool for users in getting started with the language. A more comprehensive analysis of the language's characteristics can be found in the Language Reference Manual, and a complete explanation of how the built-in functions work can be found in the white paper.

4 Project Plan

Throughout the development of our language, Fixel has continued to evolve. During this process, our team has used various tools to aid in organization and consistency. Continuous reganization and communication has proved invaluable as we have worked to create, maintain, and meet the requirements of our development plan.

4.1 Initial Concept Development

Our group began developing project ideas at the very beginning of February. We began this process by all brainstorming various ideas on a Google Doc. In this way we ensured that everyone had the opportunity to contribute and enough time to explore possible areas of interest. Following online discussion we convened for our first team meeting on February 3rd to narrow down our options. We quickly came to consensus and decided on an image processing language. Due to the widespread sharing of images on social media tools such as Instagram and Facebook, it became clear that we wanted to include features in our language that tied together elements and conventions found in these products.

4.2 Project Management Tools

It was clear that, due to the large nature of the project, as well as the fact that the entire team would be contributing over the course of the semester, it would not be trivial to keep everyone up to date. However, we quickly realized that maintaining strong communication was integral to the success of the project. As a result, we worked early on to create an environment that made collaboration and communication easy. The primary tools we used in this effort were email, Google Docs, and Git hosted on BitBucket. Email was used for updates as well as questions in between our weekly meetings while Git was used for version control. We utilized Google Docs extensively for many purposes including, but not limited to, collaborating on the written deliverables and tracking the decisions made at meetings. Finally, the meeting tool When2Meet ended up being very useful in order to determine meeting times.

4.3 Planning

Beginning on February 3rd, we met as a team every Monday evening. This provided us with an opportunity to discuss the work we had accomplished in the previous week, raise any issues or quesitons we had found regarding Fixel, and assign work for the coming week. In addition to this meeting, we would often meet during the week in smaller groups, since we typically assigned tasks to multiple people. These groups were determined based on coinciding availability in the coming week and the times of these meetings were decided upon by the individuals in the subgroup. Throughout this process we relied on the feedback

we received from our Language Tutorial and Language Reference Manual. Additionally, after implementing a large portion of the decided upon functionality we met with Professor Aho as well as our TA, Ming-Ying Chung. These meetings helped to shape the final weeks of our project as well as realize some new directions we could take with our language.

4.4 Development

Throughout the development of Fixel, we followed the necessary levels of compiler design. Having created our repository on Bitbucket we were able to develop on separate branches when necessary and then merge back into the master branch. In order to properly facilitate informed development, we aimed to keep commit messages as descriptive as possible.

We initially implemented a skeleton of the lexer and the parser separately, and once this had been accomplished we proceeded to continue development in parallel based on the development goals set at our weekly meeting. Once it was possible to link the two units together, we began to implement the code generator. We also developed the necessary built-in functions in Python simultaneously, and linked those together once code generation was complete. We continued to develop and update all of the portions until we were satisfied that our language met the design goals.

4.5 Testing

In order to ensure that throughout the development process the new code added did not introduce new bugs to our translator, we unit tested each individual stage. These tests were mainly responsible for checking that the output at each step, as well as the final output, correctly corresponded to that section's input. The testing plan is covered in more detail in chapter 8.

4.6 Responsibilities

In order to ensure that collaboration was not only fair, but also as efficient as possible, we decided on all weekly coding responsibilities during our weekly meetings. These were detailed on a Google Document associated with that week's meeting. This document provided everyone with a reminder of the information covered during that particular session as well as kept track of what contribution was expected. We worked to ensure that everyone got to participate in the development of each level of the compiler as well as testing. In order to maintain consistency we made sure that one of the people in the group for each section had worked on it the previous week or had experience in that particular section. Each team member's primary areas of work will be discussed in more detail in chapter 6. In addition, everyone was in charge of a particular portion of the devlopment at a high level as follows:

Project Manager - Kavita Jain-Cocks

Language and Tools Guru - Amrita Mazumdar

System Architect - Darien Nurse
System Integrator: Nilkanth Patel
Tester and Validator - Matthew Patey

4.7 Timeline

The following is the timeline we followed for the development of Fixel:
February 3, 2014....Decide on language idea
March 3, 2014....Determine basic language syntax
March 24, 2014....Create full grammar
April 7, 2014....Generate single line programs in python
April 14, 2014....Create Skeleton of lexer and parser
April 21, 2014....Complete backend python code
April 28, 2014....Generate multiline programs in python
May 5, 2014....Turn fixel source code into executable python
May 12, 2014....Additional functionality, complete Presentation and Final Report

5 Language Evolution

During the development of Fixel, we chose to follow an iterative approach, starting with the simplest Fixel programs and building up our implementation from there. Feedback and advice from past groups also supported this development strategy. Consequently, we identified a hierarchy of program features to implement, and added them to our language in a series of phases.

In Phase 1, we sought to implement an extremely basic version of Fixel, where the user could successfully write and execute one-line programs invoking a built-in function, with the end result of editing an image. Although the Fixel programs were simple in construction, writing the underlying back-end involved crafting a full-featured lexer, constructing a complete grammar for the parser, and including rudimentary code generation. Moreover, we had to formulate all the built-in functions, as well as integrate the compiled Fixel with our native libraries.

We then moved on to Phase 2: implementing support for multi-line programs and the additional power they afford to the user. Development in this phase included multiple function calls, variables as parameters to functions, modifying multiple images, and built-in functions utilizing multiple image objects. In this stage of Fixel's implementation, we detected and removed many ambiguities in our grammar, as well as grew our code generation modules to accommodate the more complex programs Fixel would support. After resolving our grammar's ambiguities, we also found that the control flow we'd defined worked seamlessly without significant additional effort.

In Phase 3, we added functionality for supporting Color and Pixel. We chose to add these types in the third phase rather than earlier so as to ensure that the Fixel language could adequately create basic filters before providing additional

means of control and image modification. In this phase, we also finalized some ambiguities in the requirements we would impose upon the structure of programs written in Fixel, such as the ordering of function definitions and invocations. By implementing our grammar and code generation in an earlier phase and then revisiting the structure of Fixel programs in this phase, we were able to make more educated decisions about how users would choose to write Fixel programs and how to best match those use cases with our underlying system architecture.

5.1 Compiler Tools

After evaluating our goals for Fixel as well as our group's programming background, we made the design decision to use Python as the target language for our compiler. Additionally, we chose to implement the compiler in Python. This allowed us to use the well-documented *PLY* (Python Lex-Yacc) lexer and parser tools to develop our translator, as well as utilize Python's runtime interpreter to facilitate more user-friendly development.

5.2 Libraries

We chose to construct our built-in functions on top of `PIL` (Python Imaging Library). Using PIL but abstracting away its lower levels provided us with the ability to support many complex image processing algorithms while focusing on the user-facing interface to these functions in Fixel.

For testing, we chose to use Python's built-in unit testing framework, `unittest`, as well as the external libraries `Mock` to assist in isolating logical units of our code, `Nose` to assist in running the tests and `Coverage` to track our code coverage. These libraries allowed us to quickly and effectively achieve testing coverage across our code base during the development process.

5.3 Consistency

When implementing the initial LRM we developed, PLY's Yacc tool as well as our test cases quickly demonstrated the gaps and ambiguities in our grammar. As we resolved these rules, we were sure to also update our LRM specification in a timely fashion so that our written documents would stay in sync with our technical implementation.

We applied a similar workflow to the development of built-in functions. Initially we constructed a list of necessary built-ins and developed our library from that list. As development progressed, we found that a certain function could be reduced to two more fundamental built-ins, or some types of workflows could benefit from a new built-in. This required repeated additions to the built-in functions library and their integration with the code generation modules.

6 Translator Architecture

At the highest level, our translator can be said to have three distinct stages of translation: lexing, parsing, and code generation. A high-level block diagram is included for clarity.

6.1 Lexer

The first step of our translation process is lexing. The Lexer is written in PLY's lexing tool. The Lexer continuously processes the input string and matches its contents with specified tokens. The lexer was written by our Project Manager, Kavita, and our Language and Tools Guru, Amrita.

6.2 Parser

The second step of our translation process is parsing, which is written using PLY's yacc parsing tool. The Parser contains all the rules of Fixel's grammar and constrcts an abstract syntax tree (AST) from the tokens produced by the Lexer. The tokens are produced as a LexToken containing a type and value for each token. The parser evaluates and classifies the tokens while constructing an AST with a hierarchy of Node objects. The Node object contains any children, a type for the Node, and a leaf value if applicable. The parser was written by our System Architect, Darien, and our Language and Tools Guru, Amrita.

6.3 Code Generator

The AST produced by the parser is passed into the third step of our translation process, the Code Generator. The code generator is implemented in two steps: tree processing and post-processing. These steps are called by a higher-level Translator module.

Tree Processing

In the Tree Processing stage, we recursively process Nodes in the AST based on a Node's value. The result is appended to an output string to be passed to later stages of the translator Certain types of Nodes, such as function defintions, indent/dedent, and variable Nodes, have custom processing steps that are independently implemented. Other Nodes are simply added to the output string. The Tree Processing stage of Code Generation was written by our Tester and Validator, Matt.

Post-Processing

After the tree is parsed into requisite Python output, file handling, built-in function imports, and other Python code needs to prepended and appended to the output string to produce proper functionality. This step also handles file

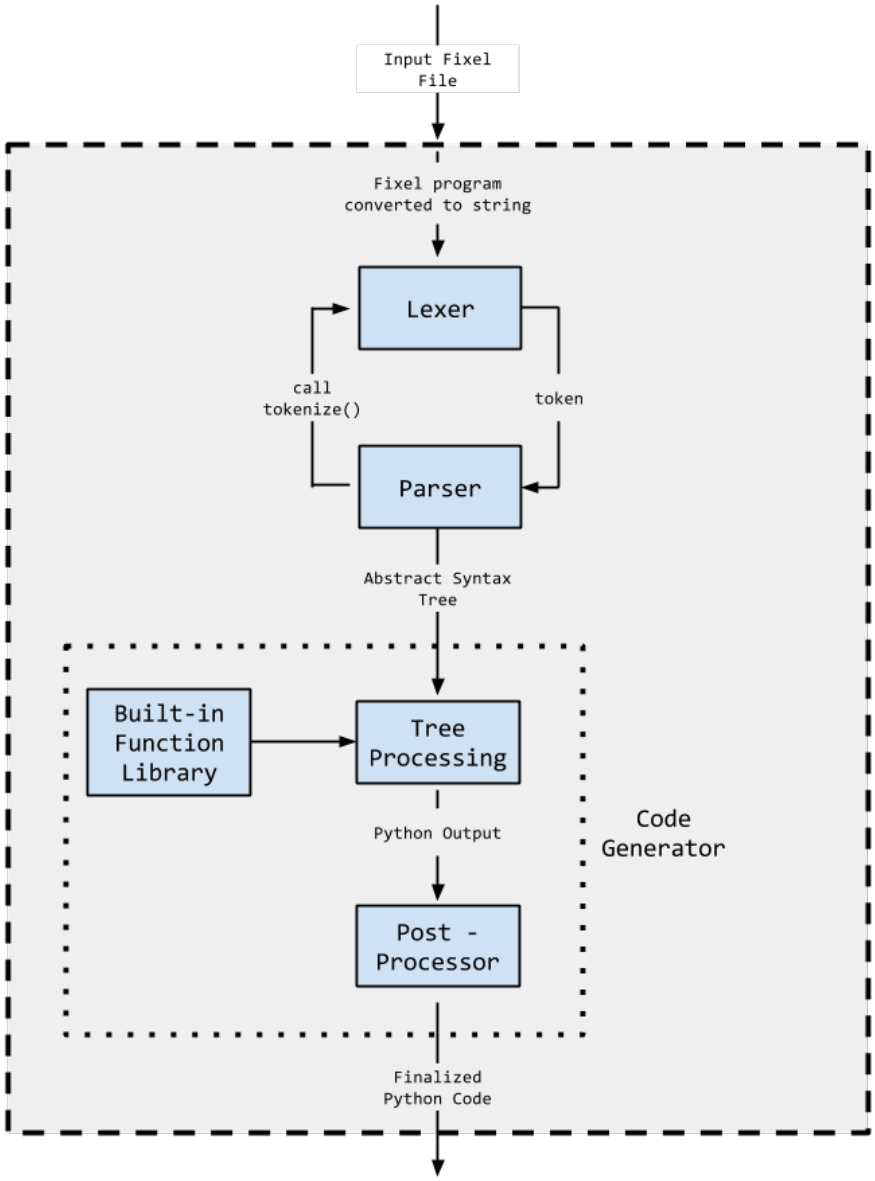

Figure 7: High-level block diagram of translation architecture

IO for images and creating the final Python program. The post-processor was written by our Tester and Validator, Matt, and our System Integrator, Neel.

Built-in Functions Library

Fixel is a language that can be used by consumers right out of the box: it comes with an assortment of functions that can be stacked and customized by the user to begin making transformations on their images. Given a basic understanding for image manipulation, the user can begin creating custom filters and if desired, more advanced programs in Fixel. The built-in functions were written by our System Integator, Neel. For a more comprehensive description of Fixel's built-in functions and their uses, please consult the white paper.

6.4 Integration

All of the different stages of our translator are brought together and run in tandem by a top-level wrapper script, `fixel`. This script was written by our System Integrator, Neel.

7 Development and Runtime Environment

This compiler was developed using both Windows and MacOS. Git hosted on Bitbucket was used for version control as well as for proper source code sharing. We used Python 2.7.4 to write and test our code.

Since python does not require a conventional Makefile, we instead we wrote a bash file in order to run the code. This file takes numerous arguments, the first is the fixel file with the extension .fxl and there are the images to be passed in as fixel arguments. The `fixel` file first creates the translator to create the executable python file. Then the python file is run on the image arguments passed in.

This is run as follows: `fixel program_name.fxl [image_arguments]`

For windows computers it should be run as follows: `fixel.py program_name.fxl [image_arguments]`

Fixel:

```
#!/usr/bin/env python

import importlib
import os
import subprocess
import sys

if len(sys.argv) < 2:
  print("must pass name of program to run followed by images to edit
      ")
  sys.exit(1)

fixel_top = os.path.dirname(__file__)
```

```
ret = subprocess.call(['python', os.path.join(fixel_top, 'src', '
    run_translator.py'), '-f', sys.argv[1]])

if ret != 0:
  sys.exit(ret)

path, name = os.path.split(sys.argv[1])
name = name.split('.')[0]
outname = os.path.join(path, name + '.py')
file_args = sys.argv[2:]
sys.argv = [outname]
sys.argv.extend(file_args)
try:
  importlib.import_module(name)
except AttributeError, e:
  print("make sure you define variables before using them, don't use
       image variables for images you don't pass to the program and
      don't call properties a variable doesn't have.\n" + e.message)
except NameError, e:
  print("make sure you define variables before using them\n" + e.
      message)
except IndexError, e:
  print("make sure you don't try to access elements that aren't in a
       collection\n" + e.message)
except Exception, e:
  print("looks like something went wrong\n" + e.message)
```

8 Test Plan

The majority of the testing is broken up into unit tests that focus on checking the output each part of the translator produces given a variety of inputs. These unit tests are focused on the the lexer, the parser and the generator. Each test creates an instance of the given stage, feeds it an input, and compares the output it produces to an expected output. There are also some tests that run the entire translator on a source program, and check only the final output. This allows for more complex programs to be written, since the intermediate forms for these test cases don't have to be written down. Lastly, all of the built in functions and classes that Fixel programs use at runtime are tested. The built-in functions are too complex to make assertions about, as they manipulate image files, so they are simply run to make sure they don't raise any errors.

8.1 Tools

The Python Coverage utility assisted in writing the test cases to make sure as much of the source code as possible was tested. The coverage tool would analyze the tests and report any lines of source code in the tested files that were not executed. This was particularly useful in determining if every grammar production was tested. `Mock` was used to isolate parts of code that depended on the behaviour of other sections of code to operate properly, and the tool

`Nose` was used to run all of the tests.

8.2 Challenges

One challenge that had to be overcome was how to express the inputs and expected outputs. The first and last forms of a program throughout translation (the Fixel source and Python source) were easily written with multiline strings, but the intermediate forms were a bit harder to write down. A list of tuples of strings made writing token streams fairly easy, but expressing syntax trees was not quite so simple. The method chosen was a string representation that recursively defines a tree node and all of it's subtrees. This format could be both parsed and generated programmatically, and could thus be used to express test cases. It was not very easy to read, however, and as such was impractical to write by hand even for programs that were just a few lines long. To solve this, a simple tool was written for the test suite that creates a tree string from a given Fixel source string. The validity of this output was then confirmed with a tool found at `http://ironcreek.net/phpsyntaxtree` which displays a graphical representation of a tree given in the string syntax.

Another challenge was presented when figuring out how to isolate the parser. The lexer and generator were easily run in isolation as they simply produce an output when given an input. The parser, however, requires a lexer object, from which it obtains an input token stream. The solution used was creating a mock lexer with the Mock tool that presents the same interface as a real lexer, but produces a predetermined token stream.

8.3 Relevant Files

All of the tests files are in the tests directory within the src directory.

test_strings.py

Many program representations are needed by multiple test files (e.g. a token stream will be used as both the expected output of a lexer test and as the input for a parser test), so every string used as input and/or expected output for tests are kept in this file.

fixel_tests.py

Runs the entire translator on various fixel source progpams and compares the output to the expected Python source.

lexer_tests.py

A collection of test cases for the lexer. Each test runs the lexer on a Fixel program and compares the output to the expected token stream.

parser_tests.py

A collection of test cases for the parser. Each test runs the parser on a token stream and compares the output to the expected syntax tree.

generator_tests.py

A collection of test cases for the generator. Each test runs the generator on a syntax tree and compares the output to the expected Python source.

runtime_tests.py

Calls all of the built in functions on a test image to determine if they execute without raising an error. Testing the changes in the images was not practical given the time frame of this project, so no assertions are made about image manipulating functions during this test. Also tests the behavior and operations of the data types Fixel programs use at runtime.

test_utils.py

Contains several methods that assist in writing and running tests, including creating trees from strings and vice versa.

8.4 Continuous Integration Server

Fixel used a Raspberry Pi running the open source continuous integration server Jenkins. Jenkins was used as a second reference for running tests, and to collect statistics about the state of Fixel's tests over time. Whenever someone pushed the master branch to the remote on BitBucket, Jenkins would check out the commit pushed and run the tests. If a test failed, Jenkins would email the whole team informing them that the failure occurred and what caused it. This helped ensure that master stayed clean, and kept everyone informed about when it might be problematic to create a new branch from master. From all of these test runs, Jenkins collected data about how many tests were run, how many failed, and what percent of code is covered, and how this data changed over time.

9 Conclusions

9.1 Lessons Learned

As a team, we learned that it is very important to begin discussing the specifics of the language being developed early on. Therefore, it is very useful to write a sample program on day one when deciding what language to make. Additionally, it is very important to take the little extra time to ensure that everyone is on the same page when big project decisions are being made. This pays off in the

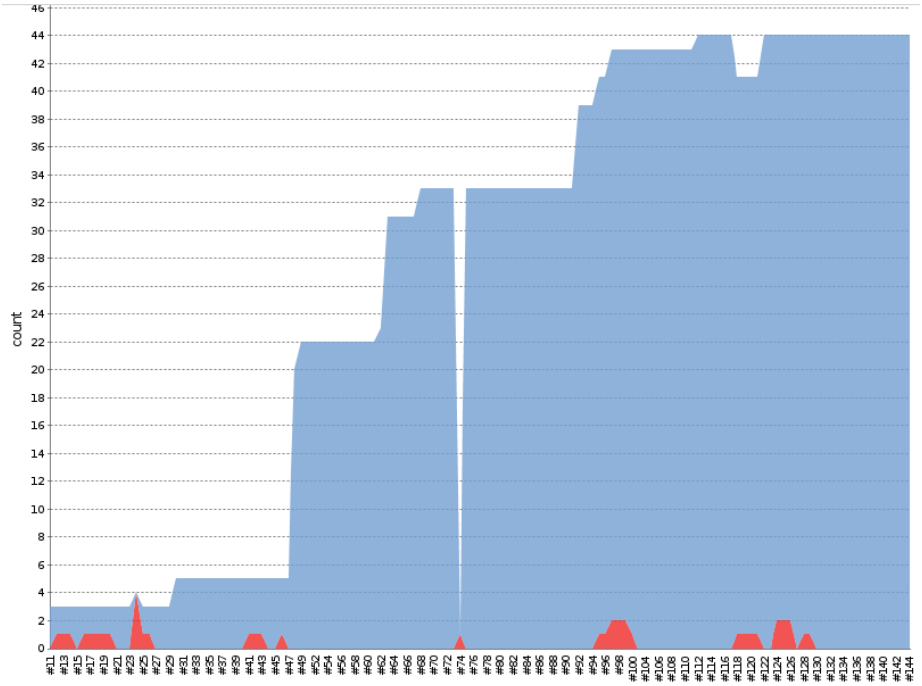

Figure 8: Test Results for each build run by Jenkins.

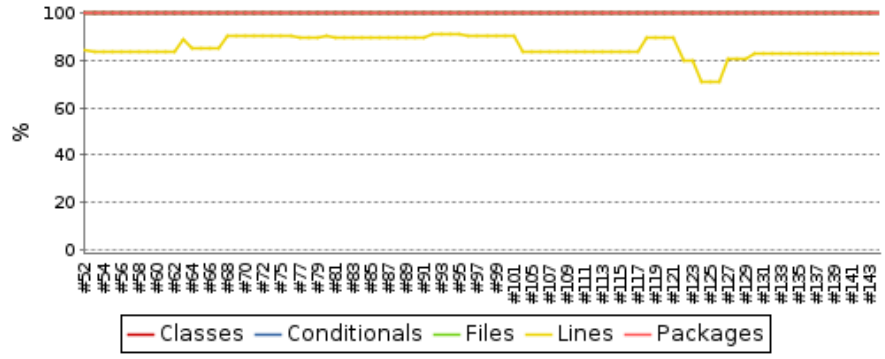

Figure 9: Coverage over time.

future as the project gets more complex and also ensures that nobody spends their time on unnecessary work due to misunderstandings.

Amrita

Writing Fixel not only gave me insight into the art of crafting a programming language, but also a deeper understanding of our target language, Python, as well. In making design decisions about our own language, I learned a great deal about Python, and how its implementation would inform our own language's structure. Working on a translator gave was a great way to really critically understand the implications of the concepts learned during the semester, such as typing, scope, etc.

Writing Fixel would certainly not have been as efficient or pleasant as it was if not for our team group dynamic. Early on, we decided on a weekly meeting schedule to best accommodate our team members' varied schedules. This emphasis on weekly meetings really helped maintain productivity levels throughout the semester.

Darien

I was both impressed and intimidated when I first heard that Computer Engineering students were required to take a course like Programming Languages and Translators. I was under the impression that developing a computer language required an large team of scientist and engineers. However in just a few months, my team and I created a fantastic language for editing images. Even though this project was very technical, I think the skills I learned regarding team building and organization. Tools like Git, Google Docs, and Jenkins helped facilitate this process.

I would advise future teams to start small with their projects. Every team will make changes as the project progresses and making these changes will be difficult if the project is too complicated. Create a minimal viable language with a small "hello world" type program. From there, add any additional functionality to your language may need.

Neel

The idea of building a programming language from the ground up is daunting, but given the right mix of expertises, it can be a pleasant experience. What worked well for us was our shared enthusiasm for the idea of the program—everyone was on board with the goals laid out in our initial meetings, and so we had a pretty comprehensive vision for the language from the beginning. We did, however, struggle with explicitly defining the finer details of our language early on, like issues with syntax. That being said, when we saw issues, we did a good job of communicating the potential problems with the team during our weekly meetings and coming to a consensus on how to deal with them. Everyone seemed to find their niche fairly quickly, and I think we did a good job of distributing assignments that tapped into our team members' strengths.

One piece of advice I'd give to future teams is to really hammer down the tutorial as early as possible. If there is the possibility for change, that should be discussed from the onset, rather than halfway into the process for building the lexer or parser. It's difficult to understate the importance of the tutorial, as the programs described therein will form the foundation of the compiler.

Kavita

Creating fixel provided a novel opportunity to view programming in a new light. While working on this project it became necessary to understand why decisions were made in programming languages and not just memorize the rules. Here are just some of the many things I learned:

1. Make sure to understand everything along the way. A small misunderstanding early on can snowball into a much bigger problem later on.
2. Planning and communication are essential.
3. Weekly team meetings provide consistency and an organizational framework.
4. Never underestimate the importance and usefulness of Git.
5. A great team makes all the difference!

Matt

Developing a language and translator is hard work that requires a lot of time, effort and organization. It is also a highly rewarding experience that brings a better understanding of programming languages and programming languages in general. By developing a parser I gained an intuitive understanding of what Context free grammar's are and how they can be used. Here is some advice about how future teams should go about this project.

1. The earlier the test suite is started the better. Even before we had a working translator to test, thinking about how the translator would be tested helped clarify what needed to be done. Also, having testing working early allowed the whole team to write tests as they work on the translator, which was a lot easier than going back afterwards and trying to figure out how to write tests for a large code base.
2. It was very important to decide on a general work flow as a team. Developing such a complex software project is a challenge in its own right, and adding source control and collaboration to the mix makes it even more complicated. Having the whole team use the same work pattern makes handling merge conflicts and other source control issues manageable.
3. Designing the layout for a project as big as this is not a trivial task. Creating a layout that is intuitive, simple, and works well is quite a challenge. Making the project easy to use and test is also very useful. Being able to

easily run the translator on an input and see all the intermediate steps of translation makes debugging a lot easier.

9.2 Advice for Future Teams

Developing your own language is a very rigorous and rewarding endeavour. Here's some advice to help in the process.

1. Start early.
2. Plan ahead. Stick to your plans.
3. Develop clear milestones to maintain progress.
4. Make sure your goals are practical. Its better to do it right.
5. Use version control. Especially in large groups this is paramount.
6. Maintain open communication with TAs and the professor. Also, referencing examples can help make sure you are on the right track
7. Keep track of decisions made so everyone is on the same page.
8. Make skeleton of your program before delving into more complex functionality. This will make your job easier later on.
9. Have weekly scheduled meetings or other times when the team can meet in person. This really helps with communication.
10. Plan as a group, work individually or in pairs.
11. Have fun!

9.3 Suggestions

The following are a few suggestions for this class based on our experience:

1. While the information learned in class was very helpful for writing the lexer and parser, learning more about translation to high level languages in class would have helped with code generation.
2. It would be helpful to have more structure and feedback for our language along the way. This could take the form of milestones which would help ensure that teams are on the right track with their languages. These could serve as a way to measure progress from a timeline perspective as well as ensure that the language being developed met the necessary requirements.

A Git Log

This log is included to show the evolution of our language. Since we utilized pair proramming for the majority of this project, this commit history is not indicative of work distribution.

Author	Date	Message
Matthew Patey	2014-02-05	First commit, added README.
Matthew Patey	2014-02-05	Added a git ignore file.
Matthew Patey	2014-02-05	Added a setup script. Added content to readme.
Matthew Patey	2014-03-04	added initial test file to test jenkins.
Matthew Patey	2014-03-04	Changed tests.py to pass.
Matthew Patey	2014-03-04	jenkins test.
Kavita Jain-Cocks	2014-03-04	Kavita broke everything.
Matthew Patey	2014-03-04	Yay I fixed it!
Matthew Patey	2014-03-04	jenkins test.
Matthew Patey	2014-03-04	Okay sorry guys, should be the last jenkins test. It's gonna be awesome!
Amrita Mazumdar	2014-03-29	amrita is trying
Amrita Mazumdar	2014-03-29	i did it yay
Nilkanth Patel	2014-03-31	Am I doing this right hope so
Amrita Mazumdar	2014-04-05	OUR LEXER
Nilkanth Patel	2014-04-05	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Nilkanth Patel	2014-04-05	Created new branch neel
Nilkanth Patel	2014-04-05	I think i'm doing this right? these are our sample python programs.
Matthew Patey	2014-04-06	Added grammar file to repo. Might be useful (at least at first) to have on git.
Matthew Patey	2014-04-06	Added field and list access, dealt with functions (didn't end up forbidding inlining).
Amrita Mazumdar	2014-04-06	more to lexer
Amrita Mazumdar	2014-04-06	sorted out tabs
Amrita Mazumdar	2014-04-06	lexer working properly on fixel input
Matthew Patey	2014-04-06	made parser.py file.
Matthew Patey	2014-04-06	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-04-06	Fixed grammar, started parser.
Darien	2014-04-06	Darien did some parsing junk: function-def, parameter-dec, statement-list, statement
Matthew Patey	2014-04-07	wrote part of parser.

Matthew Patey	2014-04-07	Merge branch ‘master’ of bitbucket.org:mpatey/fixel
Matthew Patey	2014-04-07	Added ply to the src directory.
Matthew Patey	2014-04-07	By golly it’s working! Also made another fix for grammar.
Matthew Patey	2014-04-07	modified git ignore and added copy of lexer modified to work with yacc.
Matthew Patey	2014-04-07	tried out character literals
Matthew Patey	2014-04-07	added more newline flexibility.
Amrita Mazumdar	2014-04-07	added literals
Matthew Patey	2014-04-09	First pass at code generation, works for 1 liner, but leaves in # and @.
Matthew Patey	2014-04-09	Maybe dealt with epsilon.
Matthew Patey	2014-04-09	turns out imports always break when running a script in a package, so made one ooutside of the package.
Matthew Patey	2014-04-10	Removed leaf type, now all tree nodes are of type Node. To make a leaf, simply don’t pass any children arguments.
Matthew Patey	2014-04-10	Merge branch ‘master’ into generator
Matthew Patey	2014-04-10	Fixes to generator now that Leaf is gone.
Nilkanth Patel	2014-04-10	added new buil-in functions
Nilkanth Patel	2014-04-10	small change
Nilkanth Patel	2014-04-10	Merge branch ‘neel’
Matthew Patey	2014-04-11	fixed typos in grammar.
Amrita Mazumdar	2014-04-14	day 1 darien and amrita parser additions
Amrita Mazumdar	2014-04-14	parser finished and working, yay
Nilkanth Patel	2014-04-14	Merge branch ‘master’ of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-04-14	Merge branch ‘generator’
Matthew Patey	2014-04-09	I mocked lex and got yacc to build a tree with it!
Matthew Patey	2014-04-10	Worked on nodes, needs to be reworked. Wrote method to obtain string from tree.

Matthew Patey	2014-04-10	Created test_utils module that contains functions to be used across test modules and debugging functions. Also created simple script to run it with an example. Wrote code to create tree from string representation, prints it out with tree's repr function. Desired use is to see tree-strings in human friendly format.
Matthew Patey	2014-04-10	wrote script so that any string in test_strings can be run through the pretty formatter. use -h for usage.
Matthew Patey	2014-04-11	Reorganized project so that annoying imports work. Requires having top level directory ("code") that IS NOT a package, which contains any scripts that will be run as well as all packages.
Matthew Patey	2014-04-11	Finished parser and lexer test framework.
Matthew Patey	2014-04-13	Added function to print tree string for AST built from given source code.
Matthew Patey	2014-04-14	Wrote test to fail, testing out jenkins with the awesome new testing suite.
Matthew Patey	2014-04-14	Fixed test.
Amrita Mazumdar	2014-04-14	fixed epsilon
Nilkanth Patel	2014-04-14	linespace
Nilkanth Patel	2014-04-14	Merge branch 'neel' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-04-14	testing testy test.
Matthew Patey	2014-04-14	removed test.txt.
Matthew Patey	2014-04-14	test!
Matthew Patey	2014-04-14	stuff
Matthew Patey	2014-04-15	Fixed the oneliner test for the parser.
Amrita Mazumdar	2014-04-16	commented generator for clarity
Nilkanth Patel	2014-04-16	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Nilkanth Patel	2014-04-16	small changes to py functions
Nilkanth Patel	2014-04-16	Merge branch 'neel' of https://bitbucket.org/mpatey/fixel into neel
Nilkanth Patel	2014-04-16	lexer updates for indent dedent
Amrita Mazumdar	2014-04-16	basic tex framework
Amrita Mazumdar	2014-04-16	markdown files for sections completed added

Amrita Mazumdar	2014-04-16	gitignore for tex files
Amrita Mazumdar	2014-04-16	added pdf to gitignore
Amrita Mazumdar	2014-04-16	base runscript for creating paper
Amrita Mazumdar	2014-04-16	script now compiles md to tex and all tex to pdf together
Amrita Mazumdar	2014-04-16	added cleanup to script
Amrita Mazumdar	2014-04-16	added some info to lrm but this will be a more involved section
Amrita Mazumdar	2014-04-16	added some source code to appendix
Amrita Mazumdar	2014-04-16	Merge branch 'master' of https://bitbucket.org/mpatey/fixel into paper
Nilkanth Patel	2014-04-16	INDENT DEDENT WORKS MOSTLY
Matthew Patey	2014-04-16	Revert "small changes to py functions"
Nilkanth Patel	2014-04-16	fix to get the lexer working again.
Nilkanth Patel	2014-04-16	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Nilkanth Patel	2014-04-16	Merge branch 'master' into neel
Nilkanth Patel	2014-04-16	lexer is working as it should with indent/dedent!
Nilkanth Patel	2014-04-16	small fix for lexer decent thing
Matthew Patey	2014-04-17	Added multiline support to lex, allowing ^ to be used to match beginning of line. Gave example of use in indent rule.
Matthew Patey	2014-04-17	Added test util that runs the lexer only. Works similar to tree verification where name of attribute in test_strings is supplied (e.g. use oneliner_source to run lexer on that string). Output is tokens.
Matthew Patey	2014-04-17	Changed fixel name op main function to translate, which now takes source string as argument and returns a python string.
Kavita Jain-Cocks	2014-04-17	lexer no longer consumes first character at start of line for no dedents

Darien Nurse	2014-04-17	Changed parenthesis to brackets in test_utils. Changed oneliner_tree in test_strings to use brackets instead of parenthesis. Made a test case for comparing input fixel code to target python code. Added a tree making website to the README
Matthew Patey	2014-04-17	Made tree strings not be ws dependent. Deleted sloppy code at end of test_utils. Added generator tests module.
Matthew Patey	2014-04-17	Merge branch 'tests' of bitbucket.org:mpatey/fixel into tests
Matthew Patey	2014-04-17	Fixed last change from (to [for tree strings.
Kavita Jain-Cocks	2014-04-17	remove testing from lexer
Kavita Jain-Cocks	2014-04-18	Merge branch 'master' into neel
Kavita Jain-Cocks	2014-04-18	allows newline when dedent starts at edge of file
Matthew Patey	2014-04-18	Added custom functions for nodes to generator. This allows function expressions to generate empty '(' ')' if it has no parameters.
Matthew Patey	2014-04-18	quick fix, wrote epsilon rule for parameter declaration.
Matthew Patey	2014-04-18	Just kidding! turns out epsilon is not needed in parameter_declaration because function_definition has a production with no parameter_declaration. Updated grammar.txt aslo.
Matthew Patey	2014-04-19	Added -v option to run_fixel script that prints each intermediate step of translation.
Matthew Patey	2014-04-19	Revert "Added test util that runs the lexer only. Works similar to tree verification where name of attribute in test_strings is supplied (e.g. use oneliner_source to run lexer on that string). Output is tokens."

Matthew Patey	2014-04-20	Added index to custom behavior in generator so it can occur after any child. Added custom behavior for function definition. Removed EOF token that I added awhile ago, but never figured out how to use it. Stopped lexer from changing the value of number tokens to int (but still checks if the conversion succeeds). Changed lots of rules in parser; parser is a representation of fixel, so I think it's best to leave all tokens in and let generator decide what to omit (e.g. '#'), plus all tokens must be turned into new nodes. Changed oneliner tests to have newline at end.
Matthew Patey	2014-04-20	Reworked generator to use strings table and custom behavior table (both are dictionary of dictionaries). Added comma to literals in lexer. Changed ID to function so that it can check for reserved words. Indent and dedent tokens are now accompanied by identical values. Changed function_definition and parameter_declaration in parser (and grammar.txt) so that parameter_declaration goes to epsilon.
Matthew Patey	2014-04-20	Lexer now adds all missing DEDENT's at EOF, and discards blank lines. As result of eof behavior, files with missing at end of line will not be recognized. Changed tree printing to use repr so we can see whitespace.
Matthew Patey	2014-04-20	Removed auxIndent from Lexer. If we decide we want variable indent widths, we need to use a stack. Added windows newlines to all regex's matching . Added rule to skip blank lines.

Matthew Patey	2014-04-20	Fixed error in some regex (inside [] special characters lose there meaning... my bad!).
Matthew Patey	2014-04-21	Reworked generator again in preparation for INDENT/DEDENT, now each node either has custom function to process it or uses default that includes an insert spaces lookup. Our grammar was ambiguous whoops! LR parser can't tell difference between function call and function definition. Implemented a fix by removing # from declaratoin (we can change to something else). Changed test_utils to print repr of tree_strings, so whitespace is printed as string representation (e.g. show instead of an actual newline).
Amrita Mazumdar	2014-04-21	Merge branch 'master' of https://bitbucket.org/mpatey/fixel into paper
Amrita Mazumdar	2014-04-21	Added a README for clarity
Amrita Mazumdar	2014-04-21	added new rules for intermediate expression to parser
Matthew Patey	2014-04-21	Updated grammar text file.
Matthew Patey	2014-04-21	Merge branch 'master' of bitbucket.org:mpatey/fixel
Nilkanth Patel	2014-04-21	made a sample python function to show professor aho on wednesday. generates dummy images and uses new method for function calling.
Nilkanth Patel	2014-04-21	Merge branch 'neel' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-04-21	Fixed tests to use new intermediate expression grammar symbol.
Matthew Patey	2014-04-21	Merge branch 'master' of bitbucket.org:mpatey/fixel

Matthew Patey	2014-04-22	In lexer changed leadingWhitespace_leftmostDEDENT match beginning of line instead of consuming a newline, to avoid conflict with blank line rule. Changed for loops (and list access for consistency) to use primary instead of intermediate expressions because our function calls don't nest well (ambiguous commas). Note we support an expression surrounded by parentheses, which is considered a primary expression. Fixed assignment expression to only assign to variables, not ID's.
Matthew Patey	2014-04-25	test
Matthew Patey	2014-04-25	Revert "test"
Matthew Patey	2014-04-26	Added util that generates list syntax token stream from source, similar to tree generation from source. Added '.' to lexer literals. Added lexer indent test.
Kavita Jain-Cocks	2014-04-26	added some tests source strings and fixed for in parser and grammar
Matthew Patey	2014-04-26	Fixed whitespace in grammar.txt. Added "function_def" "or_and" and "not" test source.
Matthew Patey	2014-04-26	Merge branch 'tests' of bitbucket.org:mpatey/fixel into tests
Matthew Patey	2014-04-26	Fix to parser, got all lexer tests passing.
Matthew Patey	2014-04-26	Parser: fixed primary expression to token productions. Got all the parser tests working. Added fix to tree strings to handle brackets as node names.
Matthew Patey	2014-04-26	Added coverage output file to ignore.
Matthew Patey	2014-04-26	Updated tests to improve coverage based on coverage results. Fixed two tests that were not being run because of names.
Amrita Mazumdar	2014-04-28	indent and dedent added to code generation, and newline procesing, yay
Neel	2014-04-28	small changes to fix final sample python script

Matthew Patey	2014-04-30	changed generator dedent from slicing to del to avoid copying list.
Amrita Mazumdar	2014-04-30	added pix 2 paper, language evolution section, etc
Amrita Mazumdar	2014-04-30	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Amrita Mazumdar	2014-04-30	working on integrating images to paper
Amrita Mazumdar	2014-04-30	update gitignore for macs
Kavita Jain-Cocks	2014-04-30	added project plan - still unfinished, need testing
Kavita Jain-Cocks	2014-04-30	Merge branch 'tests' of bitbucket.org:mpatey/fixel into tests
Kavita Jain-Cocks	2014-04-30	fixed some spacing stuff for project plan
Kavita Jain-Cocks	2014-04-30	fixed merge conflicts
Amrita Mazumdar	2014-05-01	adding this awkward section with silly picture
Amrita Mazumdar	2014-05-01	added code generator tests
Amrita Mazumdar	2014-05-01	fixed tab spaces for one of the tests that was weird
Kavita Jain-Cocks	2014-05-01	added first half of generator tests
Kavita Jain-Cocks	2014-05-01	fix merge conflicts
Matthew Patey	2014-05-01	added better error message for trees that don't match in tests.
Matthew Patey	2014-05-01	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-01	Added parser test for indent.
Matthew Patey	2014-05-02	Removed duplicate function in generator tests.
Matthew Patey	2014-05-02	Merge branch 'tests'
Matthew Patey	2014-05-02	Added in generator test for indent.
Darien Nurse	2014-05-02	Added File I/O
Matthew Patey	2014-05-02	Added post process step that adds additional python necessary to run files. Added generator code that appends necessary things to certain expressions. Made generator indent count an instance variable.
Matthew Patey	2014-05-02	Merge branch 'generator' of bitbucket.org:mpatey/fixel into generator

Matthew Patey	2014-05-02	Fixed bugs I introduced to generator. Added more necessary python to post_process. Commented out full fixel test. Will need to address new post-processing we do. Updated generator tests to handle two strings returned by generator. Updated python tests strings to have "ns." and "fixelFunctions." where appropriate, and to have main after def's. Made py a package so we can import from it. Added a test fixel function. If I remember correctly we decided on .fxl, but if I got that wrong we can change it.
Neel	2014-05-04	added support for saving color values
Amrita Mazumdar	2014-05-04	ADDED ASCII ART MOST IMPT TASK
Neel	2014-05-04	added test string to file (still missing expected tree output), added support for grabbing pixel data from image
Neel	2014-05-04	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-05-05	updated post process to reflect changes Neel made. Added complex full fixel test. Added diff printout to fixel tests. Changed python test items to be tuple of main and rest of program to allow better testing.
Amrita Mazumdar	2014-05-05	added font
Amrita Mazumdar	2014-05-05	fixed font path
Amrita Mazumdar	2014-05-05	added conclusiosn to paper
Amrita Mazumdar	2014-05-05	filled in rest of concl skeleton
Matthew Patey	2014-05-05	Wrote script that translates fixel file passed as first argument and then calls it with the rest of the arguments.
Matthew Patey	2014-05-05	Merge branch 'master' of bitbucket.org:mpatey/fixel

Matthew Patey	2014-05-06	Changed test script to call tests on correct directory regardless of where it is called from. Changed post process to add fixel to program path so imports will work regardless of where output python is run from. caption bif now uses path to file when opening font file so it can be opened regardless of current directory. Wrote new top level python script that can be invoked directly (if python executable is in your path). By adding the top level fixel directory to your path, the script can be invoked from anywhere.
Amrita Mazumdar	2014-05-06	fixed file mode issue for grayscale
Matthew Patey	2014-05-06	changed file output to use name of input file.
Kavita Jain-Cocks	2014-05-06	lessons learned for Kavita
Kavita Jain-Cocks	2014-05-06	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-06	Renamed some stuff so that it's a little less confusing.
Amrita Mazumdar	2014-05-06	executable permissions on fixel
Matthew Patey	2014-05-06	created image runtime class.
Amrita Mazumdar	2014-05-06	Merge branch 'runtime_image' of https://bitbucket.org/mpatey/fixel into runtime_image
Matthew Patey	2014-05-06	small changes to fixel functions. Implemented bracket pixel access, lazy pixel data loading for image class.
Amrita Mazumdar	2014-05-06	rework fixelFunctions for new runtime image class
Matthew Patey	2014-05-06	Merge branch 'runtime_image' of bitbucket.org:mpatey/fixel into runtime_image
Matthew Patey	2014-05-06	small fix
Amrita Mazumdar	2014-05-06	added pixel access to grammar
Darien Nurse	2014-05-06	Changed image indexes to 0
Kavita Jain-Cocks	2014-05-06	fixed true false so now True False in python
Kavita Jain-Cocks	2014-05-06	Merge branch 'master' of bitbucket.org:mpatey/fixel
Darien Nurse	2014-05-06	Blocked Python keyword from Fixel

Darien Nurse	2014-05-06	Merge branch 'master' of bitbucket-accountB:mpatey/fixel
Matthew Patey	2014-05-06	Added pixel access to language. Uncommented strings in post process. Added pixel access tests. Updated not test strings to reflect changes to grammar.
Matthew Patey	2014-05-06	Merge commit '90f2d2f33a50f0d5c677a2fa6ac8232da333aad3' Bring directory changes and updated fixel script into master.
Kavita Jain-Cocks	2014-05-06	added advice for future teams to paper
Kavita Jain-Cocks	2014-05-06	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-06	run translator changes working directory during translation so that parsing output files always go to the same place. changes back to original before writing out files.
Matthew Patey	2014-05-06	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-06	added color to bif list in generator. Added better error messages to parser. Changed fixel functions to call set image data function on image objects, allowing them to null out their pixel data.
Amrita Mazumdar	2014-05-08	Merge branch 'runtime_image' of https://bitbucket.org/mpatey/fixel into runtime_image
Amrita Mazumdar	2014-05-08	amritas lessons are ok
Matthew Patey	2014-05-08	Added some Matt paper stuff.
Amrita Mazumdar	2014-05-08	more updates to grammar from bracket overloading
Amrita Mazumdar	2014-05-08	adding variable to grammar
Matthew Patey	2014-05-08	fixed tabby spacey thing.
Amrita Mazumdar	2014-05-08	fixed some errors in new variable parser
Amrita Mazumdar	2014-05-08	fixed typo
Matthew Patey	2014-05-08	Fixed parser tests to handle new grammar.
Matthew Patey	2014-05-08	Merge branch 'runtime_image' of bitbucket.org:mpatey/fixel into runtime_image

Neel	2014-05-08	fixed color function to accept text or rgb values
Neel	2014-05-08	added neel's stuff for the paper
Neel	2014-05-08	added collage function to fixelFunctions
Neel	2014-05-08	added collage function to fixelFunctions
Matthew Patey	2014-05-09	First pass at forp loop. Functions for 1 loop, does not currently support nested loops.
Matthew Patey	2014-05-09	Got generator to work for nested loops, still needs improvements. Wrote forp tests.
Kavita Jain-Cocks	2014-05-09	removed vin diesel
Amrita Mazumdar	2014-05-09	updated paths to more code
Amrita Mazumdar	2014-05-09	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Darien Nurse	2014-05-09	added lessons learned for Darien
Darien Nurse	2014-05-09	Merge branch 'master' of bitbucket-accountB:mpatey/fixel
Darien Nurse	2014-05-09	Remove typo in lessons learned
Matthew Patey	2014-05-09	Merge branch 'runtime_image'
Matthew Patey	2014-05-09	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-09	turns out leaving out the "(object)" in class declaration isn't shorthand, it creates old style objects (whatever those are).
Matthew Patey	2014-05-09	Added comments to lexer, added a few forbidden words to lexer, made lexer tests error messages better.
Matthew Patey	2014-05-09	Reworked some fixel function stuff, Added a lot to runtime classes, mostly color class.
Matthew Patey	2014-05-09	Fixed file name splitting in run_translator and fixel scripts. Fixed forp bug.
Matthew Patey	2014-05-09	Moved runtime into code so it can be tested. Added runtime tests, fixed up fixel functions. Removed blank line grammar rule.
Darien Nurse	2014-05-10	Handled attribute errors in the Generator. Removed eofDedent and dedentCount warnings from lexer.
Matthew Patey	2014-05-10	Fix to file name in runtime tests.

Matthew Patey	2014-05-10	Another big reorganization. Renamed imports and directory names as needed.
Matthew Patey	2014-05-10	Removed comment token to get rid of warning.
Kavita Jain-Cocks	2014-05-10	added some development and runtime
Darien Nurse	2014-05-10	Improved error handling
Darien Nurse	2014-05-10	Merge branch 'master' of bitbucket-accountB:mpatey/fixel
Neel	2014-05-10	lessons learned for neel
Neel	2014-05-10	bunch of stuff for the tutorial
Matthew Patey	2014-05-10	Paper. Some small changes in lang_evol. Added test_plan section.
Matthew Patey	2014-05-10	Merge branch 'master' of bitbucket.org:mpatey/fixel
Kavita Jain-Cocks	2014-05-10	commented lexer and added development environment to paper
Kavita Jain-Cocks	2014-05-10	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-10	Updated grammar. Added LRM to paper, and added lrm and test plan sections to paper.tex.
Matthew Patey	2014-05-10	Added small fix to parameter declaration rule in parser, updated tests.
Matthew Patey	2014-05-10	Merge branch 'master' of bitbucket.org:mpatey/fixel
Kavita Jain-Cocks	2014-05-10	added suggestions to conclusions
Kavita Jain-Cocks	2014-05-10	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-11	Runtime. Improved how pixel objects manage color, added r,g,b fields to pixel objects.
Matthew Patey	2014-05-11	Merge branch 'master' of bitbucket.org:mpatey/fixel
Neel	2014-05-11	finished tutorial
Neel	2014-05-11	whitepaper fixes, improved collage function
Neel	2014-05-11	some error handling for python, not pushed to run_time yet, though, still working in py. will move to run_time once i'm done
Amrita Mazumdar	2014-05-11	fixed code fragment formatting
Amrita Mazumdar	2014-05-11	encoding issues with smart quotes
Amrita Mazumdar	2014-05-11	encoding issues with smart quotes, image linking, weird unicode stuff

Amrita Mazumdar	2014-05-11	added code listings, fixed some broken paths
Kavita Jain-Cocks	2014-05-11	added some comments to generator
Kavita Jain-Cocks	2014-05-11	Merge branch 'master' of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-11	fixed some changes i made to the backtick/apostrophes
Kavita Jain-Cocks	2014-05-11	added some generator comments
Kavita Jain-Cocks	2014-05-11	Merge branch 'master' of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-11	finally got git commit history auto-adding into the paper nicely
Amrita Mazumdar	2014-05-11	some encoding fixes
Neel	2014-05-11	more error-handling for python stuff, pushed to src
Neel	2014-05-11	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Amrita Mazumdar	2014-05-11	comments to parser
Amrita Mazumdar	2014-05-11	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Neel	2014-05-12	formatting fixes for the tutorial
Amrita Mazumdar	2014-05-12	got picture resizing working
Matthew Patey	2014-05-12	Fixed bugs.
Matthew Patey	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Kavita Jain-Cocks	2014-05-12	fixed code in development and runtime environment
Kavita Jain-Cocks	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Neel	2014-05-12	python error handling now uses isinstance()
Neel	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-05-12	Added some conclusion stuff.
Matthew Patey	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Kavita Jain-Cocks	2014-05-12	added testing section to project plan
Kavita Jain-Cocks	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-12	Fixed build errors.
Matthew Patey	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-12	added tables
Matthew Patey	2014-05-12	Updated collage to use fixel image objects, added collage to test, Updated generator bif list.

Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-05-12	Added second image needed for tests.
Kavita Jain-Cocks	2014-05-12	commented generator
Kavita Jain-Cocks	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-12	Added top level error checking in fixel script.
Matthew Patey	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-12	Fixed ./ error for file names.
Kavita Jain-Cocks	2014-05-12	proofreading
Amrita Mazumdar	2014-05-12	added more text to trans arch section
Kavita Jain-Cocks	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Amrita Mazumdar	2014-05-12	added translation architecture
Kavita Jain-Cocks	2014-05-12	in prject plan – proofreading
Kavita Jain-Cocks	2014-05-12	Merge branch 'master' of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-12	proofreading
Kavita Jain-Cocks	2014-05-12	in lang evol for proofreading
Amrita Mazumdar	2014-05-12	gitlog shows in reverse order
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Kavita Jain-Cocks	2014-05-12	finished lang_evol for proofreading
Kavita Jain-Cocks	2014-05-12	finished devel and runtime proofreading
Amrita Mazumdar	2014-05-12	proofing project plan
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Kavita Jain-Cocks	2014-05-12	proofread test plan
Amrita Mazumdar	2014-05-12	proofing lang evol
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Kavita Jain-Cocks	2014-05-12	added lessons learned as a team
Amrita Mazumdar	2014-05-12	proof test plan
Amrita Mazumdar	2014-05-12	Merge branch 'master' of https://bitbucket.org/mpatey/fixel
Matthew Patey	2014-05-12	Updated tutorial, fixed error checking.

Matthew Patey	2014-05-12	Merge branch ‘master’ of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-12	Merge branch ‘master’ of https://bitbucket.org/mpatey/fixel
Amrita Mazumdar	2014-05-12	proof conclusions
Kavita Jain-Cocks	2014-05-12	fixing tutorial
Kavita Jain-Cocks	2014-05-12	Merge branch ‘master’ of bitbucket.org:mpatey/fixel
Matthew Patey	2014-05-12	Added pics for testing.
Matthew Patey	2014-05-12	Merge branch ‘master’ of bitbucket.org:mpatey/fixel
Amrita Mazumdar	2014-05-12	fixed a picture link

B Source Code

B.1 Scripts

```
#!/usr/bin/env python

import importlib
import os
import subprocess
import sys

if len(sys.argv) < 2:
  print("must pass name of program to run followed by images to edit
      ")
  sys.exit(1)

fixel_top = os.path.dirname(__file__)
ret = subprocess.call(['python', os.path.join(fixel_top, 'src', '
    run_translator.py'), '-f', sys.argv[1]])

if ret != 0:
  sys.exit(ret)

path, name = os.path.split(sys.argv[1])
name = name.split('.')[0]
outname = os.path.join(path, name + '.py')
file_args = sys.argv[2:]
sys.argv = [outname]
sys.argv.extend(file_args)
try:
  importlib.import_module(name)
except AttributeError, e:
  print("make sure you define variables before using them, don't use
       image variables for images you don't pass to the program and
      don't call properties a variable doesn't have.\n" + e.message)
except NameError, e:
```

```
    print("make sure you define variables before using them\n" + e.
        message)
  except IndexError, e:
    print("make sure you don't try to access elements that aren't in a
         collection\n" + e.message)
  except Exception, e:
    print("looks like something went wrong\n" + e.message)
```

```
  import os
  import nose
  import sys

  path_to_tests = os.path.join(os.path.dirname(__file__), 'tests')  #
      obtain path to tests relative to this script
  argv = list(sys.argv)
  argv.insert(1, path_to_tests)
  argv.insert(1, '-w')
  nose.main(argv=argv)
```

```
  import argparse
  import os
  import sys
  from translator import translator

  arg_parser = argparse.ArgumentParser()
  arg_parser.add_argument('-v', '--verbose', help='prints every
      intermediate step in translation',
              action='store_true', default=False)
  arg_parser.add_argument('-f', '--fixel-file', type=argparse.FileType
      ('r'))
  namespace = arg_parser.parse_args()

  if namespace.fixel_file:
    path, name = os.path.split(namespace.fixel_file.name)

    source_string = namespace.fixel_file.read()
    namespace.fixel_file.close()

    pwd = os.getcwd()
    script_dir = os.path.dirname(__file__)  # directory this script is
        in
    os.chdir(script_dir)  # change directory so parse outputs are in
        right place
    try:
      result = translator.translate(source_string, namespace.verbose)
      os.chdir(pwd)  # go back to original pwd before writing out file
      outname = os.path.join(path, name.split('.')[0] + '.py')
      f = open(outname, 'w')
      f.write(result)
      f.close()
    except translator.parser.ParsingError as e:
      print e.value
      sys.exit(1)

  else:
    try:
```

```
    result = translator.translate(verbose=namespace.verbose)
  except translator.parser.ParsingError as e:
    print e.value
    sys.exit(1)
```

```
import argparse
from tests import test_strings, test_utils


if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser()
    arg_parser.add_argument('-p', '--pretty-string', help='finds the
         attribute of the test_strings module with the '
                            'given name and prints a pretty version
                                of it if it is a tree string')
    arg_parser.add_argument('-t', '--tree-string', help='finds the
        attribute of the test_strings module with the '
                            'given name and prints a tree string
                                built from it if it is source code')
    arg_parser.add_argument('-s', '--token-stream', help='finds the
        attribute of the test_strings module with the '
                            'given name and prints a list of tokens
                                generated from it if it is source
                                code')
    namespace = arg_parser.parse_args()

    if namespace.pretty_string is not None:
        tree_string = getattr(test_strings, namespace.pretty_string)
        test_utils.tree_string_pretty_version(tree_string)

    if namespace.tree_string is not None:
        source = getattr(test_strings, namespace.tree_string)
        test_utils.tree_string_from_source(source)

    if namespace.token_stream is not None:
        source = getattr(test_strings, namespace.token_stream)
        test_utils.token_stream_from_source(source)
```

B.2 Lexer

```
import ply.lex as lex
import re

'''
reserved words that translate straight to tokens
correspond to the reserved words in fixel
'''
reserved = {
  'for'    : 'FOR',
  'while'  : 'WHILE',
  'if'     : 'IF',
  'else'   : 'ELSE',
  'return' : 'RETURN',
  'in'     : 'IN',
```

```
  'true'  : 'TRUE',
  'false' : 'FALSE',
  'and'   : 'AND',
  'or'    : 'OR',
  'not'   : 'NOT',
  'forp'  : 'FORP',
}

'''
reserved words that throw an exception
these words are forbidden since they are reserved words in python
'''
forbidden = ['del','from','as','elif', 'global', 'with', 'assert', '
    pass', 'True', 'False'
'yield', 'break', 'except', 'import', 'print', 'class', 'exec', '
    raise', 'continue', 'finally', 'is', 'def', 'lambda', 'try']



'''
declare new state for dedent analysis and the various cases
'''
states = (
  ('dedentCount', 'exclusive'),
    ('leadingWhitespace', 'inclusive'),
  ('eofDedent', 'exclusive')
)

'''
correspond to the literals in fixel
'''
literals = ['#', '@', '+', '-', '*', '/', '=', '(', ')',
            '[', ']', ':', ';', '<', '>', ',', '.']

'''
used to help maintain state for indent and dedent calculations
necessary to determine state transitions
'''
globalIndent = 0
currentIndent = 0

'''
a list of all tokens produced by the lexer
will be matched to various patterns
'''
tokens = [
    'ID','NUMBER',
    'LESSTHANEQ',
    'GREATERTHANEQ', 'NEQUAL',
    'STRING','NEWLINE','INDENT', 'DUBEQUAL', 'DEDENT'
    ] + list(reserved.values())

# Tokens produced by simple regexes
t_STRING      = r'\"([^"])*\"'
t_DUBEQUAL    = r'=='
t_LESSTHANEQ  = r'<='
t_GREATERTHANEQ = r'>='
```

```
t_NEQUAL      = r'!='
t_NEWLINE     = r'\r?\n'

# drop blank lines
def t_blankline(t):
  r'^[ \t]*(//.*)?\r?\n'
  pass

#recognize comments
def t_comment(t):
  r'//.*'
  pass

#match variable ids, stores the whole name
def t_ID(t):
  r'[a-zA-Z_][a-zA-Z0-9_]*'
  if t.value in reserved:
    t.type = reserved[t.value]
  elif t.value in forbidden:
    raise NameError(t.value)
  return t

# produces a token for any integer number and stores the value as an
    int
def t_NUMBER(t):
    r'\d+'
    try:
        int(t.value)
    except ValueError:
        print("Integer value too large %d", t.value)
        t.value = 0
    return t

# Ignored characters
t_ignore = " "

t_dedentCount_ignore = " "

t_eofDedent_ignore = " "

# count number of indents
def t_INDENT(t):
  r'^[\t]+(?=[^ \t\r\n])'
  global globalIndent
  global currentIndent
  currentIndent = t.value.count("\t")
  if currentIndent < globalIndent:
    if (globalIndent-currentIndent) > 1:
      t.lexer.begin('dedentCount')
    else:
      t.value = t.type = "DEDENT"
      globalIndent = currentIndent
      return t
  elif currentIndent > globalIndent:
    t.value = t.type = "INDENT"
    if globalIndent == 0:
      t.lexer.begin('leadingWhitespace')
```

```
    globalIndent += 1
    return t

'''
match anything without consuming, keep returning DEDENT's until we'
    ve balanced
accounts for the fact that it may be necessary to return multiple
    dedents
'''
def t_dedentCount_empty(t):
  r'(?=.|\n)'
  global globalIndent
  global currentIndent
  if (globalIndent > currentIndent):
    t.value = t.type = "DEDENT"
    globalIndent -= 1
    return t
  else:
    if globalIndent > 0:
      t.lexer.begin('leadingWhitespace')
    else:
      t.lexer.begin('INITIAL')


'''
enteres a new state to allow the return of necessary dedents at the
    end of the file
returns newline
'''
def t_leadingWhitespace_EOF(t):
  r'\r?\n\Z'
  t.lexer.lexpos -= 1  # rollback lexpos so there's something to
      match for subsequent DEDENT's
  t.lexer.begin('eofDedent')
  t.type = 'NEWLINE'
  return t

'''
accounts for the case when the line is dedented all the way to the
    left
'''
def t_leadingWhitespace_leftmostDEDENT(t):
  r'^(?=[^ \t\r\n])'
  global currentIndent
  currentIndent = 0
  t.lexer.begin('dedentCount')

'''
returns dedent at the end of the file until balanced
'''
def t_eofDedent_DEDENT(t):
  r'(?=.|\n)'
  global globalIndent
  if globalIndent > 0:
    globalIndent -= 1
    t.type = t.value = 'DEDENT'
    return t
```

```
  else:
    t.lexer.lexpos += 1  # undo rollback so we don't consume last \n
         twice


'''
deals with illegal character
'''
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

def t_dedentCount_error(t):
    pass

def t_eofDedent_error(t):
    pass

# Build the lexer

class VerboseLexer(object):
    def __init__(self, lex):
        self.lex = lex

    def token(self):
        """
        function that is dynamically added as instance method of an
            instance of the VerboseLexer class that is
        created at runtime if a verbose lexer is created. Returns
            one token from the lexer,
        but first prints it if it is not None.
        """
        token = self.lex.token()
        if token is not None:
            print(token)
        return token

    def input(self, data):
        self.lex.input(data)


def get_lex(verbose=False):
    """
    If verbose is false or omitted returns an instance of ply.lex
        created with lex rules in this module.
    If verbose is true, returns a wrapper around a lex instance that
         prints tokens before returning them.
    This distinction should be transparent to yacc.
    """
    my_lex = lex.lex(reflags=re.MULTILINE)

    if verbose:  # creates a wrapper around the lexer that prints
        tokens
        verbose_lexer = VerboseLexer(my_lex)
        return verbose_lexer
    else:
        return my_lex
```

B.3 Parser

```
import sys
import lexer
import ply.yacc as yacc

"""
Node object for tree generation
"""
class Node(object):
    def __init__(self, value, *children):
        self.value = value
        self.children = children

    def __str__(self):
        return self.traverse(1)

    def traverse(self, i):
        s = repr(self.value)
        indent = "\n" + i*' |'
        for child in self.children:
            #print children
            if len(child.value) > 0:
                s += indent + child.traverse(i+1)
        return s

"""
Error handling in the parser
"""
class ParsingError(Exception):
     def __init__(self, value):
         self.value = value
     def __str__(self):
         return repr(self.value)

"""
Creates node leaves for a program node
"""
def p_program(p):
    """
    program : statement_list
            | statement_list translation_unit

    """
    if len(p) == 2:
        p[0] = Node('program', p[1])
    else:
        p[0] = Node('program', p[1], p[2])

"""
Creates node leaves for a block definition
"""
def p_block(p):
    """
    block : ':' NEWLINE INDENT statement_list DEDENT
    """
```

```
        p[0] = Node('block', Node(p[1]), Node(p[2]), Node(p[3]), p[4],
            Node(p[5]))

    """
    Creates node leaves for a function definition
    """
    def p_function_definition(p):
        """
        function_definition   : ID parameter_declaration block
        """
        id_node = Node(p[1])
        p[0] = Node('function_definition', id_node, p[2], p[3])

    """
    Creates node leaves for a translation unit
    """
    def p_translation_unit(p):
        """
        translation_unit    : function_definition
                            | translation_unit function_definition
        """
        if len(p) == 2:
            p[0] = Node('translation_unit', p[1])
        else:
            p[0] = Node('translation_unit', p[1], p[2])

    """
    Creates node leaf for a parameter declaration
    (typically for a function)
    """
    def p_parameter_declaration(p):
        """
        parameter_declaration   : variable
                                | parameter_declaration ',' variable
        """
        if len(p) == 2:
            p[0] = Node('parameter_declaration', p[1])
        else:
            p[0] = Node('parameter_declaration', p[1], Node(p[2]), p[3])

    """
    Creates the node leaf for an epsilon production
    """
    def p_parameter_declaration_eps(p):
        """
        parameter_declaration   : epsilon
        """
        p[0] = p[1]

    """
    Constructs the node leaves for a statement list
    """
    def p_statement_list(p):
        """
        statement_list  : statement
                        | statement_list statement
        """
```

```
        if len(p) == 3:
            p[0] = Node('statement_list', p[1], p[2])
        else:
            p[0] = Node('statement_list', p[1])

    """
    Constructs node leaf for a statement production
    """
    def p_statement(p):
        """
        statement   : expression_statement
                    | selection_statement
                    | iteration_statement
                    | return_statement
        """
        p[0] = Node('statement', p[1])

    """
    Constructs node leaf for a return statement
    """
    def p_return_statement(p):
        """
        return_statement    : RETURN expression_statement
        """
        p[0] = Node('return_statement', Node(p[1]), p[2])

    """
    Constructs node leaf for a full expression statement
    """
    def p_expression_statement(p):
        """
        expression_statement    : expression NEWLINE
        """
        p[0] = Node('expression_statement', p[1], Node(p[2]))

    """
    Constructs a node leaf for an if-else block
    """
    def p_selection_statement(p):
        """
        selection_statement : IF expression block
                            | IF expression block ELSE block
        """
        if_node = Node(p[1])
        if len(p) == 4:
            p[0] = Node('selection_statement', if_node, p[2], p[3])
        else:
            p[0] = Node('selection_statement', if_node, p[2], p[3], Node
                (p[4]), p[5])

    """
    Constructs a node leaf for iteration statements
    i.e. for, forp, while
    """
    def p_iteration_statement(p):
        """
        iteration_statement     : FOR variable IN variable block
```

```
                            | FORP variable IN variable block
                            | WHILE expression block
    """
    if len(p) == 4:
        p[0] = Node('iteration_statement', Node(p[1]), p[2], p[3])
    else:
        forp_node = Node(p[1])
        #id_node = Node(p[2])
        in_node = Node(p[3])
        p[0] = Node('iteration_statement', forp_node, p[2], in_node,
             p[4], p[5])

"""
Constructs a node leaf for an expression
"""
def p_expression(p):
    """
    expression  : assignment_expression
    """
    p[0] = Node('expression', p[1])

"""
Constructs a node leaf for an assignment expression
"""
def p_assignment_expression(p):
    """
    assignment_expression   : variable_expression '='
        assignment_expression
                            | logical_OR_expression
    """
    if len(p) == 2:
        p[0] = Node('assignment_expression', p[1])
    else:
        p[0] = Node('assignment_expression', p[1], Node(p[2]), p[3])

"""
Constructs a node leaf for logical OR expression production
"""
def p_logical_OR_expression(p):
    """
    logical_OR_expression   : logical_AND_expression
                            | logical_OR_expression OR
                                logical_AND_expression
    """
    if len(p) == 2:
        p[0] = Node('logical_OR_expression', p[1])
    else:
        p[0] = Node('logical_OR_expression', p[1], Node(p[2]), p[3])
"""
Constructs a node leaf for logical AND expression production
"""
def p_logical_AND_expression(p):
    """
    logical_AND_expression   : equality_expression
                            | logical_AND_expression AND
                                equality_expression
    """
```

```
        if len(p) == 2:
            p[0] = Node('logical_AND_expression', p[1])
        else:
            p[0] = Node('logical_AND_expression', p[1], Node(p[2]), p
                [3])

    """
    Constructs a node leaf for an equality expression
    """
    def p_equality_expression(p):
        """
        equality_expression   : relational_expression
                              | equality_expression DUBEQUAL
                                  relational_expression
                              | equality_expression NEQUAL
                                  relational_expression
        """
        if len(p) == 2:
            p[0] = Node('equality_expression', p[1])
        else:
            p[0] = Node('equality_expression', p[1], Node(p[2]), p[3])

    """
    Constructs a node leaf for a relational expression
    """
    def p_relational_expression(p):
        """
        relational_expression   : additive_expression
                                | relational_expression '<'
                                    additive_expression
                                | relational_expression '>'
                                    additive_expression
                                | relational_expression LESSTHANEQ
                                    additive_expression
                                | relational_expression GREATERTHANEQ
                                    additive_expression
        """
        if len(p) == 2:
            p[0] = Node('relational_expression', p[1])
        else:
            p[0] = Node('relational_expression', p[1], Node(p[2]), p[3])

    """
    Constructs a node leaf for an additive expression
    """
    def p_additive_expression(p):
        """
        additive_expression : multiplicative_expression
                            | additive_expression '+'
                                multiplicative_expression
                            | additive_expression '-'
                                multiplicative_expression
        """
        if len(p) == 2:
            p[0] = Node('additive_expression', p[1])
        else:
            p[0] = Node('additive_expression', p[1], Node(p[2]), p[3])
```

```
    """
    Constructs a node leaf for a multiplicative expression
    """
    def p_multiplicative_expression(p):
        """
        multiplicative_expression   : logical_NOT_expression
                                    | multiplicative_expression '*'
                                        logical_NOT_expression
                                    | multiplicative_expression '/'
                                        logical_NOT_expression
        """
        if len(p) == 2:
            p[0] = Node('multiplicative_expression', p[1])
        else:
            p[0] = Node('multiplicative_expression', p[1], Node(p[2]), p
                [3])

    """
    Constructs a node leaf for a logical NOT expression
    """
    def p_logical_NOT_expression(p):
        """
        logical_NOT_expression  : intermediate_expression
                                | NOT logical_NOT_expression
        """
        if len(p) == 2:
            p[0] = Node('logical_NOT_expression', p[1])
        else:
            p[0] = Node('logical_NOT_expression', Node(p[1]), p[2])

    """
    Constructs a node leaf for intermediate expressions
    """
    def p_intermediate_expression(p):
        """
        intermediate_expression     : primary_expression
                                    | function_expression
        """
        p[0] = Node('intermediate_expression', p[1])

    """
    Constructs a node leaf for a primary expression token
    tokens are: string, number, true, false
    """
    def p_primary_expression_token(p):
        """
        primary_expression  : STRING
                            | NUMBER
                            | TRUE
                            | FALSE
        """
        p[0] = Node('primary_expression', Node(p[1]))

    """
    Construct a node leaf for primary expressions
    """
```

```
def p_primary_expression(p):
    """
    primary_expression  : variable_access_expression
                        | '[' parameters ']'
                        | '(' expression ')'
    """
    if len(p) == 2:
        p[0] = Node('primary_expression', p[1])
    else:
        p[0] = Node('primary_expression', Node(p[1]), p[2], Node(p
            [3]))

"""
Construct a node leaf for function expressions
"""
def p_function_expression(p):
    """
    function_expression : '#' ID parameters
    """
    hashtag = Node(p[1])
    iden = Node(p[2])
    p[0] = Node('function_expression', hashtag, iden, p[3])

"""
Constructs a node leaf for parameters
"""
def p_parameters(p):
    """
    parameters : primary_expression
               | parameters ',' primary_expression
    """
    if len(p) == 2:
        p[0] = Node('parameters', p[1])
    else:
        p[0] = Node('parameters', p[1], Node(p[2]), p[3])

"""
Constructs a node leaf for the epsilon production of parameters
"""
def p_parameters_eps(p):
    """
    parameters  : epsilon
    """
    p[0] = p[1]

"""
Constructs a node leaf for a variable access expression
"""
def p_variable_access_expression(p):
  """
  variable_access_expression : variable_expression
                             | variable_access_expression '.' ID
  """
  if len(p) == 2:
    p[0] = Node('variable_access_expression', p[1])
  elif len(p) == 4:
```

```
        p[0] = Node('variable_access_expression', p[1], Node(p[2]), Node
            (p[3]))

    """
    Constructs a node leaf for a variable expression
    """
    def p_variable_expression(p):
      """
      variable_expression : variable
                | variable_expression '[' parameters ']'
      """
      if len(p) == 2:
        p[0] = Node('variable_expression', p[1])
      else:
        p[0] = Node('variable_expression', p[1], Node(p[2]), p[3], Node(
            p[4]))

    """
    Constructs a node leaf for a variable
    """
    def p_variable(p):
      """
      variable : '@' ID
      """
      at = Node(p[1])
      iden = Node(p[2])
      p[0] = Node('variable', at, iden)

    """
    Constructs a node leaf for the epsilon production
    """
    def p_epsilon(p):
        """
        epsilon :
        """
        p[0] = Node('')

    """
    Produces a parsing erorr message
    """
    def p_error(p):
        if p is not None:
            sys.stderr.write('syntax error for ' + p.type + ' ' + p.
                value + ' at ' + str(p.lexpos) + '\n')
            raise ParsingError('Parsing error - review the syntax error(
                s) above')

    tokens = lexer.tokens

    """
    Method to run the parser
    """
    def get_yacc():
        return yacc.yacc()
```

B.4 Code Generation

```
"""
keys are nodes that should have spaces printed around them, and
    values say where to print.
1 means before, 2 means after and 3 means both (picture bitwise or).
"""
spaces_table = {
  ',': 2,
  'return': 2,
  'if': 2,
  'for': 2,
  'in': 3,
  'while': 2,
  '=': 3,
  'or': 3,
  'and': 3,
  '!=': 3,
  '==': 3,
  '<': 3,
  '>': 3,
  '<=': 3,
  '>=': 3,
  '+': 3,
  '-': 3,
  '*': 3,
  '/': 3,
  'not': 2
}

'''
keywords corresponding to the list of built in functions
'''
bif_set = {
  'saveImage',
  'grayscale',
  'scale',
  'stretch',
  'rotate',
  'overlay',
  'blur',
  'sharpen',
  'brighten',
  'contrast',
  'border',
  'cropit',
  'caption',
  'color',
  'collage'
}

'''
remove these characters that are only used in fixel
characters not needed for translation
'''
ignore = ['#', '@', '']
```

```
class Generator(object):
  def __init__(self, tree):
    self.indent_level = 0
    self.in_main = True
    self.main_list = []  # output python for main function
    self.function_def_list = []  # output python for rest of program
    self.string_list = self.main_list
    self.tree = tree
    self.forp_pixel_stack = []
    self.forp_image_table = {}
    self.in_forp = False
    self.in_assignment_left = False
    self.process_tree(tree)

  #combines the string corresponding to the main and the funciton
      definitions
  def get_strings(self):
    return ''.join(self.main_list), ''.join(self.function_def_list)

  '''
  this calls the individual functions depending on the node being
      processed
  if no such funciton exists, then default processing is used
  '''
  def process_tree(self, node):
    if node.value in custom_functions_table:  # call custom
        processing function
      custom_function = custom_functions_table[node.value]
      custom_function(self, node)
    else:  # if no custom function, use default processing
      # get spaces information
      spaces = spaces_table.get(node.value, 0)
      space_before = spaces & 1
      space_after = spaces & 2

      if space_before:  # add space before processing
        self.string_list.append(' ')

      if len(node.children) > 0:  # non-leaf
        # process the children
        for child in node.children:
          self.process_tree(child)
      elif node.value not in ignore:  # leaf
        # if it's a leaf add it to the string list
        self.string_list.append(node.value)

      if space_after:
        self.string_list.append(' ')

  '''
  parses the program node
  '''
  def process_program(self, node):
    if len(node.children) == 1:
      self.process_tree(node.children[0])
    else:
```

```
        statement_list, translation_unit = node.children
        self.process_tree(statement_list)
        self.string_list = self.function_def_list
        self.in_main = False
        self.process_tree(translation_unit)

    '''
    parses the function definition node
    '''
    def process_function_definition(self, node):
      id_node, parameter_declaration, block = node.children
      self.string_list.append('def ')
      self.process_tree(id_node)
      self.string_list.append('(')
      self.process_tree(parameter_declaration)
      self.string_list.append(')')
      self.process_tree(block)

    '''
    parses the function expression node
    '''
    def process_function_expression(self, node):
      hashtag, id_node, parameters = node.children
      if id_node.value in bif_set:
        self.string_list.append('fixelFunctions.')
      self.process_tree(id_node)
      self.string_list.append('(')
      self.process_tree(parameters)
      self.string_list.append(')')

    '''
    parses the newline
    '''
    def process_newline(self, node):
      self.string_list.append(node.value)
      for i in range(0, self.indent_level):
        self.string_list.append('\t')

    '''
    parses the indent node
    '''
    def process_indent(self, node):
      self.indent_level += 1
      self.string_list.append('\t')

    '''
    parses the dedent node
    '''
    def process_dedent(self, node):
      self.indent_level -= 1
      del self.string_list[-1]  #todo worry about index errors

    '''
    process a variable node
    '''
    def process_variable(self, node):
      id_node = node.children[1]
```

```
    if self.in_main:
      self.string_list.append('ns.')

    # check if we need to replace pixel with image pixel access
    variable_name = id_node.value
    if self.in_forp and self.in_assignment_left:
      self.in_assignment_left = False

      if variable_name in self.forp_image_table:
        forp_image = self.forp_image_table[variable_name] #assigns
            image based on pixel
        if self.in_main:
          variable_name = 'ns.' + variable_name #prepend 'ns.' if
              necessary
        str_list = [forp_image, '[', variable_name, '.x, ',
            variable_name, '.y', ']'] #changes form to allow for
            pixel access in python
        self.string_list.extend(str_list)
      else:
        self.process_tree(id_node)
    else:
      self.process_tree(id_node)


  '''
  process iteration statements
  '''
  def process_iteration_statement(self, node):
    children = node.children
    if children[0].value == 'forp':
      self.in_forp = True

      # grab names of two variables
      image = children[3].children[1].value
      pixel = children[1].children[1].value
      self.forp_pixel_stack.append(pixel)
      self.forp_image_table[pixel] = image

      if self.in_main:  # prepend 'ns.' if needed
        pixel = 'ns.' + pixel
        image = 'ns.' + image

      # create the pixel object
      self.string_list.extend([pixel, ' = runtime_classes.Pixel(',
          image, ')\n', ('\t' * self.indent_level)])

      # create two loops and set pixel color
      self.indent_level += 1  # increment for first loop
      self.string_list.extend(['for ', pixel, '.x in xrange(0, ',
          image, '.width):\n', ('\t' * self.indent_level)])
      self.indent_level += 1  # increment for second loop
      self.string_list.extend(['for ', pixel, '.y in xrange(0, ',
          image, '.height):\n', ('\t' * self.indent_level)])

      # only process the statement list of the block
      block = children[4]
      self.process_tree(block.children[3])
```

```
        # get back to state before the forp loop
        self.indent_level -= 2
        del self.string_list[-2:]
        pixel = self.forp_pixel_stack[-1]
        del self.forp_image_table[pixel]
        del self.forp_pixel_stack[-1]
        if len(self.forp_pixel_stack) == 0:
          self.in_forp = False
      else:
        for child in children:
          self.process_tree(child)

    '''
    processes node corresponding to assignment expressions
    '''
    def process_assignment_expression(self, node):
      children = node.children
      if self.in_forp and len(children) == 3: #special forp case
        self.in_assignment_left = True
      for child in children: #process children on the tree
        self.process_tree(child)

    def process_true(self, node): #replaces 'true' with 'True' which
        will be valid in Python
      self.string_list.append('True')

    def process_false(self, node): #replaces 'false' with 'False'
        which will be valid in Python
      self.string_list.append('False')


'''
list of functions for nodes which require special custom processing
'''
custom_functions_table = {
  'program': Generator.process_program,
  'function_definition': Generator.process_function_definition,
  'function_expression': Generator.process_function_expression,
  '\n': Generator.process_newline,
  'INDENT': Generator.process_indent,
  'DEDENT': Generator.process_dedent,
  'variable': Generator.process_variable,
  'iteration_statement': Generator.process_iteration_statement,
  'assignment_expression': Generator.process_assignment_expression,
  'true': Generator.process_true,
  'false': Generator.process_false
}
```

```
import os


def create_program(main_string, functions_string):
  path_to_fixel_src = os.path.join(os.path.dirname(__file__), '..')
  return header + repr(path_to_fixel_src) + '))\n' +
      import_fixel_functions + functions_string + main_pre_fixel +
      main_string + main_post_fixel
```

```
header = '''\
\'\'\'
  ______   __      __  __      ______      __
 /\  ___\ /\ \    /\_\_\_\    /\  ___\    /\ \
 \ \  __\ \ \ \   \/_/\_\/_   \ \  __\    \ \ \____
  \ \_\    \ \_\    /\_\/\_\   \ \_____\   \ \_____\
   \/_/     \/_/    \/_/\/_/    \/_____/    \/_____/

\'\'\'

import os
import sys

# add fixel top to path so fixel functions can be imported
sys.path.append(os.path.abspath('''

import_fixel_functions = '''
from runtime import fixelFunctions
from runtime import runtime_classes

'''


main_pre_fixel = '''
inputImages = sys.argv[1:]
if len(inputImages) < 1:
  print "\\nNo images were used as arguments. Please append the
      paths to the images you'd like to use as arguments and run
      this Fixel program again.\\n"
  sys.exit(0)
inputImageCount = 0
Namespace = type('Namespace', (object,), {'images': []})  # cleaner
    than having to declare a class
ns = Namespace()

# create variables for each image
for currentImage in inputImages:
  image = runtime_classes.Image(currentImage)
  setattr(ns, "image"+str(inputImageCount), image)
  ns.images.append(image)
  inputImageCount += 1

'''

main_post_fixel = '''
for image in ns.images:
  fixelFunctions.saveImage(image, "JPEG")
'''
```

```
import parser
import lexer
import generator
import post_process
```

```
def translate(source_string=None, verbose=False):
  my_lex = lexer.get_lex(verbose)
  my_parser = parser.get_yacc()
  if source_string is None:
    source_string = 'if @hey < not 1:\n\t#sup\n#hey @image1,@image2\
        nsup:\n\treturn 5\n'

  if verbose:
    print("source:\n")
    print(source_string + '\n')
    print('tokens:\n')

  tree = my_parser.parse(source_string, lexer=my_lex)
  gen = generator.Generator(tree)
  main_string, functions_string = gen.get_strings()
  python_string = post_process.create_program(main_string,
      functions_string)

  if verbose:
    print("\nAST:\n")
    print(str(tree) + '\n')
    print('python:\n')
    print(python_string)

  return python_string
```

B.5 Testing

```
from unittest import TestCase
from translator import lexer
import test_strings


class LexerTests(TestCase):
  def oneliner_test(self):
    self.run_lexer_on_source(test_strings.oneliner_source,
        test_strings.oneliner_tokens)

  def indent_test(self):
    self.run_lexer_on_source(test_strings.indent_source,
        test_strings.indent_tokens)

  def function_def_test(self):
    self.run_lexer_on_source(test_strings.function_def_source,
        test_strings.function_def_tokens)

  def or_and_test(self):
    self.run_lexer_on_source(test_strings.or_and_source,
        test_strings.or_and_tokens)

  def not_test(self):
    self.run_lexer_on_source(test_strings.not_source, test_strings.
        not_tokens)

  def selection_if_test(self):
```

```
      self.run_lexer_on_source(test_strings.selection_if_source,
          test_strings.selection_if_tokens)

    def selection_ifelse_test(self):
      self.run_lexer_on_source(test_strings.selection_ifelse_source,
          test_strings.selection_ifelse_tokens)

    def iteration_for_test(self):
      self.run_lexer_on_source(test_strings.iteration_for_source,
          test_strings.iteraton_for_tokens)

    def multiplicative_test(self):
      self.run_lexer_on_source(test_strings.multiplicative_source,
          test_strings.multiplicative_tokens)

    def equality_notequal_test(self):
      self.run_lexer_on_source(test_strings.equality_notequal_source,
          test_strings.equality_notequal_tokens)

    def pixel_test(self):
      self.run_lexer_on_source(test_strings.pixel_source, test_strings
          .pixel_tokens)

    def forp_test(self):
      self.run_lexer_on_source(test_strings.forp_source, test_strings.
          forp_tokens)

    def run_lexer_on_source(self, source, expected_tokens):
      my_lex = lexer.get_lex()
      my_lex.input(source)

      for expected_token in expected_tokens:
        actual_token = my_lex.token()
        self.assertEqual(expected_token, (actual_token.type,
            actual_token.value))

      self.assertIsNone(my_lex.token())  # make sure token stream was
          exhausted
```

```
import difflib
from unittest import TestCase
import mock
from translator import parser
import test_utils
import test_strings


class ParserTests(TestCase):
  def test_test(self):
    self.assertTrue(True)

  def oneliner_test(self):
    self.run_parser_on_tokens(test_strings.oneliner_tokens,
        test_strings.oneliner_tree)

  def indent_test(self):
```

```
    self.run_parser_on_tokens(test_strings.indent_tokens,
        test_strings.indent_tree)

  def function_def_test(self):
    self.run_parser_on_tokens(test_strings.function_def_tokens,
        test_strings.function_def_tree)

  def or_and_test(self):
    self.run_parser_on_tokens(test_strings.or_and_tokens,
        test_strings.or_and_tree)

  def not_test(self):
    self.run_parser_on_tokens(test_strings.not_tokens, test_strings.
        not_tree)

  def selection_if_test(self):
    self.run_parser_on_tokens(test_strings.selection_if_tokens,
        test_strings.selection_if_tree)

  def selection_ifelse_test(self):
    self.run_parser_on_tokens(test_strings.selection_ifelse_tokens,
        test_strings.selection_ifelse_tree)

  def iteration_for_test(self):
    self.run_parser_on_tokens(test_strings.iteraton_for_tokens,
        test_strings.iteration_for_tree)

  def multiplicative_test(self):
    self.run_parser_on_tokens(test_strings.multiplicative_tokens,
        test_strings.multiplicative_tree)

  def equality_notequal_test(self):
    self.run_parser_on_tokens(test_strings.equality_notequal_tokens,
         test_strings.equality_notequal_tree)

  def pixel_test(self):
    self.run_parser_on_tokens(test_strings.pixel_tokens,
        test_strings.pixel_tree)

  def forp_test(self):
    self.run_parser_on_tokens(test_strings.forp_tokens, test_strings
        .forp_tree)

  def run_parser_on_tokens(self, tokens, expected_tree):
    mock_lex = mock.MagicMock()
    mock_tokens = []
    for token in tokens:
      mock_token = test_utils.get_mock_token(token[0], token[1])
      mock_tokens.append(mock_token)

    # pad iteration in case lookahead goes past end of stream
    mock_lex.token.side_effect = mock_tokens + [None, None]

    my_parser = parser.get_yacc()
    tree = my_parser.parse('', lexer=mock_lex)
    tree_string = test_utils.tree_to_string(tree)
    diff_string = self.get_diff(expected_tree, tree_string)
```

```
      self.assertEqual(expected_tree.replace(' ', ''), tree_string.
          replace(' ', ''), 'diff:\n' + diff_string)

    def get_diff(self, tree1, tree2):
      diff_gen = difflib.context_diff(tree1.split(' '), tree2.split('
          '))
      diff_string = ''
      for diff in diff_gen:
        diff_string += diff
      return diff_string
```

```
from unittest import TestCase
from translator import generator
import test_strings
import test_utils


class GeneratorTests(TestCase):
  def oneliner_test(self):
    self.run_generator_on_tree(test_strings.oneliner_tree,
        test_strings.oneliner_python)

  def indent_test(self):
    self.run_generator_on_tree(test_strings.indent_tree,
        test_strings.indent_python)

  def function_def_test(self):
    self.run_generator_on_tree(test_strings.function_def_tree,
        test_strings.function_def_python)

  def or_and_test(self):
    self.run_generator_on_tree(test_strings.or_and_tree,
        test_strings.or_and_python)

  def not_test(self):
    self.run_generator_on_tree(test_strings.not_tree, test_strings.
        not_python)

  def selection_if_test(self):
    self.run_generator_on_tree(test_strings.selection_if_tree,
        test_strings.selection_if_python)

  def selection_ifelse_test(self):
    self.run_generator_on_tree(test_strings.selection_ifelse_tree,
        test_strings.selection_ifelse_python)

  def equality_notequal_test(self):
    self.run_generator_on_tree(test_strings.equality_notequal_tree,
        test_strings.equality_notequal_python)

  def iteration_for_test(self):
    self.run_generator_on_tree(test_strings.iteration_for_tree,
        test_strings.iteration_for_python)

  def multiplicative_test(self):
    self.run_generator_on_tree(test_strings.multiplicative_tree,
        test_strings.multiplicative_python)
```

```
  def pixel_test(self):
    self.run_generator_on_tree(test_strings.pixel_tree, test_strings
        .pixel_python)

  def forp_test(self):
    self.run_generator_on_tree(test_strings.forp_tree, test_strings.
        forp_python)

  def run_generator_on_tree(self, tree_string, expected_python):
    tree_string_no_ws = tree_string.replace(' ', '')
    tree, _ = test_utils.string_to_tree(tree_string_no_ws, 0)
    gen = generator.Generator(tree)
    main_string, functions_string = gen.get_strings()
    self.assertEqual(expected_python[0], main_string)
    self.assertEqual(expected_python[1], functions_string)
```

```
import difflib
from unittest import TestCase
from translator import translator
from translator import post_process
import test_strings


class FixelTests(TestCase):
  def oneliner_test(self):
    self.run_fixel_on_input(test_strings.oneliner_source,
        test_strings.oneliner_python)

  def complex_test(self):
    self.run_fixel_on_input(test_strings.complex_source,
        test_strings.complex_python)

  def big_test(self):
    self.run_fixel_on_input(test_strings.big_source, test_strings.
        big_python)

  def run_fixel_on_input(self, source, expected_python):
    fixel_output = translator.translate(source)

    # post process is too simple to test, just call to get proper
        output
    expected_python_post_process = post_process.create_program(
        expected_python[0], expected_python[1])

    d = difflib.Differ()
    diff = d.compare(fixel_output.splitlines(),
        expected_python_post_process.splitlines())
    diff_string = '\n'.join(diff)
    self.assertEqual(fixel_output, expected_python_post_process, '
        diff\n' + diff_string)
```

```
'''
source strings
'''
```

```
oneliner_source = '#grayscale @image1\n'

indent_source = '''\
if @hey < not 1:
  #sup
#hey @image1
for @image in @images:
  if @image == @image1:
    #hey @image
hey @myImage:
  if @myImage.width > 100:
    if @myImage.height > 50 and @myImage.height < 60:
      @x = 10
      @y = 50
  else:
    @x = 100
    @y = 20
  return @x
'''

function_def_source = '''\
#sup @image1, @image2

sup @myImage1, @myImage2:
  return @myImage1.height + @myImage2.height
'''

or_and_source = '''\
true and false or @str == "cool string"
true and (false or @str == "cool string")
'''

not_source = '''\
@myList = [40, 500]
not 100 <= @myList[1]
'''

selection_if_source = '''\
if @x<1:
  @y=5
'''

selection_ifelse_source = '''\
if @x>1 and @y>=1:
  @y=@y-5
else:
  @y=3
'''

iteration_for_source = '''\
for @image in @images:
  @x=@x+1
'''

multiplicative_source = '''\
@x=@y*5
@y=@x/5
```

```
'''

equality_notequal_source = '''\
@test=(5!=3)
'''

color_setting_source = '''\
@red = color(0,0,0)
@image1[302,19] = @red
'''

complex_source = '''\
#ghost @image1
#my_collage @image1, @image2, @image3, @image4

ghost @image:
  #blur @image
  #grayscale @image

my_collage @image1, @image2, @image3, @image4:
  @my_list = [@image1, @image2, @image3, @image4]
  for @image in @my_list:
    if @image == @image1 or @image == @image2:
      #sharpen @image
    else:
      #caption @image, "pretty colors"
'''

pixel_source = '''\
@color = @image1[1,1]
#grayscale @image1
@image1[1,1] = @color
'''

forp_source = '''\
forp @pixel in @image1:
  @color = @pixel.color
  @pixel = @color
'''

big_source = '''\
@l1 = [@image0, @image1, @image2]
@i2 = [@image3, @image4, @image5]

for @image in @l1:
  #invert @image

for @image in @l2:
  #my_blur @image

invert @image:
  forp @p in @image:
    @p = 255 - @p.color

my_blur @image:
  forp @p1 in @image:
    forp @p2 in @image:
```

```
      if @p2.y - @p1.y < 5:
        @p1 = (@p1.color + @p2.color)/2
'''

'''
token streams
'''


oneliner_tokens = [('#', '#'), ('ID', 'grayscale'), ('@', '@'), ('ID
    ', 'image1'), ('NEWLINE', '\n')]

indent_tokens = [('IF', 'if'), ('@', '@'), ('ID', 'hey'), ('<', '<')
    , ('NOT', 'not'), ('NUMBER', '1'), (':', ':'), ('NEWLINE', '\n')
    , ('INDENT', 'INDENT'), ('#', '#'), ('ID', 'sup'), ('NEWLINE', '
    \n'), ('DEDENT', 'DEDENT'), ('#', '#'), ('ID', 'hey'), ('@', '@'
    ), ('ID', 'image1'), ('NEWLINE', '\n'), ('FOR', 'for'), ('@', '@
    '), ('ID', 'image'), ('IN', 'in'), ('@', '@'), ('ID', 'images'),
     (':', ':'), ('NEWLINE', '\n'), ('INDENT', 'INDENT'), ('IF', 'if
    '), ('@', '@'), ('ID', 'image'), ('DUBEQUAL', '=='), ('@', '@'),
     ('ID', 'image1'), (':', ':'), ('NEWLINE', '\n'), ('INDENT', '
    INDENT'), ('#', '#'), ('ID', 'hey'), ('@', '@'), ('ID', 'image')
    , ('NEWLINE', '\n'), ('DEDENT', 'DEDENT'), ('DEDENT', 'DEDENT'),
     ('ID', 'hey'), ('@', '@'), ('ID', 'myImage'), (':', ':'), ('
    NEWLINE', '\n'), ('INDENT', 'INDENT'), ('IF', 'if'), ('@', '@'),
     ('ID', 'myImage'), ('.', '.'), ('ID', 'width'), ('>', '>'), ('
    NUMBER', '100'), (':', ':'), ('NEWLINE', '\n'), ('INDENT', '
    INDENT'), ('IF', 'if'), ('@', '@'), ('ID', 'myImage'), ('.', '.'
    ), ('ID', 'height'), ('>', '>'), ('NUMBER', '50'), ('AND', 'and'
    ), ('@', '@'), ('ID', 'myImage'), ('.', '.'), ('ID', 'height'),
    ('<', '<'), ('NUMBER', '60'), (':', ':'), ('NEWLINE', '\n'), ('
    INDENT', 'INDENT'), ('@', '@'), ('ID', 'x'), ('=', '='), ('
    NUMBER', '10'), ('NEWLINE', '\n'), ('@', '@'), ('ID', 'y'), ('='
    , '='), ('NUMBER', '50'), ('NEWLINE', '\n'), ('DEDENT', 'DEDENT'
    ), ('DEDENT', 'DEDENT'), ('ELSE', 'else'), (':', ':'), ('NEWLINE
    ', '\n'), ('INDENT', 'INDENT'), ('@', '@'), ('ID', 'x'), ('=', '
    ='), ('NUMBER', '100'), ('NEWLINE', '\n'), ('@', '@'), ('ID', 'y
    '), ('=', '='), ('NUMBER', '20'), ('NEWLINE', '\n'), ('DEDENT',
    'DEDENT'), ('RETURN', 'return'), ('@', '@'), ('ID', 'x'), ('
    NEWLINE', '\n'), ('DEDENT', 'DEDENT')]

function_def_tokens = [('#', '#'), ('ID', 'sup'), ('@', '@'), ('ID',
     'image1'), (',', ','), ('@', '@'), ('ID', 'image2'), ('NEWLINE'
    , '\n'), ('ID', 'sup'), ('@', '@'), ('ID', 'myImage1'), (',', ',
    '), ('@', '@'), ('ID', 'myImage2'), (':', ':'), ('NEWLINE', '\n'
    ), ('INDENT', 'INDENT'), ('RETURN', 'return'), ('@', '@'), ('ID'
    , 'myImage1'), ('.', '.'), ('ID', 'height'), ('+', '+'), ('@', '
    @'), ('ID', 'myImage2'), ('.', '.'), ('ID', 'height'), ('NEWLINE
    ', '\n'), ('DEDENT', 'DEDENT')]

or_and_tokens = [('TRUE', 'true'), ('AND', 'and'), ('FALSE', 'false'
    ), ('OR', 'or'), ('@', '@'), ('ID', 'str'), ('DUBEQUAL', '=='),
    ('STRING', '"cool string"'), ('NEWLINE', '\n'), ('TRUE', 'true')
    , ('AND', 'and'), ('(', '('), ('FALSE', 'false'), ('OR', 'or'),
    ('@', '@'), ('ID', 'str'), ('DUBEQUAL', '=='), ('STRING', '"cool
     string"'), (')', ')'), ('NEWLINE', '\n')]
```

```
not_tokens = [('@', '@'), ('ID', 'myList'), ('=', '='), ('[', '['),
    ('NUMBER', '40'), (',', ','), ('NUMBER', '500'), (']', ']'), ('
    NEWLINE', '\n'), ('NOT', 'not'), ('NUMBER', '100'), ('LESSTHANEQ
    ', '<='), ('@', '@'), ('ID', 'myList'), ('[', '['), ('NUMBER', '
    1'), (']', ']'), ('NEWLINE', '\n')]

selection_if_tokens = [('IF', 'if'), ('@', '@'), ('ID', 'x'), ('<',
    '<'), ('NUMBER', '1'), (':', ':'), ('NEWLINE', '\n'), ('INDENT',
     'INDENT'), ('@', '@'), ('ID', 'y'), ('=', '='), ('NUMBER', '5')
    , ('NEWLINE', '\n'), ('DEDENT', 'DEDENT')]

selection_ifelse_tokens = [('IF', 'if'), ('@', '@'), ('ID', 'x'), ('
    >', '>'), ('NUMBER', '1'), ('AND', 'and'), ('@', '@'), ('ID', 'y
    '), ('GREATERTHANEQ', '>='), ('NUMBER', '1'), (':', ':'), ('
    NEWLINE', '\n'), ('INDENT', 'INDENT'), ('@', '@'), ('ID', 'y'),
    ('=', '='), ('@', '@'), ('ID', 'y'), ('-', '-'), ('NUMBER', '5')
    , ('NEWLINE', '\n'), ('DEDENT', 'DEDENT'), ('ELSE', 'else'), (':
    ', ':'), ('NEWLINE', '\n'), ('INDENT', 'INDENT'), ('@', '@'), ('
    ID', 'y'), ('=', '='), ('NUMBER', '3'), ('NEWLINE', '\n'), ('
    DEDENT', 'DEDENT')]

iteraton_for_tokens = [('FOR', 'for'), ('@', '@'), ('ID', 'image'),
    ('IN', 'in'), ('@', '@'), ('ID', 'images'), (':', ':'), ('
    NEWLINE', '\n'), ('INDENT', 'INDENT'), ('@', '@'), ('ID', 'x'),
    ('=', '='), ('@', '@'), ('ID', 'x'), ('+', '+'), ('NUMBER', '1')
    , ('NEWLINE', '\n'), ('DEDENT', 'DEDENT')]

multiplicative_tokens = [('@', '@'), ('ID', 'x'), ('=', '='), ('@',
    '@'), ('ID', 'y'), ('*', '*'), ('NUMBER', '5'), ('NEWLINE', '\n'
    ), ('@', '@'), ('ID', 'y'), ('=', '='), ('@', '@'), ('ID', 'x'),
     ('/', '/'), ('NUMBER', '5'), ('NEWLINE', '\n')]

equality_notequal_tokens = [('@', '@'), ('ID', 'test'), ('=', '='),
    ('(', '('), ('NUMBER', '5'), ('NEQUAL', '!='), ('NUMBER', '3'),
    (')', ')'), ('NEWLINE', '\n')]

color_setting_tokens = [('@', '@'), ('ID', 'red'), ('=', '='), ('ID'
    , 'color'), ('(', '('), ('NUMBER', '0'), (',', ','), ('NUMBER',
    '0'), (',', ','), ('NUMBER', '0'), (')', ')'), ('NEWLINE', '\n')
    , ('@', '@'), ('ID', 'image1'), ('=', '='), ('@', '@'), ('ID', '
    red') ]

pixel_tokens = [('@', '@'), ('ID', 'color'), ('=', '='), ('@', '@'),
     ('ID', 'image1'), ('[', '['), ('NUMBER', '1'), (',', ','), ('
    NUMBER', '1'), (']', ']'), ('NEWLINE', '\n'), ('#', '#'), ('ID',
     'grayscale'), ('@', '@'), ('ID', 'image1'), ('NEWLINE', '\n'),
    ('@', '@'), ('ID', 'image1'), ('[', '['), ('NUMBER', '1'), (',',
     ','), ('NUMBER', '1'), (']', ']'), ('=', '='), ('@', '@'), ('ID
    ', 'color'), ('NEWLINE', '\n')]

forp_tokens = [('FORP', 'forp'), ('@', '@'), ('ID', 'pixel'), ('IN',
     'in'), ('@', '@'), ('ID', 'image1'), (':', ':'), ('NEWLINE', '\
    n'), ('INDENT', 'INDENT'), ('@', '@'), ('ID', 'color'), ('=', '=
    '), ('@', '@'), ('ID', 'pixel'), ('.', '.'), ('ID', 'color'), ('
    NEWLINE', '\n'), ('@', '@'), ('ID', 'pixel'), ('=', '='), ('@',
    '@'), ('ID', 'color'), ('NEWLINE', '\n'), ('DEDENT', 'DEDENT')]
```

```
'''
abstract syntax tress
'''


oneliner_tree = '[program [statement_list [statement [
    expression_statement [expression [assignment_expression [
    logical_OR_expression [logical_AND_expression [
    equality_expression [relational_expression [additive_expression
    [multiplicative_expression [logical_NOT_expression [
    intermediate_expression [function_expression [#] [grayscale] [
    parameters [primary_expression [variable_access_expression [
    variable_expression [variable [@] [image1]]]]]]]]]]]]]]]] [\n
    ]]]]]'

indent_tree = '[program [statement_list [statement_list [
    statement_list [statement [selection_statement [if] [expression
    [assignment_expression [logical_OR_expression [
    logical_AND_expression [equality_expression [
    relational_expression [relational_expression [
    additive_expression [multiplicative_expression [
    logical_NOT_expression [intermediate_expression [
    primary_expression [variable_access_expression [
    variable_expression [variable [@] [hey]]]]]]]]]] [<] [
    additive_expression [multiplicative_expression [
    logical_NOT_expression [not] [logical_NOT_expression [
    intermediate_expression [primary_expression [1]]]]]]]]]]]] [
    block [:] [\n] [INDENT] [statement_list [statement [
    expression_statement [expression [assignment_expression [
    logical_OR_expression [logical_AND_expression [
    equality_expression [relational_expression [additive_expression
    [multiplicative_expression [logical_NOT_expression [
    intermediate_expression [function_expression [#] [sup]
    [[[[[[[[[[[[]]]] [\n]]]] [DEDENT]]]]] [statement [
    expression_statement [expression [assignment_expression [
    logical_OR_expression [logical_AND_expression [
    equality_expression [relational_expression [additive_expression
    [multiplicative_expression [logical_NOT_expression [
    intermediate_expression [function_expression [#] [hey] [
    parameters [primary_expression [variable_access_expression [
    variable_expression [variable [@] [image1]]]]]]]]]]]]]]]] [\n
    ]]]] [statement [iteration_statement [for] [variable [@] [image
    ]] [in] [variable [@] [images]] [block [:] [\n] [INDENT] [
    statement_list [statement [selection_statement [if] [expression
    [assignment_expression [logical_OR_expression [
    logical_AND_expression [equality_expression [equality_expression
     [relational_expression [additive_expression [
    multiplicative_expression [logical_NOT_expression [
    intermediate_expression [primary_expression [
    variable_access_expression [variable_expression [variable [@] [
    image]]]]]]]]]] [==] [relational_expression [
    additive_expression [multiplicative_expression [
    logical_NOT_expression [intermediate_expression [
    primary_expression [variable_access_expression [
    variable_expression [variable [@] [image1]]]]]]]]]]]]] [block
    [:] [\n] [INDENT] [statement_list [statement [
```

```
expression_statement [expression [assignment_expression [
logical_OR_expression [logical_AND_expression [
equality_expression [relational_expression [additive_expression
[multiplicative_expression [logical_NOT_expression [
intermediate_expression [function_expression [#] [hey] [
parameters [primary_expression [variable_access_expression [
variable_expression [variable [@] [image]]]]]]]]]]]]]]]] [\n
]]]] [DEDENT]]]]] [DEDENT]]]]] [translation_unit [
function_definition [hey] [parameter_declaration [variable [@] [
myImage]]] [block [:] [\n] [INDENT] [statement_list [
statement_list [statement [selection_statement [if] [expression
[assignment_expression [logical_OR_expression [
logical_AND_expression [equality_expression [
relational_expression [relational_expression [
additive_expression [multiplicative_expression [
logical_NOT_expression [intermediate_expression [
primary_expression [variable_access_expression [
variable_access_expression [variable_expression [variable [@] [
myImage]]]] [.] [width]]]]]]]] [>] [additive_expression [
multiplicative_expression [logical_NOT_expression [
intermediate_expression [primary_expression [100]]]]]]]]]]] [
block [:] [\n] [INDENT] [statement_list [statement [
selection_statement [if] [expression [assignment_expression [
logical_OR_expression [logical_AND_expression [
logical_AND_expression [equality_expression [
relational_expression [relational_expression [
additive_expression [multiplicative_expression [
logical_NOT_expression [intermediate_expression [
primary_expression [variable_access_expression [
variable_access_expression [variable_expression [variable [@] [
myImage]]]] [.] [height]]]]]]]] [>] [additive_expression [
multiplicative_expression [logical_NOT_expression [
intermediate_expression [primary_expression [50]]]]]]]] [and] [
equality_expression [relational_expression [
relational_expression [additive_expression [
multiplicative_expression [logical_NOT_expression [
intermediate_expression [primary_expression [
variable_access_expression [variable_access_expression [
variable_expression [variable [@] [myImage]]]] [.] [height
]]]]]]]] [<] [additive_expression [multiplicative_expression [
logical_NOT_expression [intermediate_expression [
primary_expression [60]]]]]]]]]]] [block [:] [\n] [INDENT] [
statement_list [statement_list [statement [expression_statement
[expression [assignment_expression [variable_expression [
variable [@] [x]]] [=] [assignment_expression [
logical_OR_expression [logical_AND_expression [
equality_expression [relational_expression [additive_expression
[multiplicative_expression [logical_NOT_expression [
intermediate_expression [primary_expression [10]]]]]]]]]]]]] [\n
]]]] [statement [expression_statement [expression [
assignment_expression [variable_expression [variable [@] [y]]]
[=] [assignment_expression [logical_OR_expression [
logical_AND_expression [equality_expression [
relational_expression [additive_expression [
multiplicative_expression [logical_NOT_expression [
intermediate_expression [primary_expression [50]]]]]]]]]]]]] [\n
]]]] [DEDENT]]]]] [DEDENT]] [else] [block [:] [\n] [INDENT] [
```

```
        statement_list [statement_list [statement [expression_statement
        [expression [assignment_expression [variable_expression [
        variable [@] [x]]] [=] [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [100]]]]]]]]]]]] [\
        n]]]] [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [y]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [20]]]]]]]]]]]] [\n
        ]]]] [DEDENT]]]]] [statement [return_statement [return] [
        expression_statement [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [variable [@] [x
        ]]]]]]]]]]]]]] [\n]]]]] [DEDENT]]]]]'

    function_def_tree = '[program [statement_list [statement [
        expression_statement [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [function_expression [#] [sup] [
        parameters [parameters [primary_expression [
        variable_access_expression [variable_expression [variable [@] [
        image1]]]]]] [,] [primary_expression [variable_access_expression
         [variable_expression [variable [@] [image2]]]]]]]]]]]]]]] [\n
        ]]]] [translation_unit [function_definition [sup] [
        parameter_declaration [parameter_declaration [variable [@] [
        myImage1]]] [,] [variable [@] [myImage2]]] [block [:] [\n] [
        INDENT] [statement_list [statement [return_statement [return] [
        expression_statement [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_access_expression [variable_expression [variable [@] [
        myImage1]]]] [.] [height]]]]]]] [+] [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_access_expression [variable_expression [variable [@] [
        myImage2]]]] [.] [height]]]]]]]]]] [\n]]]]] [DEDENT]]]]]'

    or_and_tree = '[program [statement_list [statement_list [statement [
        expression_statement [expression [assignment_expression [
        logical_OR_expression [logical_OR_expression [
        logical_AND_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [true]]]]]]]] [and]
```

```
        [equality_expression [relational_expression [
      additive_expression [multiplicative_expression [
      logical_NOT_expression [intermediate_expression [
      primary_expression [false]]]]]]]]] [or] [logical_AND_expression
       [equality_expression [equality_expression [
      relational_expression [additive_expression [
      multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [
      variable_access_expression [variable_expression [variable [@] [
      str]]]]]]]]]] [==] [relational_expression [additive_expression
      [multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression ["cool string
      "]]]]]]]]]] [\n]]]] [statement [expression_statement [
      expression [assignment_expression [logical_OR_expression [
      logical_AND_expression [logical_AND_expression [
      equality_expression [relational_expression [additive_expression
      [multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [true]]]]]]]] [and]
       [equality_expression [relational_expression [
      additive_expression [multiplicative_expression [
      logical_NOT_expression [intermediate_expression [
      primary_expression [(] [expression [assignment_expression [
      logical_OR_expression [logical_OR_expression [
      logical_AND_expression [equality_expression [
      relational_expression [additive_expression [
      multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [false]]]]]]]]] [or
      ] [logical_AND_expression [equality_expression [
      equality_expression [relational_expression [additive_expression
      [multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [
      variable_access_expression [variable_expression [variable [@] [
      str]]]]]]]]]] [==] [relational_expression [additive_expression
      [multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression ["cool string
      "]]]]]]]]]] [)]]]]]]]]]]] [\n]]]]]'

not_tree = '[program [statement_list [statement_list [statement [
      expression_statement [expression [assignment_expression [
      variable_expression [variable [@] [myList]]] [=] [
      assignment_expression [logical_OR_expression [
      logical_AND_expression [equality_expression [
      relational_expression [additive_expression [
      multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [\\[] [parameters [
      parameters [primary_expression [40]]] [,] [primary_expression
      [500]]] [\\]]]]]]]]]]]] [\n]]]] [statement [
      expression_statement [expression [assignment_expression [
      logical_OR_expression [logical_AND_expression [
      equality_expression [relational_expression [
      relational_expression [additive_expression [
      multiplicative_expression [logical_NOT_expression [not] [
      logical_NOT_expression [intermediate_expression [
      primary_expression [100]]]]]]] [<=] [additive_expression [
      multiplicative_expression [logical_NOT_expression [
      intermediate_expression [primary_expression [
      variable_access_expression [variable_expression [
```

```
        variable_expression [variable [@] [myList]]] [\\[] [parameters [
        primary_expression [1]]] [\\]]]]]]]]]]]]]]] [\n]]]]]'

    selection_if_tree = '[program [statement_list [statement [
        selection_statement [if] [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [variable [@] [x
        ]]]]]]]]]] [<] [additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [1]]]]]]]]]]] [block [:] [\n] [INDENT] [
        statement_list [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [y]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [5]]]]]]]]]]]] [\n
        ]]]] [DEDENT]]]]]]'

    selection_ifelse_tree = '[program [statement_list [statement [
        selection_statement [if] [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        logical_AND_expression [equality_expression [
        relational_expression [relational_expression [
        additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_expression [variable [@] [x]]]]]]]]] [>] [
        additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [1]]]]]]]]] [and] [equality_expression [
        relational_expression [relational_expression [
        additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_expression [variable [@] [y]]]]]]]]] [>=] [
        additive_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [1]]]]]]]]]] [block [:] [\n] [INDENT] [
        statement_list [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [y]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [variable [@] [y
        ]]]]]]]]] [-] [multiplicative_expression [logical_NOT_expression
         [intermediate_expression [primary_expression [5]]]]]]]]]]]] [\
        n]]]] [DEDENT]] [else] [block [:] [\n] [INDENT] [statement_list
        [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [y]]]
        [=] [assignment_expression [logical_OR_expression [
```

```
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [3]]]]]]]]]]]] [\n
        ]]]] [DEDENT]]]]]]'

    iteration_for_tree = '[program [statement_list [statement [
        iteration_statement [for] [variable [@] [image]] [in] [variable
        [@] [images]] [block [:] [\n] [INDENT] [statement_list [
        statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [x]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [variable [@] [x
        ]]]]]]]]] [+] [multiplicative_expression [logical_NOT_expression
         [intermediate_expression [primary_expression [1]]]]]]]]]]]]] [\
        n]]]] [DEDENT]]]]]]'

    multiplicative_tree = '[program [statement_list [statement_list [
        statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [x]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_expression [variable [@] [y]]]]]]]] [*] [
        logical_NOT_expression [intermediate_expression [
        primary_expression [5]]]]]]]]]]]]] [\n]]]] [statement [
        expression_statement [expression [assignment_expression [
        variable_expression [variable [@] [y]]] [=] [
        assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [multiplicative_expression [
        logical_NOT_expression [intermediate_expression [
        primary_expression [variable_access_expression [
        variable_expression [variable [@] [x]]]]]]]] [/] [
        logical_NOT_expression [intermediate_expression [
        primary_expression [5]]]]]]]]]]]]] [\n]]]]]'

    equality_notequal_tree = '[program [statement_list [statement [
        expression_statement [expression [assignment_expression [
        variable_expression [variable [@] [test]]] [=] [
        assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [(] [expression [
        assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [equality_expression
         [relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
```

```
        intermediate_expression [primary_expression [5]]]]]]]] [!=] [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [3]]]]]]]]]]] ()
        ]]]]]]]]]]]] [\n]]]]]'

    color_setting_tree = '[program [statement_list [statement [
        expression_statement [expression [assignment_expression [
        variable_expression [variable [@] [test]]] [=] [
        assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [(] [expression [
        assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [equality_expression
         [relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [5]]]]]]] [!=] [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [3]]]]]]]]]]] ()
        ]]]]]]]]]]]] [\n]]]]]'

    pixel_tree = '[program [statement_list [statement_list [
        statement_list [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable [@] [color
        ]]] [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [
        variable_expression [variable [@] [image1]]] [\\[] [parameters [
        parameters [primary_expression [1]]] [,] [primary_expression
        [1]]] [\\]]]]]]]]]]]]]] [\n]]]] [statement [
        expression_statement [expression [assignment_expression [
        logical_OR_expression [logical_AND_expression [
        equality_expression [relational_expression [additive_expression
        [multiplicative_expression [logical_NOT_expression [
        intermediate_expression [function_expression [#] [grayscale] [
        parameters [primary_expression [variable_access_expression [
        variable_expression [variable [@] [image1]]]]]]]]]]]]]]] [\n
        ]]]] [statement [expression_statement [expression [
        assignment_expression [variable_expression [variable_expression
        [variable [@] [image1]]] [\\[] [parameters [parameters [
        primary_expression [1]]] [,] [primary_expression [1]]] [\\]]]
        [=] [assignment_expression [logical_OR_expression [
        logical_AND_expression [equality_expression [
        relational_expression [additive_expression [
        multiplicative_expression [logical_NOT_expression [
        intermediate_expression [primary_expression [
        variable_access_expression [variable_expression [variable [@] [
        color]]]]]]]]]]]]]]] [\n]]]]]'

    forp_tree = '[program [statement_list [statement [
        iteration_statement [forp] [variable [@] [pixel]] [in] [variable
```

```
      [@] [image1]] [block [:] [\n] [INDENT] [statement_list [
    statement_list [statement [expression_statement [expression [
    assignment_expression [variable_expression [variable [@] [color
    ]]] [=] [assignment_expression [logical_OR_expression [
    logical_AND_expression [equality_expression [
    relational_expression [additive_expression [
    multiplicative_expression [logical_NOT_expression [
    intermediate_expression [primary_expression [
    variable_access_expression [variable_access_expression [
    variable_expression [variable [@] [pixel]]]] [.] [color
    ]]]]]]]]]]]]] [\n]]]] [statement [expression_statement [
    expression [assignment_expression [variable_expression [variable
     [@] [pixel]]] [=] [assignment_expression [logical_OR_expression
     [logical_AND_expression [equality_expression [
    relational_expression [additive_expression [
    multiplicative_expression [logical_NOT_expression [
    intermediate_expression [primary_expression [
    variable_access_expression [variable_expression [variable [@] [
    color]]]]]]]]]]]]]]] [\n]]]] [DEDENT]]]]]]'

'''
python strings
'''

oneliner_python = ('fixelFunctions.grayscale(ns.image1)\n', '')

indent_python = ('''\
if ns.hey < not 1:
  sup()
hey(ns.image1)
for ns.image in ns.images:
  if ns.image == ns.image1:
    hey(ns.image)
''', '''\
def hey(myImage):
  if myImage.width > 100:
    if myImage.height > 50 and myImage.height < 60:
      x = 10
      y = 50
  else:
    x = 100
    y = 20
  return x
''')

function_def_python = ('''\
sup(ns.image1, ns.image2)
''', '''\
def sup(myImage1, myImage2):
  return myImage1.height + myImage2.height
''')

or_and_python = ('''\
True and False or ns.str == "coolstring"
True and (False or ns.str == "coolstring")
''', '')
```

```
not_python = (’’’\
ns.myList = [40, 500]
not 100 <= ns.myList[1]
’’’, ’’)

selection_if_python = (’’’\
if ns.x < 1:
  ns.y = 5
’’’, ’’)

selection_ifelse_python = (’’’\
if ns.x > 1 and ns.y >= 1:
  ns.y = ns.y - 5
else:
  ns.y = 3
’’’, ’’)

equality_notequal_python = (’’’\
ns.test = (5 != 3)
’’’, ’’)

multiplicative_python = (’’’\
ns.x = ns.y * 5
ns.y = ns.x / 5
’’’, ’’)

iteration_for_python = (’’’\
for ns.image in ns.images:
  ns.x = ns.x + 1
’’’, ’’)

color_setting_python = (’’’\
red = color(0,0,0)
image1[302,19] = red
’’’, ’’)

pixel_python = (’’’\
ns.color = ns.image1[1, 1]
fixelFunctions.grayscale(ns.image1)
ns.image1[1, 1] = ns.color
’’’, ’’)

forp_python = (’’’\
ns.pixel = runtime_classes.Pixel(ns.image1)
for ns.pixel.x in xrange(0, ns.image1.width):
  for ns.pixel.y in xrange(0, ns.image1.height):
    ns.color = ns.pixel.color
    ns.image1[ns.pixel.x, ns.pixel.y] = ns.color
’’’, ’’)

complex_python = (’’’\
ghost(ns.image1)
my_collage(ns.image1, ns.image2, ns.image3, ns.image4)
’’’, ’’’\
def ghost(image):
  fixelFunctions.blur(image)
  fixelFunctions.grayscale(image)
```

```
  def my_collage(image1, image2, image3, image4):
    my_list = [image1, image2, image3, image4]
    for image in my_list:
      if image == image1 or image == image2:
        fixelFunctions.sharpen(image)
      else:
        fixelFunctions.caption(image, "pretty colors")
''')

big_python = ('''\
ns.l1 = [ns.image0, ns.image1, ns.image2]
ns.i2 = [ns.image3, ns.image4, ns.image5]
for ns.image in ns.l1:
  invert(ns.image)
for ns.image in ns.l2:
  my_blur(ns.image)
''','''\
def invert(image):
  p = runtime_classes.Pixel(image)
  for p.x in xrange(0, image.width):
    for p.y in xrange(0, image.height):
      image[p.x, p.y] = 255 - p.color
def my_blur(image):
  p1 = runtime_classes.Pixel(image)
  for p1.x in xrange(0, image.width):
    for p1.y in xrange(0, image.height):
      p2 = runtime_classes.Pixel(image)
      for p2.x in xrange(0, image.width):
        for p2.y in xrange(0, image.height):
          if p2.y - p1.y < 5:
            image[p1.x, p1.y] = (p1.color + p2.color) / 2
''')
```

```
import mock
from translator import lexer
from translator import parser


def get_mock_token(type, value):
    mock_token = mock.MagicMock()
    mock_token.type = type
    mock_token.value = value
    return mock_token


def tree_to_string(tree):
  s = '['
  if tree.value == '[' or tree.value == ']':
    s += '\\'
  s += tree.value
  if hasattr(tree, 'children'):
    for child in tree.children:
      s += ' ' + tree_to_string(child)
  s += ']'
  return s
```

```
def tree_string_from_source(source_string):
    """
    runs lexer/parser on source string and outputs tree string from
        the resulting AST.
    """
    my_lex = lexer.get_lex()
    my_yacc = parser.get_yacc()

    tree = my_yacc.parse(source_string, lexer=my_lex)
    tree_string = tree_to_string(tree)

    print('pretty format:')
    print(tree)
    print('\ntree string:')
    print(repr(tree_string))


def token_stream_from_source(source_string):
    my_lex = lexer.get_lex()
    my_lex.input(source_string)
    token_stream = []
    token = my_lex.token()
    while token is not None:
        token_stream.append((token.type, token.value))
        token = my_lex.token()
    print(repr(token_stream))


def tree_string_pretty_version(tree_string):
    """
    For debugging, meant for human to view tree in easier to read
        format
    """
    print('input tree string:\n')
    print(repr(tree_string) + '\n')
    no_spaces_tree_string = tree_string.replace(' ', '')
    tree, final_index = string_to_tree(no_spaces_tree_string, 0)

    if final_index < len(no_spaces_tree_string)-1:
        raise ValueError('bad format, matched open parentheses
            before reaching end of string')
    elif final_index >= len(no_spaces_tree_string):
        raise ValueError('looks like Matt screwed up the string to
            tree function')

    print('pretty format:\n')
    print(tree)


def string_to_tree(tree_string, index):
  """
  builds node for current level and creates children recursively
  must be given string with no whitespace.
  """

  # child starts with ( character
  if tree_string[index] != '[':
```

```
    raise ValueError('invalid tree string format, each subtree must
        start with \"[\"')
  index += 1

  children = []
  node_name = ''
  i = index
  while i < len(tree_string):
    char = tree_string[i]
    if char == '[':  # new child
      child, new_index = string_to_tree(tree_string, i)
      children.append(child)
      i = new_index
    elif char == ']':  # end of this tree, return
      node = parser.Node(node_name)
      node.children = tuple(children)  # passing tuple to
          constructor results in nested tuple
      return node, i  # return node and current place in iteration
    elif char == '\\' and (tree_string[i+1] == '[' or tree_string[i
        +1] == ']'):
      i += 1  # skip past \ character
      node_name += tree_string[i]
    else:  # character is part of node name
      node_name += char

    i += 1  # increment to next character

  raise ValueError('bad format, reached end of string with without
      balancing all open parentheses')
```