Programing Languages Final Project Presentation

By Kayne Khoury, Brian Bargas, German Martinez

Option A

First Section

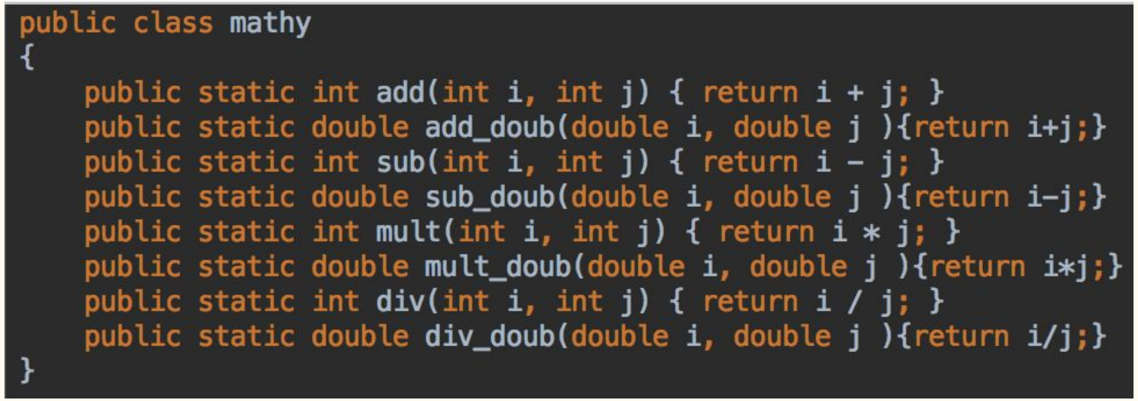
Parts: 1 - 6

1. n/a
2. Using Exec Fuction
3. Python Closures
4. Stream Operations
5. Lambda + List Comprehension
6. Overcoming Obstacles

Terminal Commands:

```
cd
Documents/Academic/CompSci/ProgLanguages/Intellij/Assigments/PLFinalProject/PLPr
oject7

python mini-lisp.py
```

2. Use the Exec Function



```
public class mathy
{
    public static int add(int i, int j) { return i + j; }
    public static double add_doub(double i, double j ){return i+j;}
    public static int sub(int i, int j) { return i - j; }
    public static double sub_doub(double i, double j ){return i-j;}
    public static int mult(int i, int j) { return i * j; }
    public static double mult_doub(double i, double j ){return i*j;}
    public static int div(int i, int j) { return i / j; }
    public static double div_doub(double i, double j ){return i/j;}
}
```

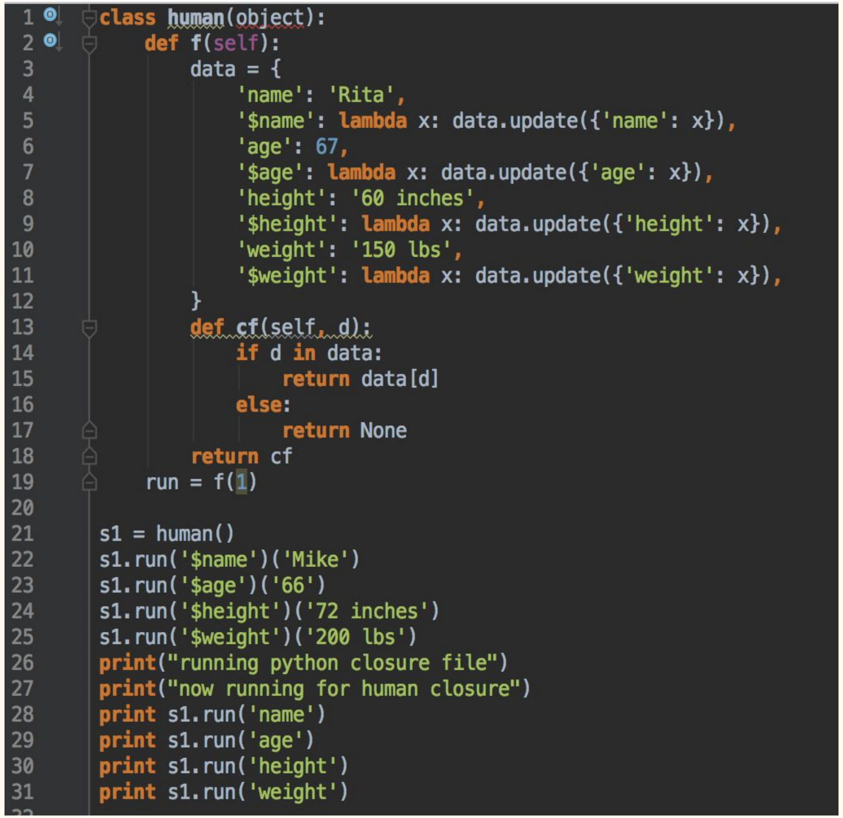
Files used:

- **Mathy.java**
- **mini-lisp.py**

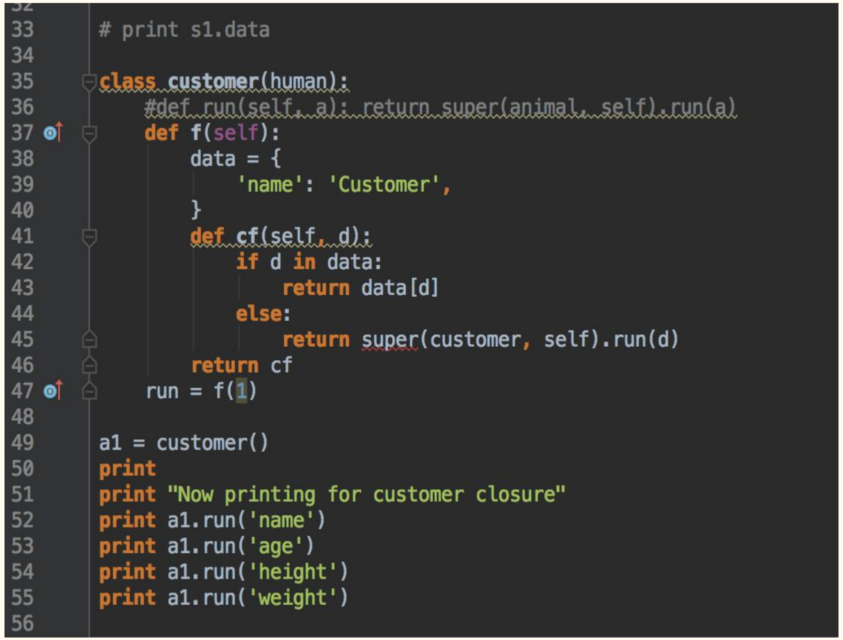
1. (exec 'import mathy; toReturn = mathy.div_doub(23.0, 34.0)')
2. (exec 'import mathy; toReturn = mathy.sub_doub(23.0, 34.0)')
3. (exec 'import mathy; toReturn = mathy.div(30, 3)')
4. (exec 'import mathy; toReturn = mathy.mult(23, 34)')
5. (exec 'import mathy; toReturn = mathy.sub(23, 34)')

3. Use the Python Closures

1. (exec 'import human;')



```python
class human(object):
    def f(self):
        data = {
            'name': 'Rita',
            '$name': lambda x: data.update({'name': x}),
            'age': 67,
            '$age': lambda x: data.update({'age': x}),
            'height': '60 inches',
            '$height': lambda x: data.update({'height': x}),
            'weight': '150 lbs',
            '$weight': lambda x: data.update({'weight': x}),
        }
        def cf(self, d):
            if d in data:
                return data[d]
            else:
                return None
        return cf
    run = f(1)

s1 = human()
s1.run('$name')('Mike')
s1.run('$age')('66')
s1.run('$height')('72 inches')
s1.run('$weight')('200 lbs')
print("running python closure file")
print("now running for human closure")
print s1.run('name')
print s1.run('age')
print s1.run('height')
print s1.run('weight')
```



```python
# print s1.data

class customer(human):
    #def run(self, a): return super(animal, self).run(a)
    def f(self):
        data = {
            'name': 'Customer',
        }
        def cf(self, d):
            if d in data:
                return data[d]
            else:
                return super(customer, self).run(d)
        return cf
    run = f(1)

a1 = customer()
print
print "Now printing for customer closure"
print a1.run('name')
print a1.run('age')
print a1.run('height')
print a1.run('weight')
```

Files used:

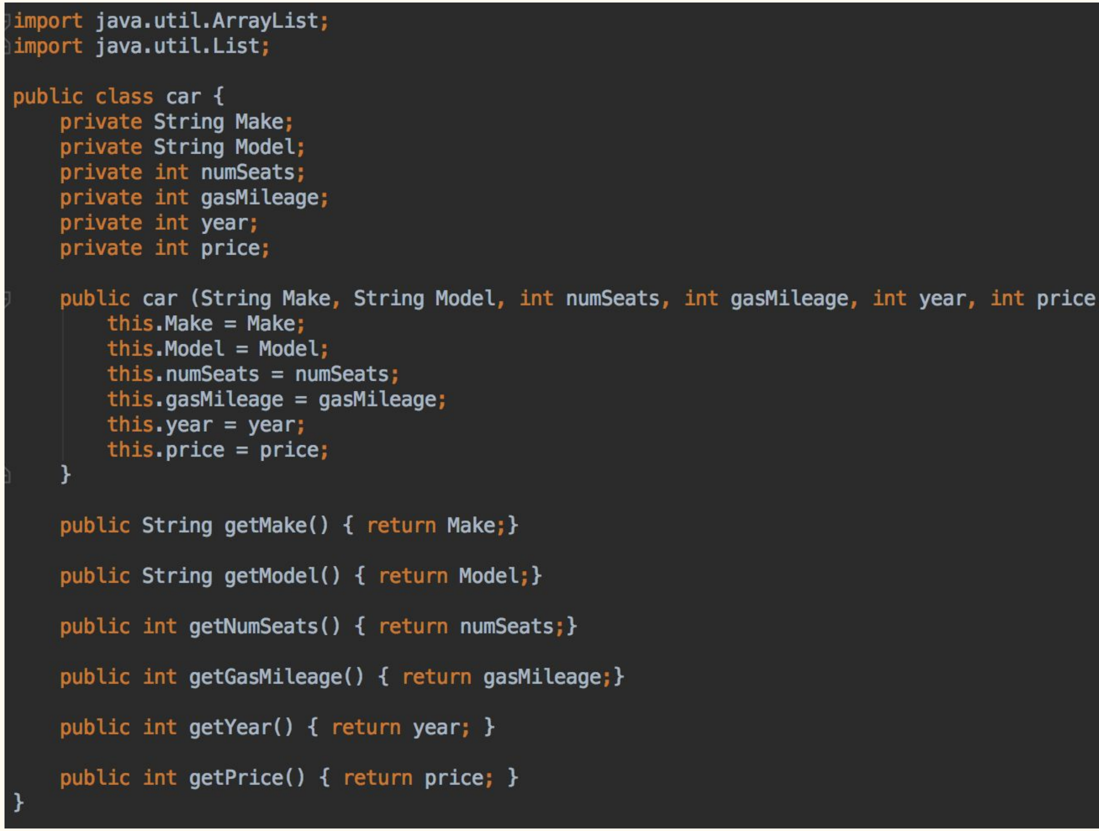
- **human.java**
- **mini-lisp.py**

4. Stream Operations

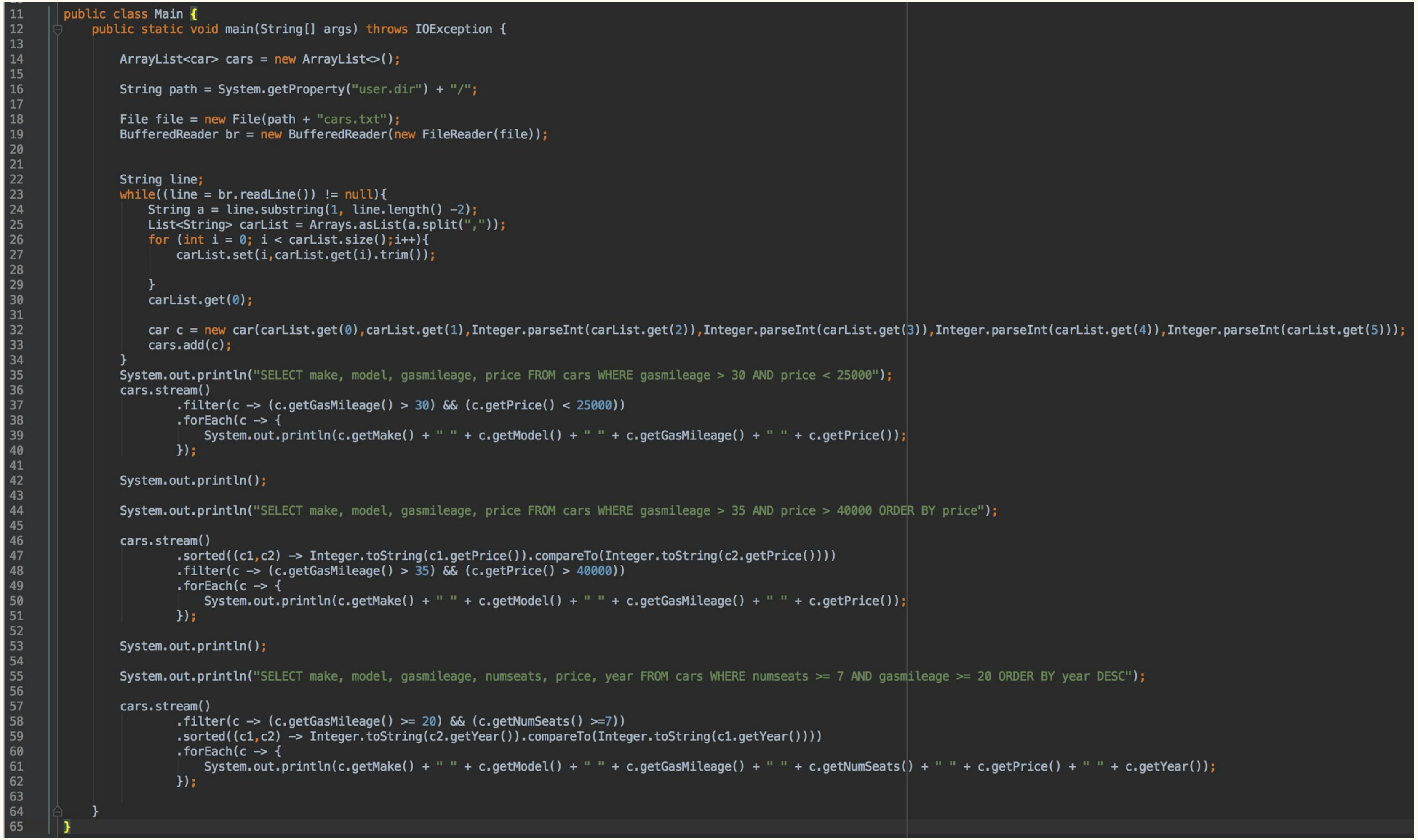
1. (exec 'import javarun')

Files used:

- Car.java
- Main.jave
- Cars.txt
- Javarun.py
- Mini-lisp.py



```java
import java.util.ArrayList;
import java.util.List;

public class car {
    private String Make;
    private String Model;
    private int numSeats;
    private int gasMileage;
    private int year;
    private int price;

    public car (String Make, String Model, int numSeats, int gasMileage, int year, int price
        this.Make = Make;
        this.Model = Model;
        this.numSeats = numSeats;
        this.gasMileage = gasMileage;
        this.year = year;
        this.price = price;
    }

    public String getMake() { return Make;}

    public String getModel() { return Model;}

    public int getNumSeats() { return numSeats;}

    public int getGasMileage() { return gasMileage;}

    public int getYear() { return year; }

    public int getPrice() { return price; }
}
```



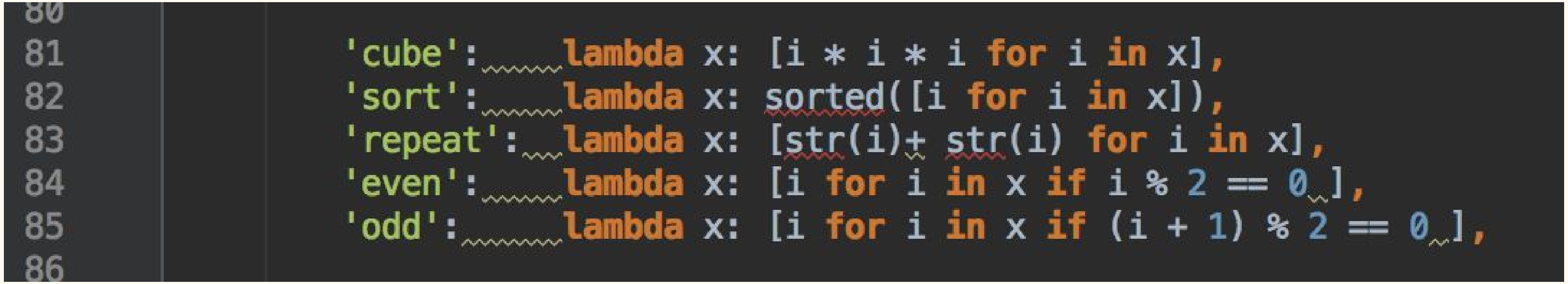
```java
public class Main {
    public static void main(String[] args) throws IOException {

        ArrayList<car> cars = new ArrayList<>();

        String path = System.getProperty("user.dir") + "/";

        File file = new File(path + "cars.txt");
        BufferedReader br = new BufferedReader(new FileReader(file));


        String line;
        while((line = br.readLine()) != null){
            String a = line.substring(1, line.length() -2);
            List<String> carList = Arrays.asList(a.split(","));
            for (int i = 0; i < carList.size();i++){
                carList.set(i,carList.get(i).trim());

            }
            carList.get(0);

            car c = new car(carList.get(0),carList.get(1),Integer.parseInt(carList.get(2)),Integer.parseInt(carList.get(3)),Integer.parseInt(carList.get(4)),Integer.parseInt(carList.get(5)));
            cars.add(c);
        }
        System.out.println("SELECT make, model, gasmileage, price FROM cars WHERE gasmileage > 30 AND price < 25000");
        cars.stream()
                .filter(c -> (c.getGasMileage() > 30) && (c.getPrice() < 25000))
                .forEach(c -> {
                    System.out.println(c.getMake() + " " + c.getModel() + " " + c.getGasMileage() + " " + c.getPrice());
                });

        System.out.println();

        System.out.println("SELECT make, model, gasmileage, price FROM cars WHERE gasmileage > 35 AND price > 40000 ORDER BY price");

        cars.stream()
                .sorted((c1,c2) -> Integer.toString(c1.getPrice()).compareTo(Integer.toString(c2.getPrice())))
                .filter(c -> (c.getGasMileage() > 35) && (c.getPrice() > 40000))
                .forEach(c -> {
                    System.out.println(c.getMake() + " " + c.getModel() + " " + c.getGasMileage() + " " + c.getPrice());
                });

        System.out.println();

        System.out.println("SELECT make, model, gasmileage, numseats, price, year FROM cars WHERE numseats >= 7 AND gasmileage >= 20 ORDER BY year DESC");

        cars.stream()
                .filter(c -> (c.getGasMileage() >= 20) && (c.getNumSeats() >=7))
                .sorted((c1,c2) -> Integer.toString(c2.getYear()).compareTo(Integer.toString(c1.getYear())))
                .forEach(c -> {
                    System.out.println(c.getMake() + " " + c.getModel() + " " + c.getGasMileage() + " " + c.getNumSeats() + " " + c.getPrice() + " " + c.getYear());
                });

    }
}
```

5. Lambda + List Comprehension

Files used:

- **lis.py**
- **mini-lisp.py**

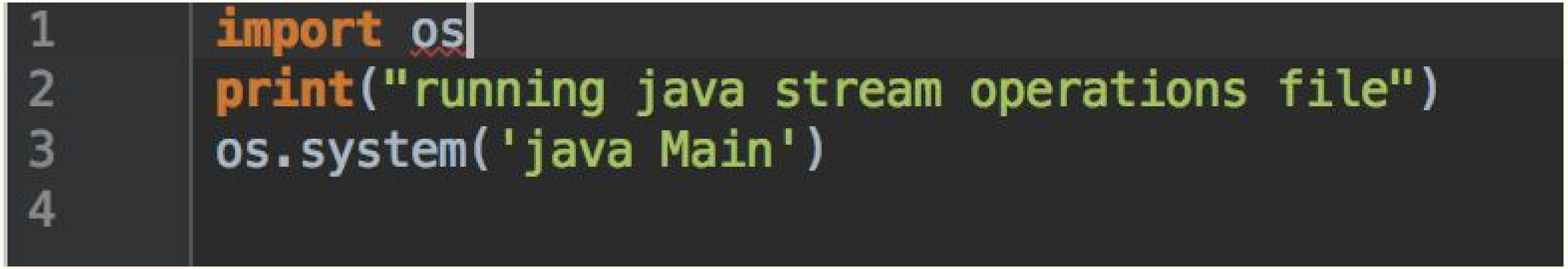
1. (cube '(1 2 3))
2. (sort '(48 3 64 12))
3. (repeat '(buffalo 22 34 swigity))
4. (odd '(1 2 3 4 5 6))
5. (even '(1 2 3 4 5 6))



```
        'cube':    lambda x: [i * i * i for i in x],
        'sort':    lambda x: sorted([i for i in x]),
        'repeat':  lambda x: [str(i)+ str(i) for i in x],
        'even':    lambda x: [i for i in x if i % 2 == 0 ],
        'odd':     lambda x: [i for i in x if (i + 1) % 2 == 0 ],
```

6. Overcoming Issues

In problem 4, stream operations we had trouble because exec wasn't running the java file we created a python file that handled running the java file. Inside the file we used the "os" module to run the java command.



```
import os
print("running java stream operations file")
os.system('java Main')

```

Files used:

- **Javarun.py**

7. Swift Implementacion

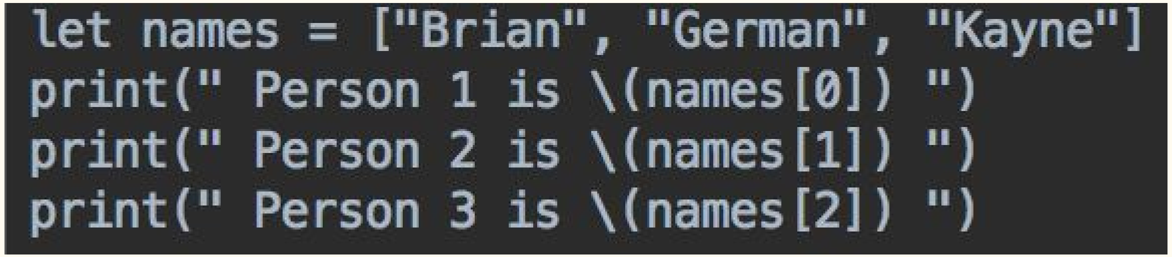
Swift Implementacion

Instead of using eval we created our own interpreter.

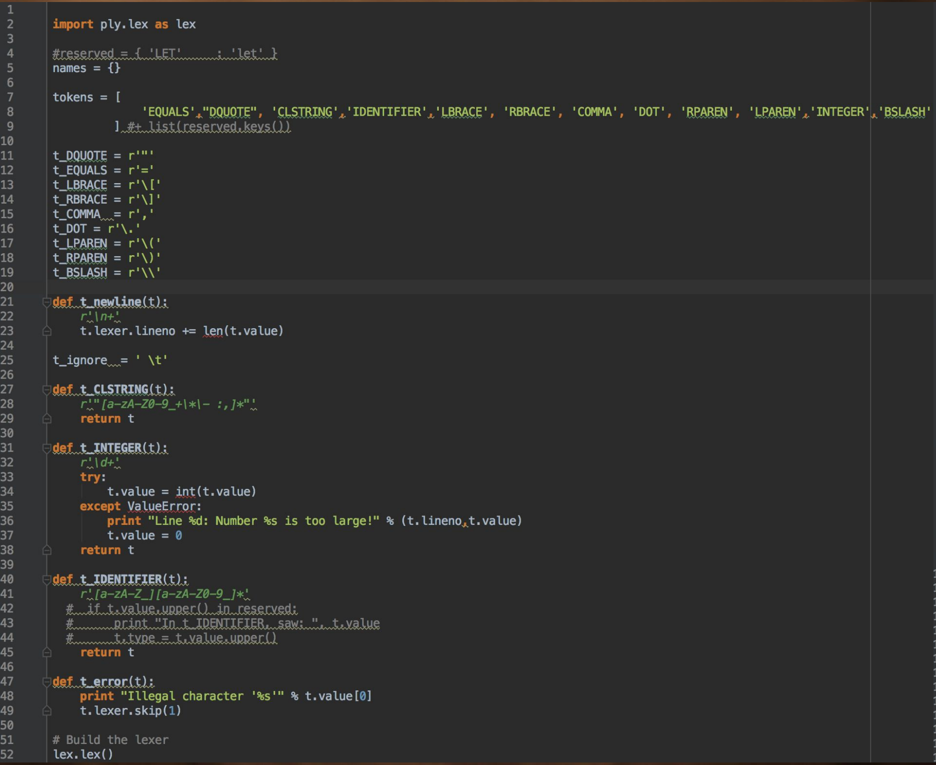
We were having trouble parsing lines in one file. Each line in separated into its own file and parsed.

Files used:

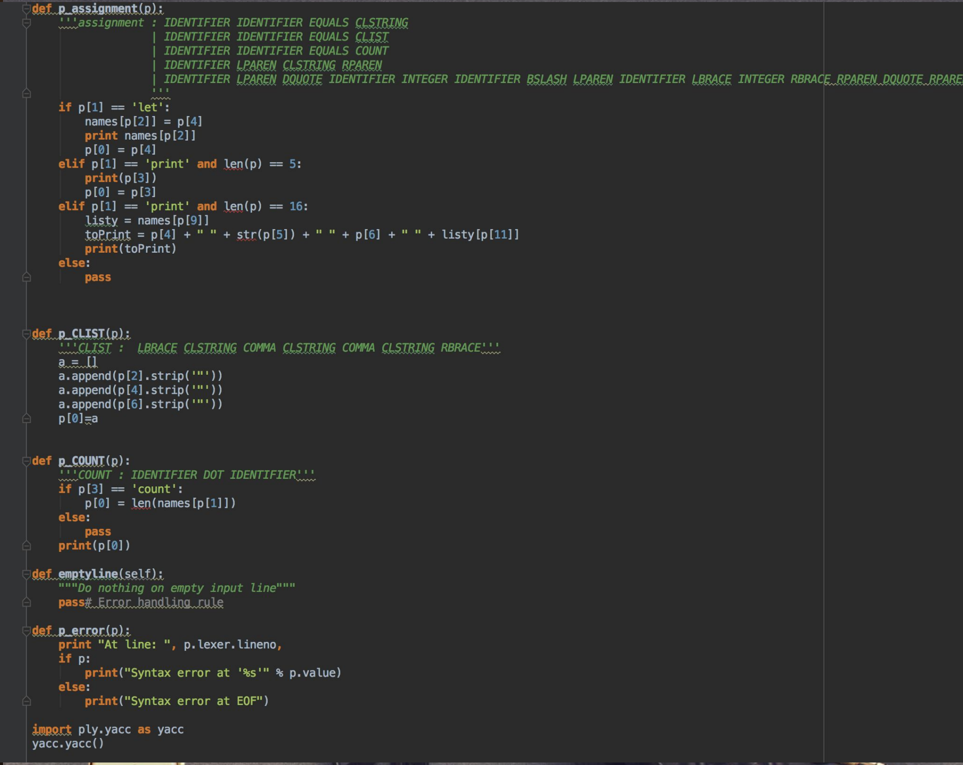
- **kgb2.py**
- **testkgb.c**



```
let names = ["Brian", "German", "Kayne"]
print(" Person 1 is \(names[0]) ")
print(" Person 2 is \(names[1]) ")
print(" Person 3 is \(names[2]) ")
```



```python

import ply.lex as lex

#reserved = { 'LET'      : 'let' }
names = {}

tokens = [
             'EQUALS',"DQUOTE", 'CLSTRING','IDENTIFIER','LBRACE', 'RBRACE', 'COMMA', 'DOT', 'RPAREN', 'LPAREN','INTEGER','BSLASH'
        ] #+ list(reserved.keys())

t_DQUOTE = r'"'
t_EQUALS = r'='
t_LBRACE = r'\['
t_RBRACE = r'\]'
t_COMMA  = r','
t_DOT = r'\.'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_BSLASH = r'\\'

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

t_ignore  = ' \t'

def t_CLSTRING(t):
    r'"[a-zA-Z0-9_+|*|- :,]*"'
    return t

def t_INTEGER(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print "Line %d: Number %s is too large!" % (t.lineno,t.value)
        t.value = 0
    return t

def t_IDENTIFIER(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
  #  if t.value.upper() in reserved:
  #       print "In t_IDENTIFIER, saw: ", t.value
  #       t.type = t.value.upper()
    return t

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

# Build the lexer
lex.lex()
```



```python
def p_assignment(p):
    '''assignment : IDENTIFIER IDENTIFIER EQUALS CLSTRING
                  | IDENTIFIER IDENTIFIER EQUALS CLIST
                  | IDENTIFIER IDENTIFIER EQUALS COUNT
                  | IDENTIFIER LPAREN CLSTRING RPAREN
                  | IDENTIFIER LPAREN DQUOTE IDENTIFIER INTEGER IDENTIFIER BSLASH LPAREN IDENTIFIER LBRACE INTEGER RBRACE RPAREN DQUOTE RPARE
                  '''
    if p[1] == 'let':
        names[p[2]] = p[4]
        print names[p[2]]
        p[0] = p[4]
    elif p[1] == 'print' and len(p) == 5:
        print(p[3])
        p[0] = p[3]
    elif p[1] == 'print' and len(p) == 16:
        listy = names[p[9]]
        toPrint = p[4] + " " + str(p[5]) + " " + p[6] + " " + listy[p[11]]
        print(toPrint)
    else:
        pass



def p_CLIST(p):
    '''CLIST :  LBRACE CLSTRING COMMA CLSTRING COMMA CLSTRING RBRACE'''
    a = []
    a.append(p[2].strip('"'))
    a.append(p[4].strip('"'))
    a.append(p[6].strip('"'))
    p[0]=a


def p_COUNT(p):
    '''COUNT : IDENTIFIER DOT IDENTIFIER'''
    if p[3] == 'count':
        p[0] = len(names[p[1]])
    else:
        pass
    print(p[0])

def emptyline(self):
    """Do nothing on empty input line"""
    pass# Error handling rule

def p_error(p):
    print "At line: ", p.lexer.lineno,
    if p:
        print("Syntax error at '%s'" % p.value)
    else:
        print("Syntax error at EOF")

import ply.yacc as yacc
yacc.yacc()

```

kgb2.py

Conclusion

Github Link:
https://github.com/germanmtz93/PLFinalProject.git

What we learned in this class:

Java syntax- static methods, public v private, classes

Functional, and Prototype programming - Python, Javascript

Lambdas - python map and lambda, list comprehension, java stream operations

Lex and Yacc - fundamentals of building a language, Godels incompleteness theorem, PLY, incorporating java and list comprehension together using Jython, building parsers and interpreters