CS 329E Final Project

Benjamin Liu
Wilson Lu

Python Closures

```
1   class calculator(object):
2       def arithmetic(self):
3           operations = {
4               'add': lambda x, y: x+y,
5               'sub': lambda x, y: x-y,
6               'mul': lambda x, y: x*y,
7               'div': lambda x, y: x/y
8               }
9           def calculate(self, op):
10              if op in operations:
11                  return operations[op]
12              else:
13                  return None
14          return calculate
15      run = arithmetic(1)
16
17  c1 = calculator()
18  print(c1.run('add')(1, 2))
```

```
21  import math
22  class scientific_calculator(calculator):
23
24      def arithmetic(self):
25          operations = {
26              'sin' : lambda x: math.sin(x),
27              'cos' : lambda x: math.cos(x),
28              'tan' : lambda x: math.tan(x),
29              'pow' : lambda x,y: x**y
30              }
31          def calculate(self, op):
32              if op in operations:
33                  return operations[op]
34              else:
35                  return super(calculator,self).run(op)
36          return calculate
37      run = arithmetic(1)
38
```

- (exec 'from Calculator import calculator; toReturn = calculator().run("add")(1, 2)')
- (exec 'from Calculator import scientific_calculator; toReturn = scientific_calculator().run("cos")(3.1416)')
- (exec 'from Calculator import scientific_calculator; toReturn = scientific_calculator().run("div")(144.0, 11.0)')

Java Stream Operations

Class: Bible.java

```java
public class Bible {
    private String id;
    private String name;
    private String author;
    private String chapters;
    private String verses;


    public Bible (String id, String name, String author, String chapters, String verses) {
        this.id = id;
        this.name = name;
        this.author = author;
        this.chapters = chapters;
        this.verses = verses;
    }

    public String getId() { return id; }

    public String getName() { return name; }

    public String getAuthor() { return author; }

    public String getChapters() { return chapters; }

    public String getVerses() { return verses; }
}
```

bible.txt

```
(1, 'Genesis', 'Moses', 50, 1533)
(2, 'Exodus', 'Moses', 40, 1213)
(3, 'Leviticus', 'Moses', 27, 859)
(4, 'Numbers', 'Moses', 36, 1288)
(5, 'Deuteronomy', 'Moses', 34, 959)
(6, 'Joshua', 'Joshua', 24, 658)
(7, 'Judges', 'Unknown', 21, 618)
(8, 'Ruth', 'Unknown', 4, 85)
(9, '1 Samuel', 'Samuel', 31, 810)
(10, '2 Samuel', 'Samuel', 24, 695)
(11, '1 Kings', 'Jeremiah', 22, 816)
(12, '2 Kings', 'Jeremiah', 25, 719)
(13, '1 Chronicles', 'Ezra', 29, 942)
(14, '2 Chronicles', 'Ezra', 36, 822)
(15, 'Ezra', 'Ezra', 10, 280)
(16, 'Nehemiah', 'Nehemiah', 13, 406)
(17, 'Esther', 'Unknown', 10, 167)
```

Java Stream Operations: ListComprehension

```
1   import ...
8
9   public class ListComprehension {
10
11      // SELECT * FROM bib;
12      public static ArrayList<String> selectAll() throws IOException {
13          ArrayList<Bible> bib = new ArrayList<>();
14
15          String path = System.getProperty("user.dir");
16          File file = new File(path + "/bible.txt");
17          BufferedReader br = new BufferedReader(new FileReader(file));
18
19          String line;
20          while ((line = br.readLine()) != null) {
21              String substr = line.substring(1, line.length() - 1);
22              List<String> bibList = Arrays.asList(substr.split(","));
23              for (int i = 0; i < bibList.size(); i++) {
24                  bibList.set(i, bibList.get(i).trim());
25              }
26              Bible b = new Bible(bibList.get(0), bibList.get(1), bibList.get(2), bibList.get(3), bibList.get(4));
27              bib.add(b);
28          }
29
30          ArrayList<String> result = new ArrayList<>();
31
32          bib.stream()
33                  .forEach(b -> {
34                      String row = b.getId() + " | " + b.getName() + " | " + b.getAuthor() + " | " + b.getChapters() + " | " + b.getVerses();
35                      result.add(row);
36                  });
37
38          return result;
39      }
40
```

(exec 'import ListComprehension; toReturn = ListComprehension.selectAll()')

Java Stream Operations: ListComprehension

```
41      // SELECT name FROM bib ORDER BY name;
42      public static ArrayList<String> Alphabetical() throws IOException {
43          ArrayList<Bible> bib = new ArrayList<>();
44
45          String path = System.getProperty("user.dir");
46          File file = new File(path + "/bible.txt");
47          BufferedReader br = new BufferedReader(new FileReader(file));
48
49          String line;
50          while ((line = br.readLine()) != null) {
51              String substr = line.substring(1, line.length() - 1);
52              List<String> bibList = Arrays.asList(substr.split(","));
53              for (int i = 0; i < bibList.size(); i++) {
54                  bibList.set(i, bibList.get(i).trim());
55              }
56              Bible b = new Bible(bibList.get(0), bibList.get(1), bibList.get(2), bibList.get(3), bibList.get(4));
57              bib.add(b);
58          }
59
60          ArrayList<String> result = new ArrayList<>();
61
62          bib.stream()
63                  .sorted((b1, b2) -> b1.getName().compareTo(b2.getName()))
64                  .forEach(b -> {
65                      String row = b.getName();
66                      result.add(row);
67                  });
68
69          return result;
70      }
71
```

(exec 'import ListComprehension; toReturn = ListComprehension.Alphabetical()')

Java Stream Operations: ListComprehension

```java
// SELECT name, chapters FROM bib WHERE chapters </> num;
public static ArrayList<String> greaterThanChapters(int num, boolean tf) throws IOException {
    ArrayList<Bible> bib = new ArrayList<>();

    String path = System.getProperty("user.dir");
    File file = new File(path + "/bible.txt");
    BufferedReader br = new BufferedReader(new FileReader(file));

    String line;
    while ((line = br.readLine()) != null) {
        String substr = line.substring(1, line.length() - 1);
        List<String> bibList = Arrays.asList(substr.split(","));
        for (int i = 0; i < bibList.size(); i++) {
            bibList.set(i, bibList.get(i).trim());
        }
        Bible b = new Bible(bibList.get(0), bibList.get(1), bibList.get(2), bibList.get(3), bibList.get(4));
        bib.add(b);
    }

    ArrayList<String> result = new ArrayList<>();

    if (tf) {
        bib.stream()
                .filter(b -> (Integer.parseInt(b.getChapters()) > num))
                .forEach(b -> {
                    String row = b.getName() + " | " + b.getChapters();
                    result.add(row);
                });
    } else {
        bib.stream()
                .filter(b -> (Integer.parseInt(b.getChapters()) < num))
                .forEach(b -> {
                    String row = b.getName() + " | " + b.getChapters();
                    result.add(row);
```

(exec 'import ListComprehension; toReturn = ListComprehension.greaterThanChapters(25, 1)')

Java Stream Operations: ListComprehension

```
112     // SELECT author, count(chapter) FROM bib GROUP BY author
113     public static ArrayList<String> countAuthorChapters() throws IOException {
114         ArrayList<Bible> bib = new ArrayList<>();
115 
116         String path = System.getProperty("user.dir");
117         File file = new File(path + "/bible.txt");
118         BufferedReader br = new BufferedReader(new FileReader(file));
119 
120         String line;
121         while ((line = br.readLine()) != null) {
122             String substr = line.substring(1, line.length() - 1);
123             List<String> bibList = Arrays.asList(substr.split(","));
124             for (int i = 0; i < bibList.size(); i++) {
125                 bibList.set(i, bibList.get(i).trim());
126             }
127             Bible b = new Bible(bibList.get(0), bibList.get(1), bibList.get(2), bibList.get(3), bibList.get(4));
128             bib.add(b);
129         }
130 
131         ArrayList<String> result = new ArrayList<>();
132 
133         bib.stream()
134                 .collect(Collectors.groupingBy(Bible::getAuthor))
135                 .entrySet()
136                 .stream()
137                 .map(a -> a.getValue())
138                 .forEach (bibList -> {
139                     int total = bibList
140                             .stream()
141                             .mapToInt(b -> Integer.parseInt(b.getChapters()))
142                             .sum();
143                     String author = bibList.get(0).getAuthor();
144                     String row = author + " | " + total;
145                     result.add(row);
```

(exec 'import ListComprehension; toReturn = ListComprehension.countAuthorChapters()')

Python Lambda

```
def standard_env():
    "An environment with some Scheme standard procedures."
    import math, operator as op
    env = Env()
    env.update(vars(math)) # sin, cos, sqrt, pi, ...
    env.update({
        '+':lambda *x: sum(x), '-':op.sub, '*': lambda *x: reduce(op.mul, x), '/':op.div,
        '>':op.gt, '<':op.lt, '>=':op.ge, '<=':op.le, '=':op.eq,
        'abs':     abs,
        'append':  op.add,
        'apply':   apply,
        'begin':   lambda *x: x[-1],
        'car':     lambda x: x[0],
        'cdr':     lambda x: x[1:],
        'cons':    lambda x,y: [x] + y,
        'eq?':     op.is_,
        'equal?':  op.eq,
        'evens':   lambda x: [i for i in x if i%2==0],
        'length':  len,
        'list':    lambda *x: list(x),
        'list?':   lambda x: isinstance(x,list),
        'exec':    lambda x: eval(compile(x,'None','single')),
        'map':     map,
        'mapp':    lambda x, y: map(x, y),
        'max':     max,
        'min':     min,
        'not':     op.not_,
        'null?':   lambda x: x == [],
        'number?': lambda x: isinstance(x, Number),
        'odds':    lambda x: [i for i in x if i%2==1],
        'procedure?': callable,
        'round':   round,
        'symbol?': lambda x: isinstance(x, Symbol),
    })
```

(evens '(1 2 3 4 5 6 7))

(odds '(1 2 3 4 5 6 7))

(mapp (lambda x (+ x 1)) '(5 10 15))

Bugs & Improvements

```
def eval(x, env=global_env):
    "Evaluate an expression in an environment."
    if isinstance(x, Symbol):      # variable reference
        return env.find(x)[x]
    elif not isinstance(x, List):  # constant literal
        return x
    elif x[0] == 'quote':          # (quote exp)
        (_, exp) = x
        return exp
    elif x[0] == 'if':             # (if test conseq alt)
        (_, test, conseq, alt) = x
        exp = (conseq if eval(test, env) else alt)
        return eval(exp, env)
    elif x[0] == 'define':         # (define var exp)
        (_, var, exp) = x
        env[var] = eval(exp, env)
    elif x[0] == 'set!':           # (set! var exp)
        (_, var, exp) = x
        env.find(var)[var] = eval(exp, env)
    elif x[0] == 'lambda':         # (lambda (var...) body)
        (_, parms, body) = x
        return Procedure(parms, body, env)
    elif x[0] == 'exec':
        proc = eval(x[0], env)
        import re
        exec(proc(re.sub(r"^'|'$", '', x[1])))
        return toReturn
    else:                          # (proc arg...)
        proc = eval(x[0], env)
        try:
            args = [eval(exp, env) for exp in x[1:]]
        except TypeError:
            args = x[1::]
        return proc(*args)
```

```
def standard_env():
    "An environment with some Scheme standard procedures."
    import math, operator as op
    env = Env()
    env.update(vars(math)) # sin, cos, sqrt, pi, ...
    env.update({
        '+':lambda *x: sum(x), '-':op.sub, '*': lambda *x: reduce(op.mul, x), '/':op.div,
        '>':op.gt, '<':op.lt, '>=':op.ge, '<=':op.le, '=':op.eq,
```

(+ 1 2 3 4 5 6 7 8 9 10)

(* 2 3 5 7 11)

(evens (1 1 2 3 5 8 13))

(car (4 6 8 9 10))

Swift

Goal:

```swift
let apples = 3
let oranges = 5
let appleSummary = "I have \(apples) apples."
let fruitSummary = "I have \(apples + oranges) pieces of fruit."
```

Major Difficulty:

Calculating the \() inside the string

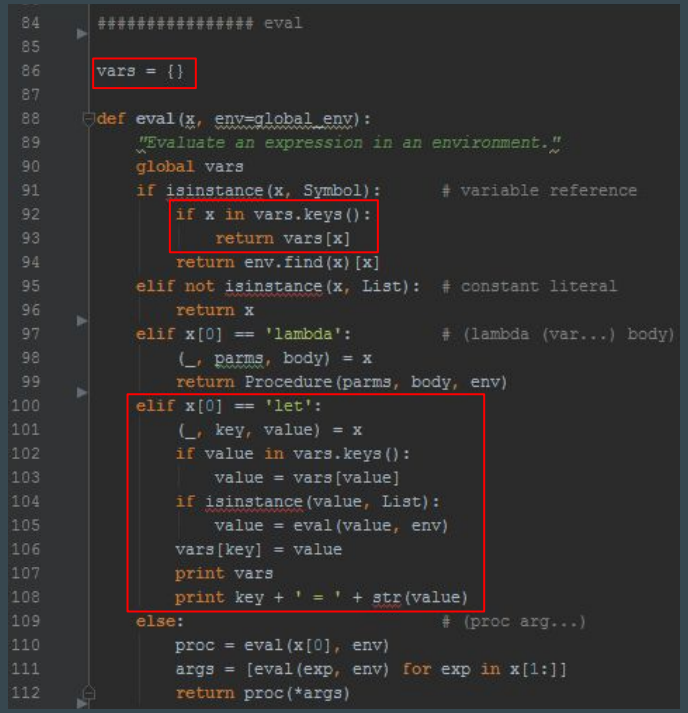
Lexer:

```python
import ply.lex as lex

# List of token names
tokens = ('LET', 'QUOTETEXT', 'SQUOTETEXT', 'INTEGER', 'VAR')

literals = ['=', '+', '/', '*', '-', '(', ')', '.']

# Reserved words
reserved = ['LET']

# Regular expression rules for simple tokens
t_QUOTETEXT = r'\".*?\"'
t_SQUOTETEXT = r'\'.*?\''


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_VAR(t):
    r'[A-Za-z_][A-Za-z0-9_]*'
    if t.value.upper() in reserved:
        t.type = t.value.upper()
    return t

# Define a rule so we can track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
```

Parser:

```python
# BNF

def p_strdeclaration(p):
    '''declaration : LET VAR "=" QUOTETEXT
                   | LET VAR "=" SQUOTETEXT
    '''
    p[0] = [p[1], p[2], p[4]]

def p_declaration(p):
    'declaration : LET VAR "=" expression'
    p[0] = [p[1], p[2], p[4]]

def p_math(p):
    ''' expression : expression "+" expression
                   | expression "-" expression
                   | expression "*" expression
                   | expression "/" expression
    '''
    p[0] = [p[2], p[1], p[3]]

def p_expression(p):
    '''expression : num
                  | VAR
    '''
    p[0] = p[1]

def p_num(p):
    '''num : INTEGER
           | float
    '''
    if type(p[1]) == float:
        p[0] = float(p[1])
    else:
        p[0] = int(p[1])
```

Swift



```
################ eval

vars = {}

def eval(x, env=global_env):
    "Evaluate an expression in an environment."
    global vars
    if isinstance(x, Symbol):      # variable reference
        if x in vars.keys():
            return vars[x]
        return env.find(x)[x]
    elif not isinstance(x, List):  # constant literal
        return x
    elif x[0] == 'lambda':         # (lambda (var...) body)
        (_, parms, body) = x
        return Procedure(parms, body, env)
    elif x[0] == 'let':
        (_, key, value) = x
        if value in vars.keys():
            value = vars[value]
        if isinstance(value, List):
            value = eval(value, env)
        vars[key] = value
        print vars
        print key + ' = ' + str(value)
    else:                          # (proc arg...)
        proc = eval(x[0], env)
        args = [eval(exp, env) for exp in x[1:]]
        return proc(*args)
```

let a = 1

let b = 3.14

let c = 5 * 1.2

let d = b - c

let e = "Hello World"