University of Virginia 7 November 2011
cs1120: Introduction of Computing
Explorations in Language, Logic, and Machines

Class 32: Interpreters

Upcoming (Remaining!) Schedule of Assignments

- **Now:** Problem Set 6 due
- **Friday, 11 November:** commitment for which PS8 you will do (web form)
- **Exam 2 Review Sessions** (Wednesday/Thursday, 16/17 November)
- **Wednesday, 16 November (11:59pm):** Problem Set 7 due (note extension from earlier deadline)
- **18 November:** finish reading Chapter 12
- **18 November:** Rice Hall Dedication
- **Monday, 21 November:** PS8, Preliminary Submission
- **Wednesday, 30 November:** Exam 2 due (will be handed out on **Monday, 21 November**)
- **Monday, 5 December (last class):** PS8, Final Submission due
- **Monday, 12 December (1:00pm):** Final Exam due

Interpreters

It is no exaggeration to regard this as the most fundamental idea in programming:

> *The evaluator, which determines the meaning of expressions in the programming language, is just another program.*

To appreciate this point is to change our images of ourselves as programmers. We come to see ourselves as designers of languages, rather than only users of languages designed by others.

Abelson and Sussman, *Structure and Interpretation of Computer Programs* (p. 360)

The evaluator takes as input an *expression* and *environment*, and outputs the value of that expression in the *environment*.

```
def meval(expr, env):
    if isPrimitive(expr):          return evalPrimitive(expr)
    elif isIf(expr):               return evalIf(expr, env)
    elif isDefinition(expr):       evalDefinition(expr, env)
    elif isName(expr):             return evalName(expr, env)
    elif isLambda(expr):           return evalLambda(expr, env)
    elif isApplication(expr):      return evalApplication(expr, env)
    else:                          error ('Unknown expression type: ' + str(expr))
```

Stateful Application Rule

To apply a constructed procedure:

1. **Construct a new environment**, whose parent is the environment of the applied procedure.
2. For each procedure parameter, create a place in the frame of the new environment with the name of the parameter. **Evaluate each operand expression in the environment of the application** and initialize the value in each place to the value of the corresponding operand expression.
3. **Evaluate the body of the procedure in the newly created environment.** The resulting value is the value of the application.

parse(*<string>*) → list representing the Charme code

```
parse("(define square (lambda (x) (* x x)))")[0] ==> ['define', 'square', ['lambda', ['x'], ['*', 'x', 'x']]]
```

How should we represent an *Environment*?

```
def evalApplication(expr, env):
  subexprvals = map (lambda sexpr: _____________________, expr)
  return __________(subexprvals[0], subexprvals[1:])
```

```
def mapply(proc, operands):
   if (isPrimitiveProcedure(proc)):
      return proc(operands)
   elif isinstance(proc, Procedure):
      params = proc.getParams()

      newenv = ____________________________________
      for i in range(0, len(params)):
           newenv.addVariable(params[i], operands[i])

      return ____________________________________
```