Class 32 - Interpreters

Schedule

To be eligible for automatic Red Belt promotion, you need to complete Project 6 (including a good answer to its bonus Problem 8) by **Friday, 22 April**, and have also completed Project 5 (including Problem 8). Otherwise, you may qualify for the Red Belt exam by completing Problems 1-7 of both Project 5 and Problem 6. If you do this by Friday, 22 April, you will be able to take the Red Belt exam starting Monday, 25 April (and have an opportunity to ensure at least an A- in the course before the end of the semester).

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*.

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

The Meta-Circular Evaluator

```
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))
```

Primitives

```
def evalPrimitive(expr):
    if isNumber(expr):
        return int(expr)
    else:
        return expr
```

Primitive Procedures

```
def primitivePlus (operands):
    if (len(operands) == 0):
       return 0
    else:
       return operands[0] + primitivePlus (operands[1:])
```

Application

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.

How should we represent an *environment*?

```
def evalApplication(expr, env):
    subexprvals = [meval(sexpr, env) for sexpr in expr]
    return mapply(_________________, _________________)

def mapply(proc, operands):
    if (isPrimitiveProcedure(proc)):
        return proc(operands)
    elif isinstance(proc, Procedure):
        params = proc.getParams()
        newenv = Environment(proc.getEnvironment())
        if len(params) != len(operands):
            evalError ('Parameter length mismatch: ...')
        for i in range(0, len(params)):
            newenv.addVariable(params[i], ___________)
        return meval(_______________, newenv)
    else:
        # error case
```

