CMSC 143: Object-Oriented Programming with Robots

Lab 13: Postfix-Python
Due December 8, 2016

In this lab, we will create a new programming language for controlling our robots. The language is similar to **Reverse Polish Notation** (RPN)[1]. In reverse polish notation, rather than putting mathematical operators between the operands (i.e. infix) we put them after the operands (i.e. postfix). For example, `3 + 2` is represented by `3 2 +` and `(3 + 2) / 4` is represented by `3 2 + 4 /`. Other than being a little strange, this notation makes writing computer programs to evaluate these expressions simpler and we don't need parentheses. The programming languages Forth, Postscript, and Joy all use postfix notation. Other languages, like Lisp, use prefix notation.

Below is the start of our small postfix-arithmetic interpreter. The interpreter uses a data structure called a **stack** to pass values to and from the postfix functions. When using a stack, we treat the list like a stack of plates or papers; we add and remove items from the end of the list. The list is accessed in last-in-first-out order. The `pop()` method for lists grabs and deletes the last element of the list. The `append()` methods pushes values onto the end of the list. Use `assert` to perform better error handling.

```
class Interpreter:
    def __init__(self):
        self.operators = {'display': self.display, '+': self.add, '-': self.sub}
        self.stack = []

    def display(self):
        assert len(self.stack) > 0, "Nothing on the stack to display"
        v = self.stack.pop()
        print (v)

    def add(self):
        v2 = float(self.stack.pop())
        v1 = float(self.stack.pop())
        self.stack.append(v1 + v2)

    def sub(self):
        v2 = float(self.stack.pop())
        v1 = float(self.stack.pop())
        self.stack.append(v1 - v2)

    def interpret(self, expression):
        for token in expression.split():
            if token in self.operators:
                operator = self.operators[token]
                operator()
            else:
                self.stack.append(token)

i = Interpreter()
i.interpret('5 display')
i.interpret('3 2 + display')
i.interpret('3 2 + 4 - display')
```

[1] http://en.wikipedia.org/wiki/Reverse_Polish_notation

A Postfix Robot Programming Language

Information flows through the postfix python programs using the stack. **The commands get their parameters from the stack and push their output onto the stack.** Your language should have the following functionality:

1. A display command for printing out values (`i.interpret('5 display')`).
2. The ability to perform simple mathematical expressions (+, -, /, *, **). (`i.interpret('5 5 *')`).
3. Simple forward, backward, turnRight, turnLeft, beep commands. (`i.interpret('1 440 beep')`).
4. Commands for reading the sensors (`i.interpret('getBattery display')`).
5. Commands for taking and showing pictures. (`i.interpret('takePicture showPicture')`).
6. `read` interprets a list of commands from a specified file (`i.interpret('test.yp read')`).
7. Lines starting with the # character should be treated as comments and ignored.
8. **EXTRA:** You might extend your language with `pop`, `dup`, `swap` commands that remove the top item on the stack, duplicate the top item on the stack, and swap the top two items on the stack, respectively.
9. **EXTRA:** The ability to store numbers in variables; for this a dictionary would be useful.

An Example Postfix-Python Program (`test.yp`)

```
# beep both an A and an E for 0.5 seconds
0.5 440 2 * 650 beep2

# print out the current left light value
getLeftLight display

# take and show a picture
takePicture showPicture

# go forward for half a second and full power
1 0.5 forward

# EXTRA: create a variable named power with the value 0.5
0.5 power store

# EXTRA: go backward at power for 1 second
power 1 backward
```

Learning Objectives

◦ Read from Files ◦ Use Dictionaries ◦ Create a Simple Interpreter

Deliverables

Your program should have your name, email, assignment description, the date, and collaboration statement at the top of the file as a comment. You should submit a zip file that expands to a folder with two files:

```
cmsc143-lab13-LASTNAME-FIRSTNAME
   lab13.py   -- Your interpreter
   test.yp    -- An example program using all your language's features.
```