

Programming Assignment 2

CptS 355 - Spring 2006

An Interpreter for a PostScript-like Language

September 11, 2006

Overview

Assigned: Sept. 11, 2006

Due: Monday, October 2, 2006. Develop all your code in a directory named “two”. When you are finished, make a gzipped tar file or zip file of the directory and turn it in using the turn-in web page as for the first assignment. The directory must also contain a shell file that will run on the linux machines in the Sloan 353 lab. The name of the shell file must be `sps`. **You are free to develop your code on your own machine, but you must ensure that it works in the 353 lab.**

Credit: This assignment will count approximately 11% of your final grade.

Policy: This assignment is to be your own work. Do not work with other students on it or copy code that you find on the web (or any other source for that matter): if you have a question or are stumped see the instructor or the TA.

The problem

In this assignment you will write an interpreter in Python for a small PostScript-like language, concentrating on key computational features of the abstract machine, omitting all PS features related to graphics, and using a somewhat-simplified syntax.

The simplified language, SPS, has the following features of PS

- integer constants, e.g. 123: $-2^{31} \leq n < 2^{31}$
- boolean constants, true and false

- name constants, e.g. `/fact`: start with a `/` and letter followed by an arbitrary sequence of letters and numbers
- names to be looked up in the dictionary stack, e.g. `fact`: as for name constants, without the `/`
- built-in operators: `add`, `sub`, `mul`, `div`, `eq`, `lt`, `gt`, `and`, `or`, `not`
- built-in sequencing operators: `if`, `ifelse`
- stack operators: `dup`, `exch`, `pop`, `roll`
- code constants: code between matched curly braces `{ ... }`
- dictionary creation operator: `n dict`
- dictionary stack manipulation operators: `begin`, `end`
- name definition operator: `def`

An SPS program is a sequence of numbers, names, operators, and braces. You may assume if you wish that there is a whitespace character (space, tab, newline, etc.) between each pair of input tokens. For example, it is ok of your interpreter cannot handle `{3}` but instead requires it to be written `{ 3 }`.

Input to the interpreter is an SPS program read from the file named on the command line. You can read the entire input in one shot with

```
open(sys.argv[1]).readlines()
```

which reads the entire input into a list of lines. Your program does not have to be interactive – assume that all of the input is available when it starts executing.

Output of the interpreter consists of the contents of the operand stack, printed one value per line, when the SPS program terminates.

The assignment

Write an interpreter for SPS in Python. Make use of Python datatypes like dictionaries and lists to implement key data structures of the interpreter, for example dictionaries and stacks. For correct SPS programs your interpreter should produce the same output (stack contents) as produced by a PostScript interpreter. For incorrect programs your interpreter should print a helpful error message.

The code for this assignment will be used as a starting place for a future assignment. Therefore, it is important to do a good job of organizing and documenting your code so it can be easily understood and modified several weeks later. In particular you must modularize the parts of your code that implement the following components of the SPS abstract machine:

- the *reader*, which is responsible for reading in the program and breaking it into individual tokens (numbers, names, braces, etc.) Make good use of Python string handling here.
- the operand stack
- SPS dictionaries
- the dictionary stack
- the execution stack: you can either make this an explicit data structure or use Python recursion to embed the SPS execution stack in the Python call stack. Ask in class when you are at this point.

Observations on the assignment

Unlike the ghostscript interpreter, the SPS interpreter is not interactive. It is not required to produce any output until all of its input has been read and processed. However, you may wish to implement the `stack` operation for your own debugging use.

Several different types of values may be stored in dictionaries and in the operand stack.

Grading

Assignments will be graded for

- Correct and appropriate implementation of reader - 10%
- Correct and appropriate implementation of operand stack - 20%
- Correct and appropriate implementation of dictionary and dictionary stack operations - 20%. Make sure that your code treats `dict`, `begin` and `end` as separate operations.
- Correct and appropriate implementation of execution stack - 15%
- Comments and design documentation - 15%
- Correct functioning on SPS programs of our choice - 20%
 - This last point means that *you* are responsible for creating test cases and examining the specification for ambiguity or unclarity. I will answer questions with email to the class. I will not answer questions of the form “what is this SPS program supposed to print” unless you point out specifically why it is not clear what the correct answer is supposed to be.
- Deductions will be made if the program produces unspecified output (for example because debugging output is being produced).

Experience in previous versions of CptS 355 shows that almost all students are able to achieve grades greater than 90% on programming projects. If you are aspiring to or receiving grades less than 90% you should be worried.

This project will probably require 200-400 lines of code. If you fully understand the PostScript execution model it will be straightforward. If you do not fully understand PostScript execution you will write spend a lot of time writing code that won't be worth much.