plOtter: Proposal

Ibrahima Niang (in2190),
Ranjith Kumar Shanmuga Visvanathan (rs3579),
Sania Arif(sa3311)

1 Introduction

plOtter is a data manipulation language built on the principles and appreciation of a minimalist aesthetic. The goal of plOtter is to provide a means for users to design and implement their own plots to visualize data, that would otherwise be done by rigid, automated software (such as Microsoft Excel). Our language will simplify charting maps through domain-specific types and operations. The users can build their custom graph templates using our language and can reuse the template to visualize different data points.
Very primitive examples could be building a bar-graph, histogram etc. Higher level examples could be drawing small album art, or other complex examples could be to visualize/plot a binary tree, Gantt chart, etc.

2 Syntax

The syntax for plOtter is based loosely on the syntax of python. We incorporate white space de-limitations, if-then-else, while and break, for loops, inherent data structures, as well as user-defined data types. Also important to a graphing-friendly language are certain built-in functions, such as line-type to plot specific types of lines, which we incorporate.

3 Target Language and Output

As of now, we plan to compile plOtter to C, and display the output in SVG (an XML-based vector image format for two-dimensional graphics)

3.1 Comments

Single Line	#
Multi Line	/* */

3.2 Built-in data types

Our language supports the following data-types:

num	32 bit floating point number
string	Sequence of characters closed with double quotes
bool	Boolean value
point	A bi-element tuple of type *num*
list	List of *num* or *string* elements
hash	Dictionary with *num* or *string* keys

3.3 Operators

Arithmetic	+, -, *, /, %, **
Comparison	==, ! =, <, >, <=, >=
Assignment	=, + =, − =, ∗ =, / =, % =, ∗ =
Membership	in, not in
Binary	&, ∧, ‖,

3.4 Loops

while condition:
for init; cond; inc/dec:
for type name = start; end:
for start; end:
for variable in list:

3.5 Conditional statements

if condition: #statements *else if* condition: #statements *else*: #statements
if condition *then* statement *else* statement

3.6 Primitive function(s)

line(*point* p1, *point* p2, *hash* options)
This draws a line from p1 to p2, along with the options for line type
(For example: arrow-headed, dotted, dashed, etc.)

3.7 Built-in functions

These are the functions our language provides, which can be built from the primitive blocks in plOtter:

forEachDo(*list* data, *fn* f)
This applies f1 to data.

printHorizontal(*point* p, *string* s, *hash* options)
This prints s, horizontally, at point p, with specified options (For example: color, size, etc.)

printVertical(*point* p, *string* s, *hash* options)
This prints s, vertically, at point p, with specified options (For example: color, size, etc.)

rectangle(*point* p, *num* width, *num* height, *hash* options)
This draws a rectangle of width and height, starting from p, with options (For example: color, border, etc.)

barGraph(*list* data, *hash* options)
This draws a barGraph given the data and specified options (For example: padding, max width, max height, etc.)

We will have more functions, specific to each type of plot (For example: line plot, pie charts, etc.) similar to the barGraph function, for the user to call.

The purpose of providing these built-in functions is to demonstrate the power of our language, enabling the user to build his own custom functions, as desired.

4 Source Code

This is a sample program that gets data from a csv file, applies some options to the data and plots the data in a bar-graph. We also supply sample code for the bar-graph function, written in plOtter, which will be provided as an in-built function to the user, to show the ability to build custom graphs/plots using our language.

```
# Function written by user
fn convertToCelcius(num f):
    return (f-32)/1.8

# Importing data from CSV
list data = getDataFromAsListCSV("data.csv")

# Data manipulations Converting Farenheit to Celcius
forEachDo(data, convertToCelcius)

hash options = {"xAxis": true,"yAxis": true,"xLabel": "Time","yLabel": "Celcius", "title":"
    Temperature with Time","grid":true}

# Calling in-built function (which could alternatively be implemented y user)
barGraph(data, options)
```

Listing 1: Data-manipulation and Bar Graph Example

The output of the above code will be like something below.

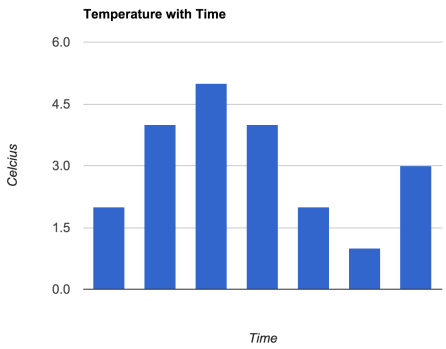

Figure 1: Simple Bar Graph

The function barGraph is built using primitive blocks and is provided as a built-in function. However, its shown here to emphasize the power user can have for creating his own graph.

```
#Bar Graph
fn barGraph(list data, hash options):
    num mx
    point pMax, pad = (10,10)

    #Apply Options specified by user, if not specified, use defaults
    options = applyOptions(options)

    #Setting size set by the user or taking default
    pMax[0] = if "xMax" in options then options.xMax else 1080
    pMax[1] = if "yMax" in options then options.yMax else 1920

```

```
13
14      #Scaling data to Fit on display appropriately
15      mx = max(data)
16      options.gap  = (pMax[0] - pad[1])/(options.barWidth*len(data))
17
18      for i in data:
19          rectangle( (st[0],(pMax[1]-pad[1])*i/mx),barWidth, barHeight, options.barStyle)
20          pad[0] += barWidth+gap
```

Listing 2: Snippet of code for in-built barGraph function in plOtter.