GUESS Interpreter Quick Reference

Introduction:

Welcome to the GUESS Interpreter Quick Reference! As you may have already discovered, a major portion of the GUESS Graph System is its Python Interpreter. The Interpreter is a powerful tool that allows the user to use Python commands and short scripts to manipulate a graph. In order to tap into what the Interpreter can do, you need to know a few Python basics. For a good, quick Python tutorial, visit: http://www.hetland.org/python/instant-python.php

Important Variables:

g – represents entire graph
g.nodes – list of all nodes in graph
g.edges – list of all edges in graph

Accessing Nodes and Edges

Access nodes by typing their name into the Interpreter.

Edges are represented by the first node name("node1") and the second node name ("node2") separated by 5 possible operators:

node1-node2 – undirected edge
node1<-node2 – directed edge (node2 to node1)
node1->node2 – directed edge (node 1 to node 2)
node1<->node2 – undirected edge (or bidirectional)
node1?node2 – any type of edge between node1 and node2

Accessing and Changing Attributes:

To access node and edge attributes, use the dot operator (".").

node1.color
(node1?node2).color
(node1->node2).weight
node1.name
(node2-node1).visible

To change Node and Edge attributes, use the dot operator followed by the assignment operator.

node1.color = blue	Make node1 blue
(node1?node2).color = cyan	Make all types of edges between node1 and node2 cyan
(node1->node2).weight = 5	Make the edge from node1 to node2 have a weight of 5
node1.visible = 0	Make node1 invisible

For numerical attributes, you may access the max or min for that attribute over all nodes or edges by typing the attribute name plus the dot operator and "min" or "max".
Example:

weight.min returns the minimum weight for all edges
salary.max returns the max salary for all nodes

List of Basic Node Attributes:

x – a double representing the node's x location (default: random)

y – a double representing the node's y location (default: random)

visible – a boolean indicating if the node should be displayed (default: true)

color – a string, the default color of the node (default: "blue"). We have a long list of color names that we know about, but if you didn't want to use one of those you could quote an rgb triplet (e.g. "124,234,222")

fixed – boolean, can the node be moved? (default: false)

style – an int indicating which style of node to use (default: 1). Currently GUESS maps: rectangle = 1, ellipse = 2, rounded rectangle = 3, text inside a rectangle = 4, text inside an ellipse = 5, text inside a rounded rectangle = 6, and an image = 7

width – double, node width (default: 4)

height – double, node height (default: 4)

label – string, a label for the node in the visualization (default is the name)

labelvisible – boolean, should we show the label? (default: false)

image – string, a filename of the image to use if the node style = 7

outdegree* – returns the node's outdegree

indegree* – returns the node's indegree

totalDegree* – returns the node's total degree

betweenness* – returns the node's betweenness

*Designates attributes that are calculated for all nodes the first time you access them. This means the first access will be slow, but following accesses will be fast.

List of Basic Edge Attributes:

visible – a boolean indicating if the edge should be displayed (default: true)

color – a string, the default color of the node (default: "green").

weight – a double indicating the edge weight (default: 1, but not currently used for calculations)

width – double, node width (default: .3)

directed – boolean, indicating edge directionality (default: false, undirected/bidirected). If true, this will assume node1 is the source and node2 is the destination.

label – string, a label for the node in the visualization (default is the edge weight)

labelvisible – boolean, should we show the label? (default: false)

node1 – access the first node of the edge

node2 – access the second node of the edge

source – access the source of the edge (only for directed edges)

target – access the target of the edge (only for directed edges)

Adding Attributes:

Use the command **addNodeField**(*name, type,default)* to add a new attribute to nodes. Type options are from the java.sql.Types class, for example Types.INTEGER or Types.VARCHAR.

To add a field to an edge, use **addEdgeField**(*name, type, default*).

Example: **addNodeField("salary", Types.INTEGER, Integer(10000))**
addNodeField("team", Types.VARCHAR, "Carolina Hurricanes")

Using Lists:

Lists in GUESS have all the same functionality as in Python, with some extra shortcuts.

For example, you can access an attribute for all elements inside a list by using the dot operator.

nodes = [node1, node2, node3]
nodes.color = maroon (all nodes in "nodes" are colored maroon)

path = [node1?node2, node2?node3, node3->node4]
path.weight = 10 (each edge in "path" gets weight 10)

For attributes common to edges and nodes, you can use the dot operator on a list of both nodes and edges.
group = [node1, node2-node1, node10]
group.color = black

You can also group nodes or edges together into lists by using attributes and boolean expressions.

For example: **color = = blue** would return all nodes and edges whose color is blue.
w**eight = = 10 or color == green** would return all nodes and edges whose weight is 10 or whose color is green
salary = = salary.max would return all nodes whose salary equals the maximum salary.
MaxSalaryGroup = (salary = = salary.max)
smallWeightGroup = (weight < 10)
group5 = (weight >= 5 and color == blue)

Other Handy Functions:

node1.unweightedShortestPath(node2) – returns the shortest path between node1 & node2 as a list of edges. Uses Breadth First Search

node1.dijkstrasShortestPath(node2) – same as above, using Dijkstras Algorithm.

addNode(name) – Make a new node

addDirectedEdge(source, destination) – adds a directed edge

addUndirectedEdge(node1, node2) – adds an undirected edge

addEdge(node1, node2) – calls addUndirectedEdge

removeNode(node) – removes and returns node

removeEdge(edge) – removes and returns edge

remove(list of nodes and edges) – removes and returns list of nodes and edges

execfile("pythonScript.py") -- loads a python script file. If there are no class or method definitions, will execute the instructions as if they were typed into the interpreter. If there are definitions, you can now access them by just typing their name. For example, if I load a .py file containing a method "doStuff()" you can type: **execfile("pythonScript.py")** and then type "**doStuff()**"

complement(node, edge, or list) – returns a list: the complement of whatever you pass it.

groupBy(node or edge attribute) – returns a list of nodes or edges grouped together by the given attribute.

sortBy(node or edge attribute) – returns a list of nodes or edges sorted by the given attribute.

groupAndSortBy(node or edge attribute) – returns a list of nodes or edges that were first grouped by the given attribute, and then sorted

generateColors(color1, color2, number of colors) – returns an array of the specified size with colors between color1 and color 2

node.getNeighbors() -- returns a list of all neighbors of "node"

v.setBackgroundImage("image address") -- adds a background image to GUESS's view

quit – quits GUESS

center – re-centers the view on the graph

Example Code:

You can type these examples straight into the interpreter. Try it on buddyGRAPH.gdf.

This code groups and sorts nodes by outdegree, and then colorizes them from blue to red with blue being lowest outdegree and red being highest.

```
g.nodes[0].outdegree – so that GUESS calculates all the outdegrees
clusts = groupAndSortBy(outdegree)
clustcol = generateColors(blue,red,len(clusts))
for z in range(0,len(clusts)):
    clusts[z].color = clustcol[z]
```

This code makes a group and then returns the diameter of that group

```
import com
bfs = com.hp.hpl.guess.BFSAlgorithm()
group = (show == '24')
bfs.getGroupDiameter(group)
```

Graph Layouts:

For screenshots of the different layouts, go to http://www.cs.duke.edu/~bspain/GUESS/layouts

GEM – gemLayout() – groups nodes that are closer together, together. Looks nice. One of the faster layouts

Fruchterman-Rheingold – frLayout() - similar to GEM but has a more blocky look; aligns nodes into lines

Spring – springLayout() - Uses a model where nodes repel each other while edges pull nodes together. Takes a longer time, may pass it an integer for the number of loop iterations to run.

Physics – physicsLayout() - similar to spring but uses different math. May pass it a parameter for the number of iterations.

Kamada-Kawai – kkLayout() – another spring-type layout

Radial – radialLayout(centerNode) – places center node at center and all other nodes at varying radii away, depending on the distance to the center node

Circular – circleLayout() - place all nodes in a circle.

Random – randomLayout() - places nodes randomly within a square

MDS – mdsLayout() - uses edge weights to determine distance between nodes

Keyboard Shortcuts:

Up-Arrow – cycle backwards through previous command history

Down-Arrow – cycle forwards through previous command history

Ctrl-A – move cursor to the beginning of the line

Ctrl-B – move cursor to the end of the line

Ctrl-C – Exits multiline code (for example if you are typing a for loop and want to exit without running it)

Ctrl-K -- "Kill" command; cuts the code in front of the cursor and puts in the clipboard

Ctrl-U -- "Yank" command; cuts the code behind the cursor and puts it in the clipboard

Ctrl-W -- "Yank Word" command; cuts everything behind the cursor until the first white space

Ctrl-Y – Paste

Ctrl-L – Clears the interpreter

Ctrl-T – Swaps the character immediately before the cursor with the character immediately after the cursor

Using the Log:

To use GUESS's Interpreter log, go to the "File" menu and select "Log..." Choose where you would like to save the log in the File Chooser that appears. Now, every command you type into the Interpreter will be saved to this log file as a Python script. You can run this script any time by choosing the "Run Script..." option within the "File" menu, or by typing execfile("scriptname.py").

You may disable logging at any time by clicking on the "Log" option a second time in the "File" Menu

Taste in GUESS:

Taste is a collaborative filtering system that allows us to get recommendations for users. For example, Taste will take a data set showing different users preferences for different music artists, and then from that data can give one user a recommendation for a new artist to listen to.

Taste accomplishes this recommendation task by finding a users musical "neighbors" -- users that have many of the same preferences with the user who requests a recommendation. This neighborhood model is also a perfect source for making Graphs where nodes are users and edges are people who share the same preferences for music, movies, or practically any other thing.

To use Taste in GUESS:

- First we need to make a GDF file from the Taste data.
  - If you want to use the DukeScrobbler Data, simply double click on the jar file "DukeScrobbler2GDF.jar" this will grab the data from the DukeScrobbler website and generate a GDF file. A box will appear telling you where the *.gdf and *.taste files are saved: remember these file paths!
  - If you want to generate a GDF from your own Taste file, double click "UserFile2GDF.jar". This program will ask you to locate the taste file you want to load and then a box will appear telling you where the generated GDF file is located.
- Now double click on DukeGUESS.jar and then choose "Load GDF/GraphML" and select the gdf file that you just generated.
- In GUESS, go to the file menu and select "Run Script...". This will prompt you to select a "*.py" file. Select "loadAll.py" in the DukeGUESS directory.
- Immediately it will prompt you to select another file, this is asking for your Taste Data File. Either choose the automatically generated one if you used DukeScrobbler data or the file you chose to generate a GDF.
- Now you are free to use the Taste functions in the "Scrobble Functions" Panel.

How To Write Your Own Taste Correlation:

- Writing your own Taste correlation to be used in GUESS is a simple process
  1. Open Notepad or your favorite Python editor
  2. Write the function "computeResult" that takes 6 double parameters (n, sumXY, sumX, sumY, sumX2, sumY2) and returns a double.
  3. After opening the Scrobble Functions panel by following the instructions above, click on the "Use Your Own Correlation" button and select the python file holding your new computeResult function.
  4. GUESS Uses your function to find neighbors for each node and then displays the corresponding edges.
  5. Here is an example from the cosineCorrelation.py file included in the DukeGUESS directory:

```
def computeResult(n, sumXY, sumX, sumY, sumX2, sumY2):
        xTerm = Math.sqrt(sumX2)
        yTerm = Math.sqrt(sumY2)
        denominator = xTerm * yTerm
        if(denominator == 0.0):
                return 0.0
        else:

                return (sumXY / denominator)
```

Explanation of parameters:

n – the total number of items users have shown preferences for (or against) in the Taste database
sumX – sum of all item preferences that user X expresses
sumY – sum of all item preferences that user Y expresses
sumXY – if X and Y express a preference for the same item, take the product of their preferences and add it to this sum
sumX2 – same as sumX except each preference is squared before it is added to the sum
sumY2 – same as above except with sumY

Fun with Jack:

Now we will explore a graph built using the relationships found in the popular TV Show: *24*.

First, start GUESS and load the graph by double-clicking on DukeGUESS.jar and then opening the file "twentyFour.gdf"

Now lets use the Radial Layout, so that we can have a better idea of where Jack is within the graph. Either click on the "Layout" menu and enter "Jack" into the box that appears or type "radialLayout(Jack)" into the Interpreter.

Now all the nodes are positioned in circles according to how close they are to Jack. Jack is in the center, people he directly interacts with are in the inner circle, people his direct contacts interact with are in the middle circle, and people his direct contact's direct contacts interact with are in the outermost circle.

Wouldn't it be nice if we could color the nodes according to which circle they are in? We can!

First, we need to load a python script: from the "File" menu select "Run Script..." and open "simplegraph.py" in the DukeGUESS directory. Now we have access to several basic graph algorithms including Breadth First Search, which calculates the shortest distance between a source node and all other nodes in the graph.

- enter ***bfs(Jack)*** **–** this calculates the distance between Jack and every other node, and then stores it as the "dist" value associated with each node
- enter ***colorize(dist,blue,red)*** **–** this tells GUESS to colorize every node depending on that nodes "dist" value. For low values it uses colors closer to blue, for higher values it uses colors closer to red.

Now we see that all the red nodes are nodes far away from Jack and all the blue nodes are nodes closer to Jack (including Jack himself, who is very blue because the distance from any node to itself is 0).

Now, what if we forget what these colors mean? It would be nice to have some sort of *legend* to remind us in forgetful moments. GUESS has built-in legends that are perfect for this application!

- enter ***GradientLegend(blue, red, dist.min, dist.max, 1)*** **–** this tells GUESS to make a new GradientLegend from colors blue to red. The minimum amount is the minimum dist value over all nodes, the maximum amounts is the maximum dist value over all nodes, and that we want tick marks at intervals of 1.

Lastly, for a cooler look, lets colorize the edges according to where they are in the graph. The ***averageEdgeColor(edge)*** function changes the color of "edge" to be the average of the colors of nodes that form the edge. Unfortunately, there is no way to simply apply ***averageEdgeColor(edge)*** to the whole graph directly, so we will have to use a **for loop**:

```
>>> for edge in g.edges:
...         averageEdgeColor(edge)
```

Doesn't that look better?

For practice, try colorizing nodes based on their **outdegree** and then making a corresponding Gradient Legend. (Remember: You need to calculate one node's outdegree before you can refer to it as an attribute) Finally, change the edge colors using the ***averageEdgeColor(edge)*** function

How to Use Java Classes in GUESS:

If you are not as proficient with Python as you are with Java, or if the code you want to write is better suited to Java, it is important to know how to call your Java classes within the GUESS framework.

Let's say we write a new class **Foo** with a constructor **Foo(int x, Node y)** and the function **doStuff()** that returns an int.

Let's also say that we place this new class in the **com.hp.hpl.guess** java package.

In order to use this class, execute the following lines of code:

```
>>>from com.hp.hpl.guess import Foo
>>>foo = Foo(5, g.nodes[0])
>>>coolNumber = foo.doStuff()
```

Notes:

- When you pass a list (of nodes, edges, or edges and nodes) from the GUESS Interpreter to Java, you are passing a PyList, although PyList implements java.utils.List, many of the usual functions are not implemented (such as contains(PyObject o) and clear())
- When using a PyList, the objects contained in it are not Node or Edge objects, but PyObjects. In order to get a node from the PyObject, call g**raph.getNodeByName(objectInList.toString)**.
- GUESS changes System.out.println() to print to the Interpreter console, not to the Eclipse (or any other) console.