MAP

**shortestPathTester.map**

- Why one MAP instead?
  - GUI-based programming is difficult and complicated in other languages.
  - Other languages don't have basic graph data structures such as edges and nodes as inherent constructs, forcing the programmer to build these themselves.

**MAP: The Graph Programming Language**

- Sara Simkus - Project Manager
- Sandiya Santkaram - Language Guru
- Alex Meniuiev - System Architect
- Tomoy Inouye - Verification & Validation Person

**Language Syntactic Structure**

- Basic data types: Text, Strings, Boolean
- Embedded code type evaluation
- Matrix as new Node (new Node)
- Node as new Node (new Node)

**Project Management Tools**

- Asana, Google groups, Gmail, texting, and GitHub
shortestPathTester.map

```java
func main()
    Graph g = new Graph();
    Node ia = new Node("temp=80, humidity=low");
    Node ib = new Node("temp=85, humidity=low");
    Node ic = new Node("temp=85, humidity=high");
    UndirEdge ia, ib = new UndirEdge(ia, ib, {cost: 150});
    UndirEdge ib, ic = new UndirEdge(ib, ic, {cost: 50});
    g.findShortestPath(ia, ic, {cost: 180});
```

**MAP: The Graph Programming Language**

Serena Simkus - Project Manager
Sandy Sankaran - Language Guru
Alex Merkulov - System Architect
Alfred Tan - System Integrator
Tommy Inouye - Verification & Validation Person

**Why use MAP instead?**
- Graph programming is difficult and complicated in other languages
- Other languages don’t have basic graph data structures such as edges and nodes as inherent data types, forcing the programmer to build them themselves

**MAP is...**
- Simple
- Familiar
- Reusable
- Intuitive
- Imperative

**Language Syntactic Structure**
- Basic data types: Text, Numeric, Boolean
- Derived Data Type examples:
  - Graph g = new Graph();
  - Node ia = new Node("temp=80");
  - UndirEdge ia, ib = new UndirEdge(ia, ib, {cost: 150});
- Scoped with curly braces
- If, for, foreach
- Reserved words and characters

**Project Management Tools**
- Asana
- Google groups
- Gmail, texting, and GitHub

**MAP is different...**
- has edges, nodes, graphs, and paths as inherent data structures
- is easier and more accessible for beginner programmers
- abstracts away syntax and set-up without loss of functionality
MAP: The Graph Programming Language

Serena Simkus - Project Manager
Sandya Sankarram - Language Guru
Alex Merkulov - System Architect
Alfred Tan - System Integrator
Tommy Inouye - Verification & Validation Person
MAP is...

Simple

Familiar

Reusable

Imperative

Intuitive

PROGRAMMING

Y U SO HARD

Troll.me
Why use MAP instead?

- Graph programming is difficult and complicated in other languages
- Other languages don't have basic graph data structures such as edges and nodes as inherent data types, forcing the programmer to build them themselves
How MAP is different...

- has edges, nodes, graphs, and paths as inherent data structures
- is easier and more accessible for beginner programmers
- abstracts away syntax and set-up without loss of functionality
Language Syntactic Structure

- Basic data types: Text, Numeric, Boolean
- Derived Data Type examples:
  - Graph g = new Graph();
  - Node la = new Node({'temp':90});
  - DirEdge la_sj = new DirEdge(la, sj, {'cost': 100});
- Scoped with curly braces
- if, for, foreach
- Reserved words and characters
Built-in Functions

Graph.add(node)
Graph.delete(node)
Graph.addEdge(edge)
Graph.findShortest(node1, node2, attribute)
Graph.draw(filename)
... among others!
func main()

Graph g = new Graph();
Node la = new Node({'temp':90, 'humidity':'low'});
Node sj = new Node({'temp':50, 'humidity':'low'});
Node sf = new Node({'temp':65, 'humidity':'high'});
UnDirEdge la_sj = new UnDirEdge(la, sj, {'cost': 100});
UnDirEdge sj_sf = new UnDirEdge(sj, sf, {'cost':50});
UnDirEdge la_sf = new UnDirEdge(la, sf, {'cost':180});
g.findShortest(la, sf, 'cost');
Project Management Tools

Asana, Google groups, Gmail, texting, and GitHub
Translator Architecture

Source Program

Target Program

Map.py

traverse.py

MAPLexer.py

yacc.py

MAP Translator
Translator Architecture

- Map.py: indentation, scope
- MAPLexer.py: lex rules
- yacc.py: grammar, AST
- Traverse.py: AST->python
Run-time Environment

```
pip install virtualenv
virtual env
source env/bin/activate
pip install ply
pip install asciitree
pip install networkx
pip install -U pyyaml nltk
```
Software Development Environment

- Python 2.7.5
- GCC 4.2.1
- Mac OS X
Compiler-Generator Tools

- PLY: Python Lex-Yacc
Testing

Lexer Tester

- Ensured the effectiveness of lex
- Mostly ensured the flexibility of ID
- Ran a test to check tokens
- Ensures consistent tokens
- Easy to add new tokens to test

```
dyn-160-39-231-16:map tommynouye$ python lexertest.py
Asserting syntax of hello world and checking generated tokens:
Hello world test passed!

  .Testing all tokens:
  asserting key: 1234
  asserting key: elif
  asserting key: findShortest
  asserting key: false
  asserting key: read
  asserting key: Graph
  asserting key: else
  asserting key: in
  asserting key: .01231
  asserting key: equals
  asserting key: break
  asserting key: func
  asserting key: Time
  asserting key: path
  asserting key: 12312.213123
  asserting key: true
  asserting key: addEdge
  asserting key: if
  asserting key: Node
  asserting key: write
  asserting key: null
  asserting key: p1NULLdel
  asserting key: for
  asserting key: deleteEdge
  asserting key: Text
  asserting key: getEdge
  asserting key: Numeric
  asserting key: l
  asserting key: return
  asserting key: add
  asserting key: continue
  asserting key: foreach
  asserting key: adjacent
  asserting key: Path
  asserting key: include
  asserting key: delete
Token test passed!
```

Ran 2 tests in 0.010s

OK
Testing

Traverse Tester

- Tested compiler
- Looks for syntax errors
- Ensured consistency
- Easy to run
- Simple to add more tests
Testing

Interpreter Tester

- Checks indentation
- Runs error checking
- Easy to run
- Simple to add more tests
- Most important test

```python
import unittest
import sys
import os

class TestInterpreterSyntax(unittest.TestCase):
    def assert_proc(self, testname, outname):
        t = open(testname, 'r')
        o = open(outname, 'r')
        teststr = t.read()
        outstr = o.read()
        self.assertEqual(teststr, outstr)

def test_helloworld(self):
    os.system("python Map.py test/helloworld.map")
    self.assert_proc("test/helloworld.py", "test/helloworld.out")

def test_factorial(self):
    os.system("python Map.py test/factorial.map")
    self.assert_proc("test/factorial.py", "test/factorial.out")

def test_foreach(self):
    os.system("python Map.py test/foreachtest.map")
    self.assert_proc("test/foreachtest.py", "test/foreachtest.out")

def test_forstatement(self):
    os.system("python Map.py test/forstatementtest.map")
    self.assert_proc("test/forstatementtest.py", "test/forstatementtest.out")

def test_elseif(self):
    os.system("python Map.py test/elseifelse.ifset")
    self.assert_proc("test/elseifelse.ifset", "test/elseifelse.ifset.out")

def test_elseif_else(self):
    os.system("python Map.py test/elseif_test.map")
    self.assert_proc("test/elseif_test.map", "test/elseif_test.ifset")

def test_else(self):
    os.system("python Map.py test/else_test.map")
    self.assert_proc("test/else_test.map", "test/else_test.out")

def testsample1(self):
    os.system("python Map.py test/sample1.map")
    self.assert_proc("test/sample1.out", "test/sample1.py")

def test_include_input(self):
    os.system("python Map.py test/testincludeinput.map")
    self.assert_proc("test/testincludeinput.out", "test/testincludeinput.py")

def test_path_test(self):
    os.system("python Map.py test/path_test.map")
    self.assert_proc("test/path_test.py", "test/path_test.out")
```
Demo!

Here is a cool program that displays a graph of our team!

I DON'T ALWAYS TEST MY CODE

BUT WHEN I DO, I DO IT IN PRODUCTION
Conclusions

What we learned:
- Progress is important!! Keep moving forward
- GitHub, version control

What went well:
- Meetings!
- Communication
- Dividing up work

What we would have done differently:
- Start implementation earlier
- Working kernel earlier

Why MAP?
- Did you see that awesome demo?!?
- Graph programming made easier.
Thank you for listening!! :)

Repository:

https://github.com/PLT-MAP/map

The Team:
**MAP**

```
func main()
Graph g := new Graph();
Node la = new Node("temp":00, "humidity":low); Node lb = new Node("temp":50, "humidity":low);
Node lc = new Node("temp":65, "humidity":high");
Undirected Edge lj = new UndirectedEdge(la, lb, 100); Undirected Edge lk = new UndirectedEdge(lb, lc, 180);
```

**Why use MAP instead?**
- Graph programming is difficult and complicated in other languages
- Other languages don't have basic graph data structures such as edges and nodes as inherent data types, forcing the programmer to build them themselves

**MAP is...**
- Simple
- Familiar
- Reusable
- Intuitive

**MAP: The Graph Programming Language**
Serena Simkus - Project Manager
Sandyia Sankarram - Language Guru
Alex Merkulov - System Architect
Alfred Tan - System Integrator
Tommy Inouye - Verification & Validation Person

**How MAP is different...**
- Has edges, nodes, graphs, and paths as inherent data structures
- Is easier and more accessible for beginner programmers
- Abstracts away syntax and set-up without loss of functionality

**Language Syntactic Structure**
- Basic data types: Text, Numeric, Boolean
- Derived data type examples:
  - Graph g := new Graph();
  - Node la := new Node("temp":00);
  - Directed Edge lj := new DirectedEdge(la, lb, 100);
- Scoped with curly braces
- If, for, foreach
- Reserved words and characters

**Project Management Tools**
Asana, Google groups, Gmail, texting, and GitHub