JythonMusic Automated Testing Framework

Joey Baldwin, Eric Hofesmann, Marge Marshall

What is JythonMusic?

File Edit Run Help

2. furElise.py

```
1  # furElise.py
2  # Generates the theme from Beethoven's Fur Elise.
3
4  from music import *
5
6  # theme has some repetition, so break it up to maximize economy
7  # (also notice how we line up corresponding pitches and durations)
8  pitches1   = [E5, DS5, E5, DS5, E5, B4, D5, C5]
9  durations1 = [SN, SN,  SN, SN,  SN, SN, SN, SN]
10 pitches2   = [A4, REST, C4, E4, A4, B4, REST, E4]
11 durations2 = [EN, SN,   SN, SN, SN, EN, SN,   SN]
12 pitches3   = [GS4, B4, C5, REST, E4]
13 durations3 = [SN,  SN, EN, SN,   SN]
14 pitches4   = [C5, B4, A4]
15 durations4 = [SN, SN, EN]
16
17 # create an empty phrase, and construct theme from the above motifs
18 theme = Phrase()
19 theme.addNoteList(pitches1, durations1)
20 theme.addNoteList(pitches2, durations2)
21 theme.addNoteList(pitches3, durations3)
```

```
Jython Environment for Music (JEM) 4.5 [Sep-02-2016]
 Based on TigerJython 2.11.00 [August-10-2016]
 Using Jython 2.7.0 [Apr-29-2015]
```

Music.py, Gui.py, Timer.py

$$d = 69 + 12 \log_2 \left(\frac{f}{440\ \text{Hz}} \right).$$

Methods tested in music.py:

- freqToNote - This method converts a frequency value given in hertz to the corresponding midi note value and midi pitch bend value
- noteToFreq - This method converts a midi note value to a frequency value in hertz
- frange - This method creates a range of floats, using a start value (inclusive), end value (exclusive) and a step value.

Methods tested in gui.py:

- colorGradient - This method creates a range of color values designated by lists of three integers between 0 and 256, exclusive. The values in the list represent Red, Green, and Blue levels. It uses two such color value lists (a start and a stop color, effectively), and an integer representing the number of gradient shades to generate in between. It then creates a range of color value lists that represent a gradient transition from the starting color value to the stop color value.

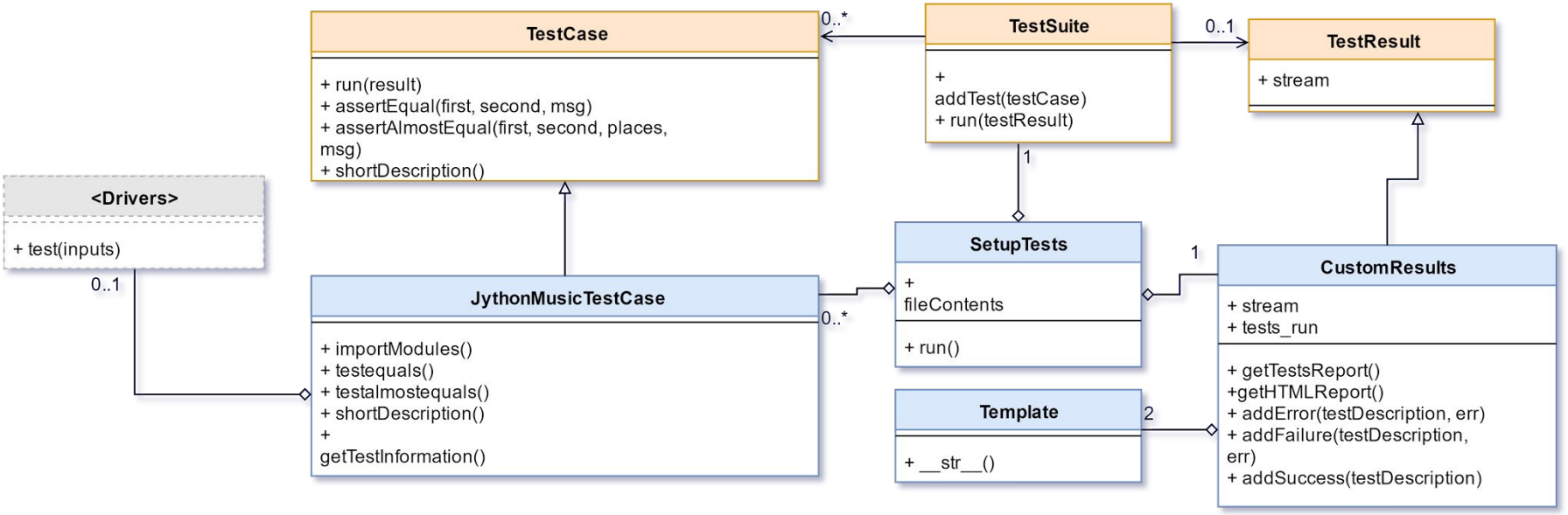
Methods tested in timer.py:

- getDelay - This method returns the delay time set for a particular timer
- getRepeat - This method returns whether or not a timer is set to repeat

Elements of Python Testing

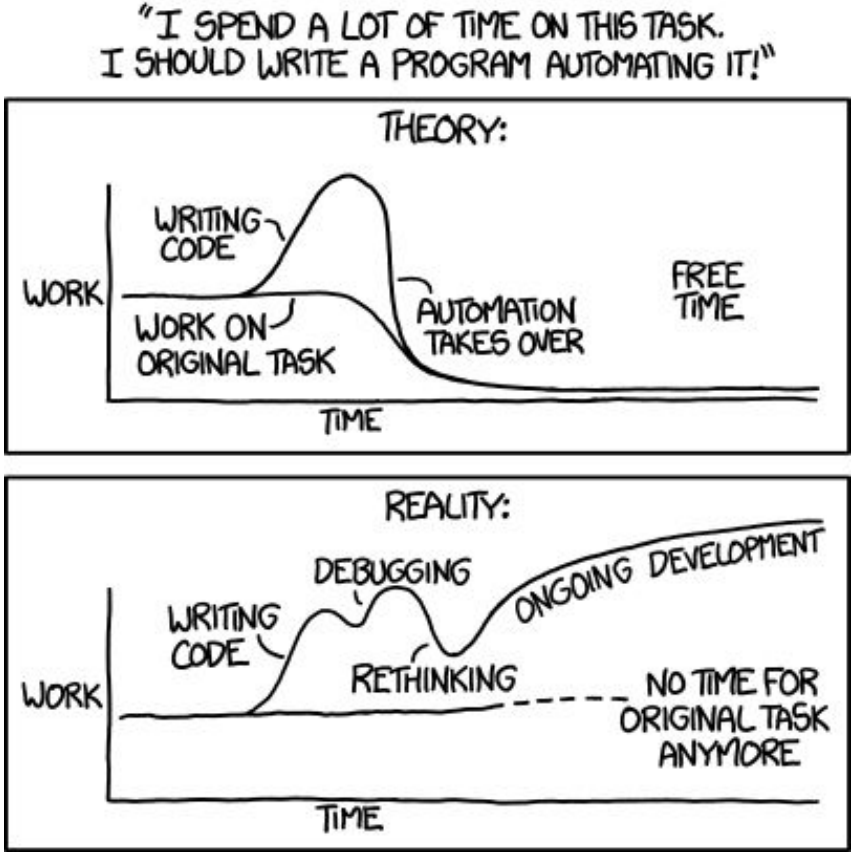
1. **Test Fixture** - prepares for tests
2. **Test Case** - logic for single test
3. **Test Suite** - collection of test cases
4. **Test Runner** - executes tests & outputs

Test Framework Model



JythonMusicTestCase

- Inherits from TestCase
- Highly Generalized
- Runs any function outside of a class
- Creates a specified driver to test classes



CustomResults

- TestSuite.run(CustomResults) **OR** TestCase.run(CustomResults)
- Contains an array of the results
- Contains the logic for creating HTML results

Assertions

```
def testequals(self):
    self.assertEqual(self.actualResults, self.outputValue)

def testalmostequals(self):
    self.assertAlmostEqual(self.actualResults, self.outputValue, self.testalmostequalsprecison) #Allow tester to set precision
```

Method	Checks that	New in
`assertAlmostEqual(a, b)`	`round(a-b, 7) == 0`	
`assertNotAlmostEqual(a, b)`	`round(a-b, 7) != 0`	
`assertGreater(a, b)`	`a > b`	2.7
`assertGreaterEqual(a, b)`	`a >= b`	2.7
`assertLess(a, b)`	`a < b`	2.7
`assertLessEqual(a, b)`	`a <= b`	2.7
`assertRegexpMatches(s, r)`	`r.search(s)`	2.7
`assertNotRegexpMatches(s, r)`	`not r.search(s)`	2.7
`assertItemsEqual(a, b)`	sorted(a) == sorted(b) and works with unhashable objs	2.7
`assertDictContainsSubset(a, b)`	all the key/value pairs in *a* exist in *b*	2.7

Method	Checks that	New in
`assertEqual(a, b)`	`a == b`	
`assertNotEqual(a, b)`	`a != b`	
`assertTrue(x)`	`bool(x) is True`	
`assertFalse(x)`	`bool(x) is False`	
`assertIs(a, b)`	`a is b`	2.7
`assertIsNot(a, b)`	`a is not b`	2.7
`assertIsNone(x)`	`x is None`	2.7
`assertIsNotNone(x)`	`x is not None`	2.7
`assertIn(a, b)`	`a in b`	2.7
`assertNotIn(a, b)`	`a not in b`	2.7
`assertIsInstance(a, b)`	`isinstance(a, b)`	2.7
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	2.7

Charts from: https://docs.python.org/2/library/unittest.html

Test Cases

<test case id>	05
<description of function's purpose>	Create range of floats
<module name> <class name if applicable>	music
<function name>	frange
<input>	1.0, 5.0, 0.5
<test type of expected output>	testequals
<expected output>	[1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5]

Drivers

- Parser will pass “None” to testClass if it isn’t specified in testCase.txt
- If there is a class, the parser opens driver and calls test() method
  - testCase_<Class name>_<Method name>.py
  - Each driver needs a method called test()
  - Passes inputs variable into test()

```
133         if self.testClass == None:
134             try:
135                 self.actualResults = getattr(getattr(self, self.testModule), self.testFunction)(*self.inputs)
136             except Exception, e:
137                 self.actualResults = type(e)
138         else:
139             self.executableName = "testCase_" + str(testClass) + "_" + str(testFunction)
140             setattr(self, self.executableName, __import__(self.executableName))
141             try:
142                 self.actualResults = getattr(self, self.executableName).test(*self.inputs)
143             except Exception, e:
144                 self.actualResults = type(e)
```

Timer2 getDelay Driver

- Inputs is an integer
- Inputs is the only variable, every other property of the class is hard coded
- Add path
- Main is for driver testing

```
1  import os,sys
2  sys.path.append(os.path.join(os.path.dirname(__file__), "../project/src"))
3  import timer
4
5  def echoTime():
6      global seconds
7      seconds = seconds + 1  # update time
8
9  def test(inputs):
10     testClass = timer.Timer2(inputs, echoTime, [], True)
11     actualResults = testClass.getDelay()
12     return actualResults
13
14 if __name__ == '__main__':
15     print test(True)
16
```

Timer2 getRepeat Driver

- Inputs is a boolean
- Inputs can be a list if multiple inputs are required

```
1  import os,sys
2  sys.path.append(os.path.join(os.path.dirname(__file__), "../project/src"))
3  import timer
4
5  def echoTime():
6      global seconds
7      seconds = seconds + 1  # update time
8
9  def test(inputs):
10     testClass = timer.Timer2(1000, echoTime, [], inputs)
11     actualResults = testClass.getRepeat()
12     return actualResults
13
14 if __name__ == '__main__':
15     print test(True)
16
```

Output Format

Number	Description	Module	Function	Type	Inputs	Expected	Actual	
00	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[69]	440.0	440.0	success
01	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[48]	130.81	130.81278265	success
02	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[72]	523.25	523.251130601	success
03	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[1]	8.66	8.66195721803	success
04	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[127]	12543.85	12543.8539514	success
05	Create range of floats	music	frange	testequals	[1.0, 5.0, 0.5]	[1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]	[1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]	success
06	Create range of floats	music	frange	testequals	[0.0, 44.0, 11.0]	[0.0, 11.0, 22.0, 33.0]	[0.0, 11.0, 22.0, 33.0]	success
07	Create range of floats	music	frange	testequals	[0.02, -0.1, -0.02]	[0.02, 0.0, -0.02, -0.04, -0.06, -0.08]	[0.02, 0.0, -0.02, -0.04, -0.06, -0.08]	success
08	Create range of floats	music	frange	testequals	[0.0, 44.1, 11.0]	[0.0, 11.0, 22.0, 33.0, 44.0]	[0.0, 11.0, 22.0, 33.0, 44.0]	success
09	Create range of floats	music	frange	testequals	[44.1, 0.1, -11.0]	[44.1, 33.1, 22.1, 11.1]	[44.1, 33.1, 22.1, 11.1]	success
10	Create range of floats	music	frange	testequals	[44.1, 0.1, 0.0]	ValueError	ValueError	success
11	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[255, 0, 0], [0, 0, 255], 3]	[[255, 0, 0], [170, 0, 85], [85, 0, 170]]	[[255, 0, 0], [170, 0, 85], [85, 0, 170]]	success
12	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[100, 100, 0], [100, 100, 0], 3]	[[100, 100, 0], [100, 100, 0], [100, 100, 0]]	[[100, 100, 0], [100, 100, 0], [100, 100, 0]]	success
13	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[0, 0, 0], [255, 255, 255], 1]	[[0, 0, 0]]	[[0, 0, 0]]	success
14	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[255, 255, 255], [0, 2, 0], 3]	[[255, 255, 255], [170, 170, 170], [85, 86, 85]]	[[255, 255, 255], [170, 170, 170], [85, 86, 85]]	success
15	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[0, 0, 0], [255, 255, 255], 2]	[[0, 0, 0], [127, 127, 127]]	[[0, 0, 0], [127, 127, 127]]	success
16	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[466.1637615181]	(70, 0)	(70, 0)	success
17	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[451.0]	(69, 1751)	(69, 1751)	success
18	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[77.22]	(39, -514)	(39, -514)	success
19	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[2003.9]	(95, 1011)	(95, 1011)	success
20	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[440.0]	(69, 0)	(69, 0)	success
21	Get if the timer repeats or not	timer Timer2	getRepeat	testequals	[False]	False	False	success
22	Get if the timer repeats or not	timer Timer2	getRepeat	testequals	[True]	True	True	success
23	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[0]	0	0	success
24	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[10000000000L]	10000000000	10000000000	success
25	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[1000]	1000	1000	success

Injected Faults

(1) Tests 0 - 4: noteToFreq
  frequency = concertPitch * 2 ** ((note - 69) / 12.0)
  frequency = concertPitch * 2 ** ((note - 70) / 12.0)

(2) Tests 5 - 10: frange
  if step > 0:
  if step < 0:
results only contained the first value in the list

(3) Tests 5 - 10: frange
  done = start >= stop
  done = start > stop
results included an extra value at the end of the list

(4) Tests 16 - 20: freqToNote
  note = round(x)
  note = x

gui.py:

(5) Tests 11-15: colorGradient
  differenceR = red2 - red1
  differenceR = red1 - red2

Fault Injection Results

Number	Description	Module	Function	Type	Inputs	Expected	Actual	
00	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[69]	440.0	415.30469758	fail
01	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[48]	130.81	123.470825314	fail
02	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[72]	523.25	493.883301256	fail
03	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[1]	8.66	8.17579891564	fail
04	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[127]	12543.85	11839.8215268	fail
05	Create range of floats	music	frange	testequals	[1.0, 5.0, 0.5]	[1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]	[1.0]	fail
06	Create range of floats	music	frange	testequals	[0.0, 44.0, 11.0]	[0.0, 11.0, 22.0, 33.0]	[0.0]	fail
07	Create range of floats	music	frange	testequals	[0.02, -0.1, -0.02]	[0.02, 0.0, -0.02, -0.04, -0.06, -0.08]	[0.02]	fail
08	Create range of floats	music	frange	testequals	[0.0, 44.1, 11.0]	[0.0, 11.0, 22.0, 33.0, 44.0]	[0.0]	fail
09	Create range of floats	music	frange	testequals	[44.1, 0.1, -11.0]	[44.1, 33.1, 22.1, 11.1]	[44.1]	fail
10	Create range of floats	music	frange	testequals	[44.1, 0.1, 0.0]	ValueError	ValueError	success
11	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[255, 0, 0], [0, 0, 255], 3]	[[255, 0, 0], [170, 0, 85], [85, 0, 170]]	[[255, 0, 0], [340, 0, 85], [425, 0, 170]]	fail
12	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[100, 100, 0], [100, 100, 0], 3]	[[100, 100, 0], [100, 100, 0], [100, 100, 0]]	[[100, 100, 0], [100, 100, 0], [100, 100, 0]]	success
13	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[0, 0, 0], [255, 255, 255], 1]	[[0, 0, 0]]	[[0, 0, 0]]	success
14	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[255, 255, 255], [0, 2, 0], 3]	[[255, 255, 255], [170, 170, 170], [85, 86, 85]]	[[255, 255, 255], [340, 170, 170], [425, 86, 85]]	fail
15	calculate the color gradient given two colors and a step	gui	colorGradient	testequals	[[0, 0, 0], [255, 255, 255], 2]	[[0, 0, 0], [127, 127, 127]]	[[0, 0, 0], [-128, 127, 127]]	fail
16	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[466.1637615181]	(70, 0)	(70, 0)	success
17	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[451.0]	(69, 1751)	(69, 0)	fail
18	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[77.22]	(39, -514)	(38, 0)	fail
19	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[2003.9]	(95, 1011)	(95, 0)	fail
20	calculate the midi pitch of a given frequency value	music	freqToNote	testequals	[440.0]	(69, 0)	(69, 0)	success
21	Get if the timer repeats or not	timer Timer2	getRepeat	testequals	[False]	False	False	success
22	Get if the timer repeats or not	timer Timer2	getRepeat	testequals	[True]	True	True	success
23	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[0]	0	0	success
24	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[10000000000L]	10000000000	10000000000	success
25	Get if the delay time interval (in milliseconds)	timer Timer2	getDelay	testequals	[1000]	1000	1000	success

Future Work

- Automate class drivers
  - Alternatively create interface to force test() instantiation
- Setter methods
- Batching Tests

Questions?