Software Testing: Project

Simran Basu
MT2020145

Jyotsna Tata
MT2020020

Md Aqueeb Jawed
MT2020110

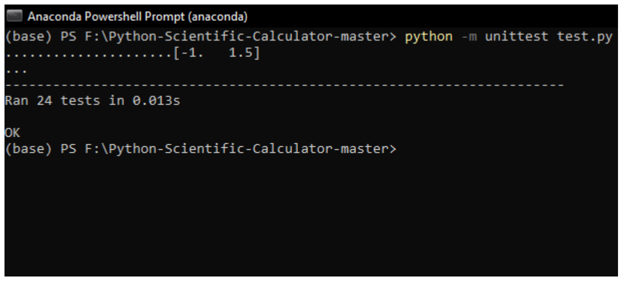
***Abstract*—This report contains the implementation of test cases on a project using open source tools and the test case design strategies.**

I. PROBLEM STATEMENT

Mutation-source-code: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used.

II. METHODOLOGY

For this project, A python program for scientific calculator was used which uses many functions such as Sine, Cosine, Logarithm, Factorial, Permutations, etc. The test cases were first designed and built for unit testing of each of these functions. Unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use. Python's unittest library was used for that. It is a unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.



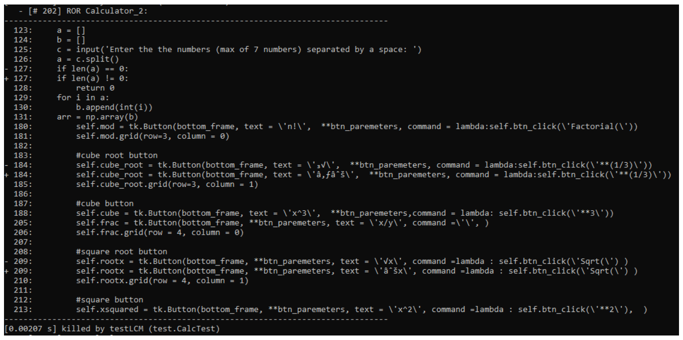
```
Anaconda Powershell Prompt (anaconda)
(base) PS F:\Python-Scientific-Calculator-master> python -m unittest test.py
.....................[-1.   1.5]
...
----------------------------------------------------------------------
Ran 24 tests in 0.013s

OK
(base) PS F:\Python-Scientific-Calculator-master>
```

Fig. 1. Unit Testing.

After ensuring that none of the test cases for unit testing were failing, mutation testing was used. Mutation testing is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways. Each mutated version is called a mutant and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. Python's mutpy was used for this. Mutpy is a Mutation testing tool in Python that generated mutants and computes a mutation score. It supports standard unittest module, generates YAML/HTML reports and has colorful output. After running mut.py, it was found that for 6 different mutation operators, mutants were strongly killed. The mutation operators used were:

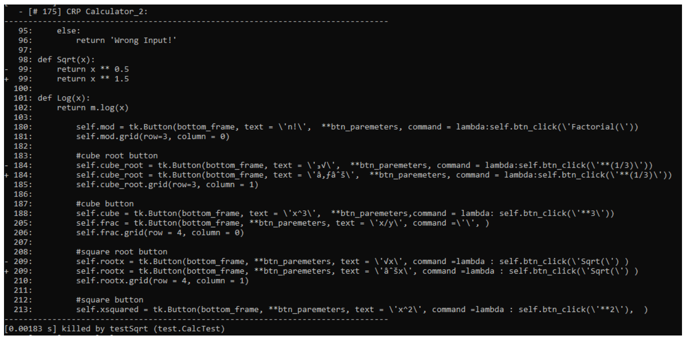
1) ROR (Relational Operator Replacement): Each occurrence of one of the relational operators (¡, ¿, ¡=, ¿=, ==, !=) is replaced by each of the other operators and by falseOp and trueOp.



```
  - [# 202] ROR Calculator_2:
--------------------------------------------------------------------------------
  123:     a = []
  124:     b = []
  125:     c = input('Enter the the numbers (max of 7 numbers) separated by a space: ')
  126:     a = c.split()
- 127:     if len(a) == 0:
+ 127:     if len(a) != 0:
  128:         return 0
  129:     for i in a:
  130:         b.append(int(i))
  131:     arr = np.array(b)
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â‚ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00207 s] killed by testLCM (test.CalcTest)
```

Fig. 2. Relational Operator Replacement.

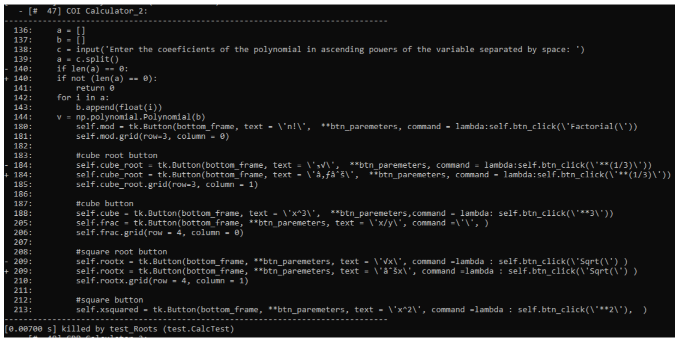
2) CRP (Constant Replacement): Each occurrence of a constant (1,0,2,-1) is replaced by other constants similar to it.



```
  - [# 175] CRP Calculator_2:
--------------------------------------------------------------------------------
   95:     else:
   96:         return 'Wrong Input!'
   97:
   98: def Sqrt(x):
-  99:     return x ** 0.5
+  99:     return x ** 1.5
  100:
  101: def Log(x):
  102:     return m.log(x)
  103:
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â‚ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00183 s] killed by testSqrt (test.CalcTest)
```

Fig. 3. Constant Replacement.

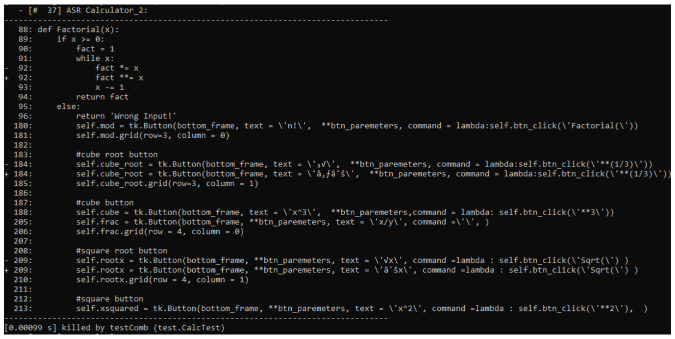
3) COI (Conditional Operator Insertion): Each occurrence of each logical operator (and-&&, or-——, and with no conditional evaluation-&, or with no conditional evaluation- —, not equivalent-ˆ) is inserted.



```
  - [#  47] COI Calculator_2:
--------------------------------------------------------------------------------
  136:     a = []
  137:     b = []
  138:     c = input('Enter the coeeficients of the polynomial in ascending powers of the variable separated by space: ')
  139:     a = c.split()
- 140:     if len(a) == 0:
+ 140:     if not (len(a) == 0):
  141:         return 0
  142:     for i in a:
  143:         b.append(float(i))
  144:     v = np.polynomial.Polynomial(b)
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â,ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00700 s] killed by test_Roots (test.CalcTest)
```

Fig. 4. Conditional Operator Insertion.

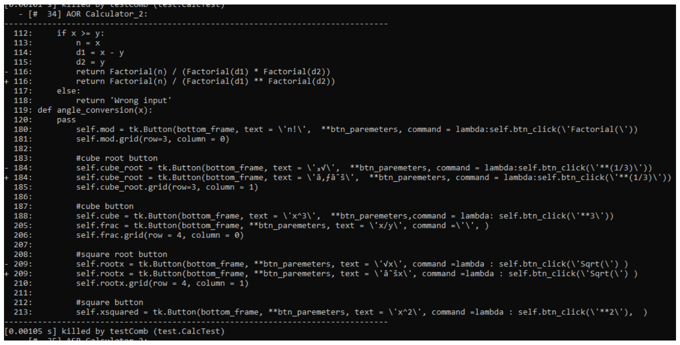
4) ASR (Assignment Operator Replacement): Each occurrence of one of the assignment operators (+=, -=, *=, ==, %=, &=, !=, ≙, ¡¡=, ¿¿=, ¿¿¿=) is replaced by each of the other operators.



```
  - [#  37] ASR Calculator_2:
--------------------------------------------------------------------------------
   88: def Factorial(x):
   89:     if x >= 0:
   90:         fact = 1
   91:         while x:
-  92:             fact *= x
+  92:             fact **= x
   93:             x -= 1
   94:         return fact
   95:     else:
   96:         return 'Wrong Input!'
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â,ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00099 s] killed by testComb (test.CalcTest)
```

Fig. 5. Assignment Operator Replacement.

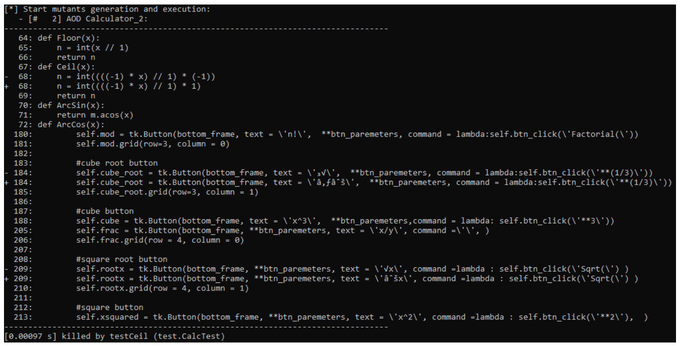
5) AOR (Arithmetic Operator Replacement): Each occurrence of one of the arithmetic operators +, -, *, /, ** and % is replaced by each of the other operators.



```
  - [#  34] AOR Calculator_2:
--------------------------------------------------------------------------------
  112:     if x >= y:
  113:         n = x
  114:         d1 = x - y
  115:         d2 = y
- 116:         return Factorial(n) / (Factorial(d1) * Factorial(d2))
+ 116:         return Factorial(n) / (Factorial(d1) ** Factorial(d2))
  117:     else:
  118:         return 'Wrong input'
  119: def angle_conversion(x):
  120:     pass
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â,ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00105 s] killed by testComb (test.CalcTest)
```

Fig. 6. Arithmetic Operator Replacement.

6) AOD (Arithmetic Operator Deletion): Each occurrence of one of the arithmetic operators +, -, *, /, ** and % is deleted.



```
[*] Start mutants generation and execution:
  - [#   2] AOD Calculator_2:
--------------------------------------------------------------------------------
   64: def Floor(x):
   65:     n = int(x // 1)
   66:     return n
   67: def Ceil(x):
-  68:     n = int((((-1) * x) // 1) * (-1))
+  68:     n = int((((-1) * x) // 1) * 1)
   69:     return n
   70: def ArcSin(x):
   71:     return m.acos(x)
   72: def ArcCos(x):
  180:         self.mod = tk.Button(bottom_frame, text = \'n!\',  **btn_paremeters, command = lambda:self.btn_click(\'Factorial(\'))
  181:         self.mod.grid(row=3, column = 0)
  182:
  183:         #cube root button
- 184:         self.cube_root = tk.Button(bottom_frame, text = \'₃√\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
+ 184:         self.cube_root = tk.Button(bottom_frame, text = \'â,ƒâˆš\',  **btn_paremeters, command = lambda:self.btn_click(\'**(1/3)\'))
  185:         self.cube_root.grid(row=3, column = 1)
  186:
  187:         #cube button
  188:         self.cube = tk.Button(bottom_frame, text = \'x^3\',  **btn_paremeters,command = lambda: self.btn_click(\'**3\'))
  205:         self.frac = tk.Button(bottom_frame, **btn_paremeters, text = \'x/y\', command =\'\', )
  206:         self.frac.grid(row = 4, column = 0)
  207:
  208:         #square root button
- 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'√x\', command =lambda : self.btn_click(\'Sqrt(\') )
+ 209:         self.rootx = tk.Button(bottom_frame, **btn_paremeters, text = \'âˆšx\', command =lambda : self.btn_click(\'Sqrt(\') )
  210:         self.rootx.grid(row = 4, column = 1)
  211:
  212:         #square button
  213:         self.xsquared = tk.Button(bottom_frame, **btn_paremeters, text = \'x^2\', command =lambda : self.btn_click(\'**2\'),  )
--------------------------------------------------------------------------------
[0.00097 s] killed by testCeil (test.CalcTest)
```

Fig. 7. Arithmetic Operator Deletion.

III. RESULTS

A total of 203 mutants were generated out of which 42 mutants were strongly killed with a Mutation Score of 22.2%. We were able to target 4 different classes of mutation operators, namely Relational, Conditional, Arithmetic, Assignment Operators, along with Constant Replacement. The detailed report for the same can be viewed in the HTML file namely "index.html" in the Mutation_Documentation folder.

REFERENCES

[1] Source Code: Python Scientific Calculator - https://github.com/samuelabidemi/Python-Scientific-Calculator.git.

[2] unittest: Unit testing framework - https://docs.python.org/3/library/unittest.html.

[3] Mutation Testing using Mutpy Module in Python - https://www.geeksforgeeks.org/mutation-testing-using-mutpy-module-in-python/.

[4] MutPy - https://pypi.org/project/MutPy/.