**Python and JavaScript
Function Testing Engine**(08/21/2022)

There are many types of programming errors that students struggle with. The most obvious are syntax errors which are easily detected by sophisticated IDEs and error messages that are displayed when programs are run. Logic errors are often more difficult for students to detect. Students don’t always think of or test the best test cases to uncover logic errors in their programs.

Grading programs is a labor-intensive responsibility for faculty members. It is tedious and error prone. Grading programs consistently and in a timely manner is often a challenge for faculty members.

The Python and JavaScript Function Testing Engine helps to solve both of these problems. When used by students, it can quickly detect logic errors in a student’s programs and direct them to look deeper into their programming logic and learn from that experience. When used by faculty members, this tool can quickly produce reports and spreadsheets with grades in a consistent and accurate format.

This project started in the Spring of 2019 when we first introduced JavaScript programming in our Intro to MIS course. It quickly became apparent that students would not really learn much about programming if we didn’t assign them graded programming assignments. It was also apparent that grading programming assignments consistently and in a timely manner would just be impossible at this scale. This is when I produced the first version of a node.js program that would grade hundreds and hundreds of JavaScript programming assignments in seconds. The core code from that system is at the foundation of this new Python and JavaScript Function Testing Engine.

# Functions Do All The Heavy-Lifting!

One idea at the core of this approach is that we teach students very early on that functions do all of the “heavy-lifting” in a program. While all programs have a “main” program, the “main” program basically does three things:

1. Collects some input from the user.
2. Calls one or more functions to perform calculations and other computations (a.k.a. the “heavy-lifting”.
3. Generates some output for the user.

If we can test the logic of each function in a program based on what we pass to the function and what it returns, we can test the logic of the entire program.

# Starter Files

Part of what makes all of this work is we provide students with “starter files” for each in-class programming activity, each assignment and each hands-on exam. These “starter files” include the “main” program which students do not touch. These “starter files” also include the function declarations along with strict instruction to not touch these function declarations. Finally, the naming convention for all starter files include having student rename the file to preface the file name with the students *lastname\_firstname*.

A typical “starter file” may look like:



# Test Cases

For each assigned program, the faculty member must create one or more test cases that will be used to invoke the functions in a program and compare the results returned by the function to what is expected to be returned by the function. Code never needs to be touched to add support for another program. All that needs to be done is add a new test case. Test cases are stored in a table in a DynamoDB table in AWS. A test case looks like:



In this test case we can see that we will evaluate the calculateTuition() function four times, the first time passing 11 (11 credits) and expecting to receive back 9691, the second time passing 12 and expecting to receive back 10572, the third time passing 18 and expecting to receive back 19572, and the last time passing 19 and expecting to receive back 11159.

The programs now support an “Input Queue”. In some cases, a function will need to prompt a user for input, so the system needs a way to script the entering of this input. For example, the following test case shows what a test case looks like when a function being tested requires scripted user input:



## Python Local Files

While it is very rare for anyone to ever want to do local file I/O in JavaScript, reading and writing local files is fairly common with Python so we also needed to support functions that read and/or write files. It is common for students in a Python class to need to learn how to open a file, read the contents, loop through the contents performing some processing and close a file. Processing could include adding up numbers in the file or performing string manipulations with data in a text file.

This system is built on a pair of AWS Lambda functions. Lambda is Amazon’s “serverless” technology. Being “serverless” there is no traditional file system where input files could be staged and then used as input by functions being tested. A Lambda function does have access to a relatively small “/tmp” file system. The issue with this is, being “serverless”, data stored in “/tmp” is not persistent. When a Lambda function is first invoked, files can be stored in “/tmp” and as long as the function is being invoked frequently, the data in “/tmp” appears to be persistent. However, at some point the function will become idle and removed from memory on the Lambda servers. The next time the Lambda function is invoked, “/tmp” is empty.

To deal with this issue, files that are needed by test cases are stored in S3. These test cases are updated to check for the existence of the needed files in “/tmp”, and if they are not fount, have the needed files downloaded from S3 to “/tmp”. This is not part of the Lambda functions, it is part of the test case.

Please see the test case for the “returnAverage” test case. A file called “steps.txt” contains 365 records. Each record includes a single number. These numbers represents the number of steps a person has taken each day for an entire year. The “returnAverage” function needs to open this file, count and add up all of the numbers in the file, close the file and return the average.

We see in the first part of the test case we check to see if “steps.txt” exists in “/tmp”. If the file doesn’t exist, we download the file from S3 before we start working our way though the various test cases. In the test case we invoke the “returnAverage” function, passing it the file “steps.txt”. Whatever is returned by “returnAverage” is converted to an int and then is compared to the number 5296 which is the average of all of the numbers in “steps.txt”. If the numbers match, the test case has been passed.



## Test Case Tips

Note that test cases are actually syntactically correct Python or JavaScript code. The test case text gets evaluated much like the students functions get evaluated and must not have any syntax errors of the Lambda function will return an error indicating that a syntax error was found. A quick test of a new test case with a good solution file will identify if you have any errors in a test case. Here are some additional tips for creating test cases:

1. Since this is a function testing engine, all functions that are tested must return a value that we can use to compare to what we expect the function to return. If the function does not return a value, it cannot be evaluated with this tool.
2. Test cases that test for numeric values returned by functions are usually the most simple and cleanest test cases to create. The most typical issue that is encountered is with functions that return floating point numbers and how many decimals are returned. For example, if I have a function that returns the value of Pi, am I expecting to receive 3.14 or 3.145 or 3.1459 or something else? This can usually be addressed by, in the use case, round the number returned by the function to a specific number of digits which will match the expected result. For example, in this case…



*Python Example*



*JavaScript Example*

1. Strings can also be problematic if the string that need to be returned is not very well defined. For example, in a game where the function returns a message that the player has won, one student may return “you won” while another may return “You won” while a third may return “You won!”. Again, if the strings are not well defined, a function returning a string can be a challenge.
2. In some cases strings are well defined. For example, if you had a function that returned the day of the week, there is only one correct way to spell “Thursday”. However, one student may return “thursday” another may return Thursday” and a third may return “THURSDAY”. In cases like this the safest thing to do is, using the proper Python or JavaScript method in the test script, convert what is returned by the function to upper case and compare that to a string that is already in upper case.



*JavaScript Example*

1. Finally, the last problem that we frequently encounter is when we have strings that return string and can have an extra space hanging on the end of the string so the string does not match what we are expecting (even if that is very difficult to see). Like we did with other test cases, using the proper Python or JavaScript method in the test script, strip any leading and trailing blanks from what is returned by the function before comparing the string to what you expect.



*Python Example*



*JavaScript Example*

# The System Itself

The system includes a single web page. The page is very simple. Towards the top of the page is some basic information about the types of programming errors and information about syntax errors vs. logic errors. This is followed by a pair of radio buttons where the user needs to specify if they are working with Python programs or JavaScript programs. This is followed with a button that allows the user to select one or more files from their computer to be evaluated.

 

The bottom part of the screen includes another pair of radio buttons to specify if you would like the output formatted as text that is easy to read or as in .CSV format. This is followed by a text area where the report is displayed. Finally, there is a button at the bottom that allows you to export the contents of the report to either a text file or a .CSV file.



A complete section of the text report for the TuitionCost.py program is shown here:



A complete .CSV report for all four of the programs that were evaluated is shown here:



When the “Test Programs” button is pressed, each file is read in one at a time and an API is invoked which executes an AWS Lambda function to evaluate the functions in that file against the list of test cases and returns the results.

# Use Cases

There are two major use cases for this system.

## Student Identifying Logic Errors

In this use case students can use this tool to evaluate the functions in their programs against a set of well-constructed test cases defined by their professor. If their program fails one or more test cases, they know that they need to dig into their programming logic further and discover the flaw in their logic. This process of digging in deeper to discover flaws is where some of the most important programming and general critical thinking skills are developed!

Faculty Member Grading Programming Assignments
If programming assignments are not graded and not factored into a student’s grade, students will not do their work and will not learn. As described earlier, this is a very time-consuming and tedious task which is a real challenge to complete in a consistent and timely manner. When students upload properly used “starter files” and upload their work to a system like Canvas where the faculty member can download all assignments into a single folder, evaluating programming assignments is quick and simple.

The faculty member simply selects the language, selects all of the files, selects “Text” as the output format and presses the “Test Programs” buttons. Within seconds all of the programs have been evaluated against the set of test cases specified by the faculty member and the report is generated. The faculty member saves the report as a text file so that if students have questions about why they didn’t earn full credit, the faculty member can quickly see which test cases their programs passed and which test cases their programs failed. The faculty then changes the output format to “.CSV”, presses the “Test Programs” button again and saves the report as a .CSV file. With basic Excel skills, the faculty member can then sort and organize this data and enter grades into the grade book of their choice.

# Why this works

The system itself is relatively simple and elegant. It leverages self-modifying code to accomplish its goals. This explanation applies equally well for both Python programs and JavaScript programs, but we will use Python in the following examples. In this section we will explain what is going on and why this works.

Let’s start by looking at a very simple program that a student in an introductory programming course may need to write. In this example the student needs to write a simple function that I passed the temperature in Fahrenheit and needs to return the temperature in Celsius. Here is an example:



Note that when the Python interpreter runs this code, the function fahrenheitToCelsius is created in memory and can be invoked by other parts of the program. We can actually use the “print(fahrenheitToCelsius)” instruction to see where the function is loaded into memory and we can use the “inspect.getsource(fahrenheitToCelsius)” instruction to actually see the function that has been loaded into memory.



The “exec()” function in Python can be passed a string and will “execute” that string as if it were code included in the Python program. While this is an unnatural act for most traditional programs, in the following example we create a string called “someString” and assign it a value that looks just like a function declaration for the “fahrenheitToCelsius()” function. We then invoke “exec(someString)” which defines this function in our memory space, just like it eas a function declaration included in out .py file. The “print(fahrenheitToCelsius)” instruction shows us that this function has been created in memory and is ready to be called.



The “exec()” function provides the foundation upon what this entire system is built on. We will show how we utilize this ability to take a string that defines a function, to load that function into our own memory space the ability to invoke that function in the next section.

Please note the two lines circled in green above. This is very typical way for a student in an introductory programming course to invoke a function passing arguments to the function and then processing the results returned by the function. In the next section we will show you how self-modifying code can be used in an innovative way to test a function against a set of test cases.

Take a look at the following two “if” statements. The “eval()” function, similar to the “exec()” function take a string and executes the string as a line of executable code. In this case, “fahrenheitToCelsius(32) == 0” and “fahrenheitToCelsius(212) == 100” are valid Boolean expressions and they are “True” if our “fahrenheitToCelsisus()” function is passed 32 and returns the value 0 and if our “fahrenheitToCelsisus()” function is passed 212 and returns the value 100. If the function does not work as expected and does not return the expected results, the Boolean expression would evaluate to “False”.



There is a logic error in the following example, (should be “\* (5/9) not + (5/9)”. We can see that due to this logic error, we now fail both test cases.



This technique of using self-modifying code to run test cases is also at the core of why this system works.