Lab Manual- 4

Mutation Testing

Mutation testing, also known as code mutation testing, is a form of white box testing in which testers change specific components of an application's source code to ensure a software test suite will be able to detect the changes. Changes introduced to the software are intended to cause errors in the program. Mutation testing can be performed after unit testing.

Unit testing is a common practice where developers write test cases together with regular code. Developers develop the products using programming languages such as Java, JavaScript, C#, and so on. As JavaScript is familiar to all, we write and test our unit testing using JavaScript. Besides, for every programming language, there are many testing frameworks. Here, For JavaScript, we use the Jest testing framework. Jest is a JavaScript testing framework designed to ensure the correctness of any JavaScript codebase. It allows you to write tests with an approachable, familiar, and feature-rich API that gives you results quickly. Jest is well-documented, requires little configuration, and can be extended to match your requirements.

Bugs, or mutants, are automatically inserted into your production code. Your tests are run for each mutant. If your tests fail, then the mutant is killed. If your tests passed, the mutant survived. The higher the percentage of mutants killed, the more effective your tests are.

To understand the mutation testing, we consider the previous example shown in Lab-3: Unit testing named Calculator where there are two types of calculators, Basic and Advanced calculators. The basic calculator has the following functionalities-

- Add(a, b): It takes two numbers as input and returns the summation (a+b) of these two numbers.
- Subtract(a, b): It takes two numbers as input and returns the subtraction (a-b) of these two numbers.
- Multiply(a, b): It takes two numbers as input and returns the multiplication value (a*b) of these two numbers.
- Divide(a, b): It takes two numbers, dividend and divisor as input and returns the quotient (a/b) of these two numbers.

The advanced calculator has the following functionalities-

- Pow(x, n): It takes two numbers as input and returns the powered value (x^n) of these two numbers.
- Modulo(a, b): It takes two numbers as input and returns the modulo value (a%b) of these two numbers.

To test the calculator, at first, we have to implement the Basic and Scientific calculator. As mentioned earlier, we will implement the calculator using JavaScript. Before implanting it, we need to install some libraries.

Now, we implement the Basic and Advanced calculator, design test cases, write the corresponding code to validate the test cases whatever we have performed in Lab-3: Unit testing.

Prerequisite: Lab-3: Unit Testing

Perform Mutation Testing:

Now, the environment is ready for mutation testing as we have implemented basic and advanced calculators, written unit testing code with test suites. To perform the mutation testing, we use Stryker that tests the test cases. It's easy to use and fast to run. Stryker will only mutate your source code, making sure there are no false positives.

At first, we install the Stryker by executing the following command

```
npm install -g stryker-cli
```

Then, we need to configure the Stryker. Run this command to configure Stryker.

```
stryker init
```

After executing the command, it is asked to install the Stryker, choose YES. Follow the Questionnaire.

```
? Do you want to install Stryker locally?: npm
```

```
? Are you using one of these frameworks? Then select a preset configuration. None/other
```

```
? Which test runner do you want to use? If your test runner isn't listed here, you can choose "command" (it uses your npm testcommand, but will come with a big performance penalty) jest
```

```
? Which reporter(s) do you want to use? html, clear-text, progress. Note: Use spacebar for multiple selection or choose html and press enter. HIT ENTER
```

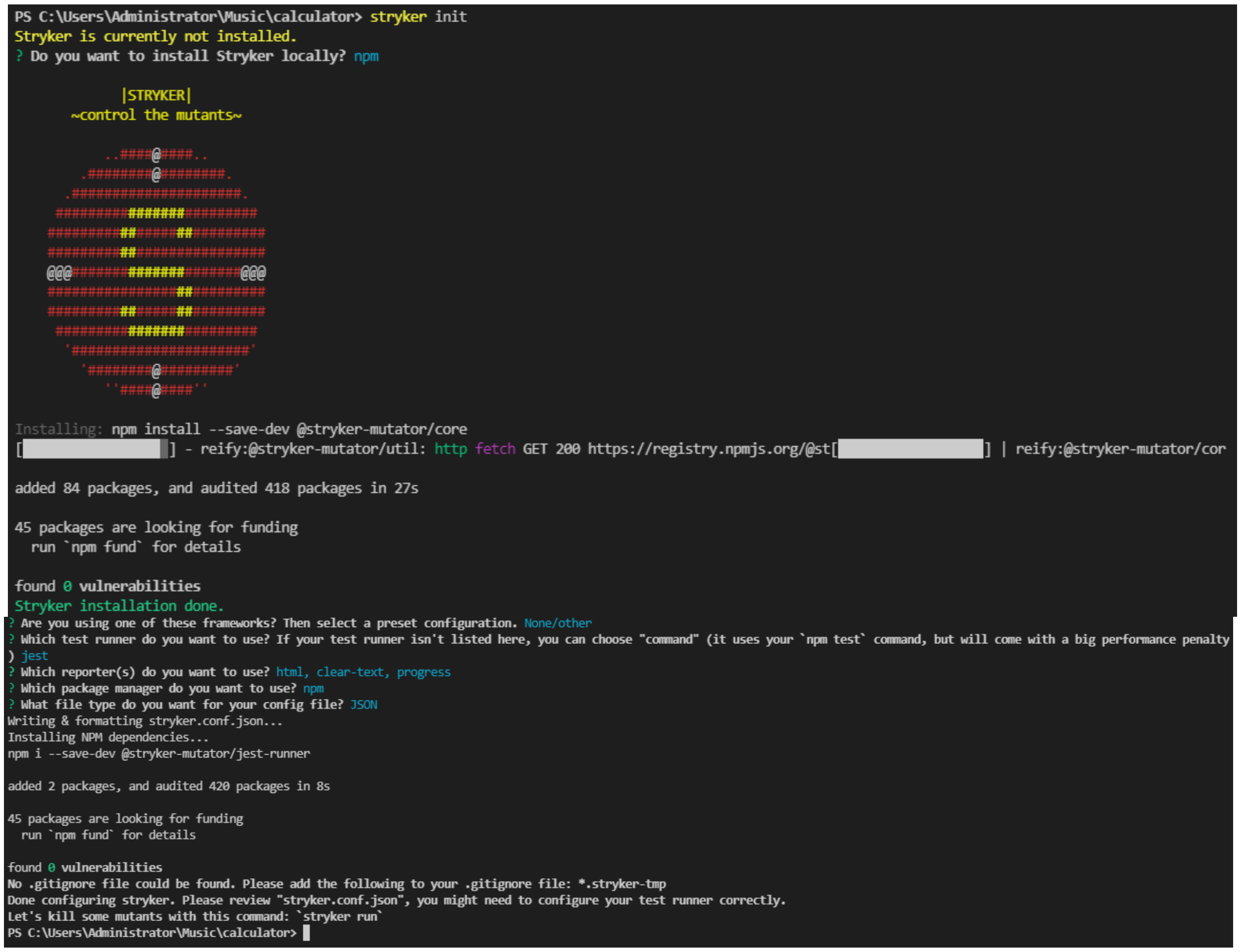
```
? Which package manager do you want to use? Npm
```

```
? What file type do you want for your config file? JSON
```

```
? What kind of code do you want to mutate? javascript
```

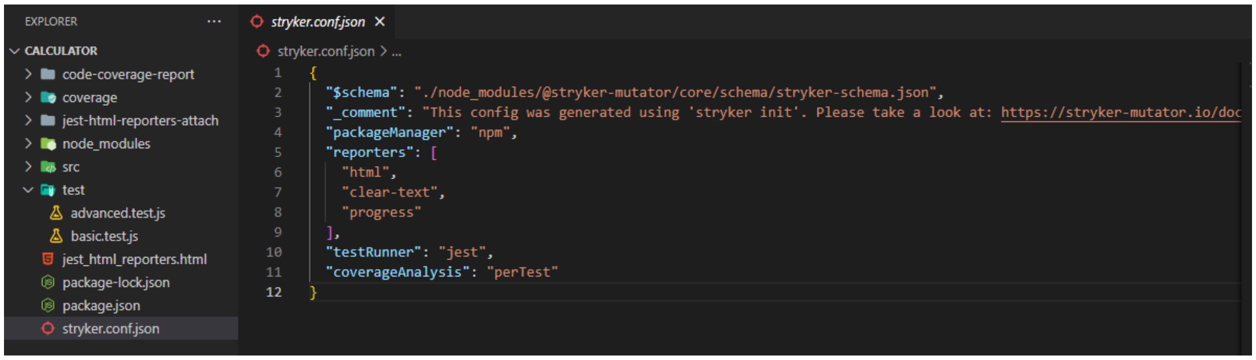
```
? [optional] What kind transformations should be applied to your code? (Press to select, to toggle all, to invert selection)
```

The console looks the following-



```
PS C:\Users\Administrator\Music\calculator> stryker init
Stryker is currently not installed.
? Do you want to install Stryker locally? npm

             |STRYKER|
       ~control the mutants~

            ..####@####..
         .########@########.
       .#####################.
      #########@@@@@@@#########
     #########@@#####@@#########
     #########@@################
     @@@#######@@@@@@@#######@@@
     ################@@#########
     #########@@#####@@#########
      #########@@@@@@@#########
       '#####################'
         '########@#########'
            ''####@####''

Installing: npm install --save-dev @stryker-mutator/core
[█████████████] - reify:@stryker-mutator/util: http fetch GET 200 https://registry.npmjs.org/@st[█████████████] | reify:@stryker-mutator/cor

added 84 packages, and audited 418 packages in 27s

45 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Stryker installation done.
? Are you using one of these frameworks? Then select a preset configuration. None/other
? Which test runner do you want to use? If your test runner isn't listed here, you can choose "command" (it uses your `npm test` command, but will come with a big performance penalty
) jest
? Which reporter(s) do you want to use? html, clear-text, progress
? Which package manager do you want to use? npm
? What file type do you want for your config file? JSON
Writing & formatting stryker.conf.json...
Installing NPM dependencies...
npm i --save-dev @stryker-mutator/jest-runner

added 2 packages, and audited 420 packages in 8s

45 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
No .gitignore file could be found. Please add the following to your .gitignore file: *.stryker-tmp
Done configuring stryker. Please review "stryker.conf.json", you might need to configure your test runner correctly.
Let's kill some mutants with this command: `stryker run`
PS C:\Users\Administrator\Music\calculator>
```

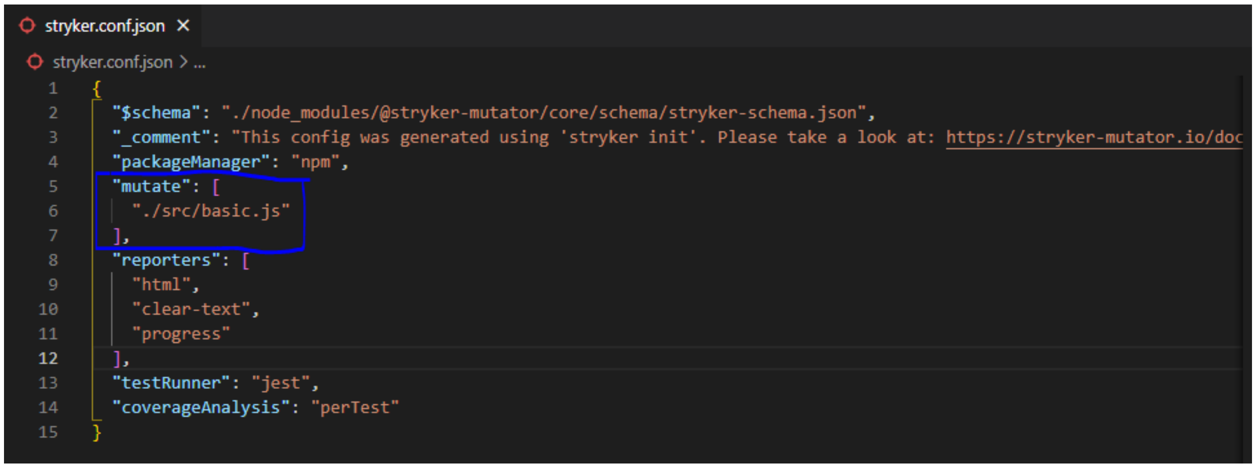
After the `init` is done, inspect the `stryker.conf.json` file. The contents would be the following.



```json
{
  "$schema": "./node_modules/@stryker-mutator/core/schema/stryker-schema.json",
  "_comment": "This config was generated using 'stryker init'. Please take a look at: https://stryker-mutator.io/doc
  "packageManager": "npm",
  "reporters": [
    "html",
    "clear-text",
    "progress"
  ],
  "testRunner": "jest",
  "coverageAnalysis": "perTest"
}
```

Now, it's time to perform mutation testing. As mentioned earlier, before mutation testing, we have to perform unit testing with the test suite. MAKE SURE, ALL TEST CASES ARE PASSED IN UNIT TESTING. IT IS MANDATORY TO SUCCESSFULLY PASS THE TEST CASES IN UNIT TESTING BEFORE RUNNING THE MUTATION TEST.

Assume, all test cases are successfully passed in unit testing. Then, suppose, at first we want to perform mutation testing for the basic calculator. For doing so, we have to add the code snippet in the `stryker.conf.json` file.



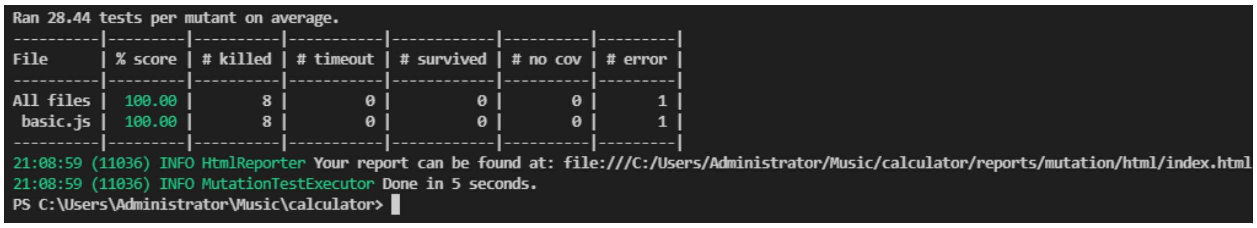
stryker.conf.json ×
stryker.conf.json > ...

```json
{
  "$schema": "./node_modules/@stryker-mutator/core/schema/stryker-schema.json",
  "_comment": "This config was generated using 'stryker init'. Please take a look at: https://stryker-mutator.io/doc
  "packageManager": "npm",
  "mutate": [
    "./src/basic.js"
  ],
  "reporters": [
    "html",
    "clear-text",
    "progress"
  ],
  "testRunner": "jest",
  "coverageAnalysis": "perTest"
}
```

Now, run the mutation test on the basic calculator.

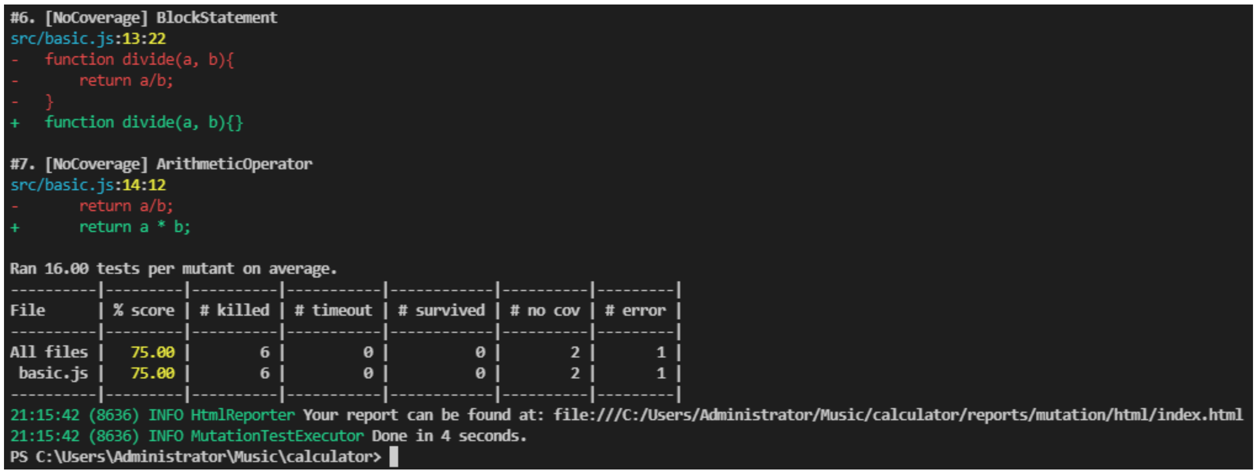
`stryker run`

It shows the following gist result on the console.



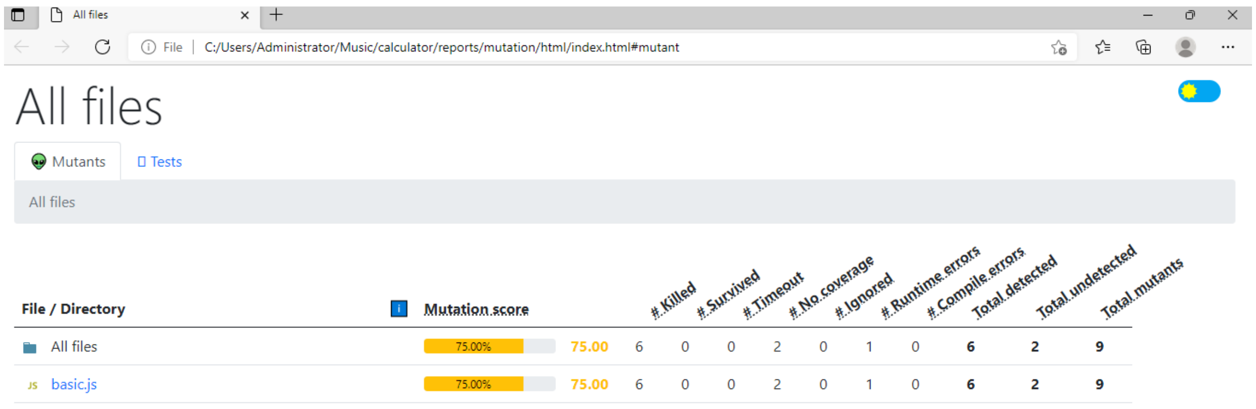
```
Ran 28.44 tests per mutant on average.
----------|---------|----------|-----------|------------|----------|---------|
File      | % score | # killed | # timeout | # survived | # no cov | # error |
----------|---------|----------|-----------|------------|----------|---------|
All files |  100.00 |        8 |         0 |          0 |        0 |       1 |
 basic.js |  100.00 |        8 |         0 |          0 |        0 |       1 |
----------|---------|----------|-----------|------------|----------|---------|
21:08:59 (11036) INFO HtmlReporter Your report can be found at: file:///C:/Users/Administrator/Music/calculator/reports/mutation/html/index.html
21:08:59 (11036) INFO MutationTestExecutor Done in 5 seconds.
PS C:\Users\Administrator\Music\calculator> 
```

Here, all the functionalities are tested using the test cases. Suppose, if we don't test a function of 4 functions, it shows the following-



```
#6. [NoCoverage] BlockStatement
src/basic.js:13:22
-   function divide(a, b){
-       return a/b;
-   }
+   function divide(a, b){}

#7. [NoCoverage] ArithmeticOperator
src/basic.js:14:12
-       return a/b;
+       return a * b;

Ran 16.00 tests per mutant on average.
----------|---------|----------|-----------|------------|----------|---------|
File      | % score | # killed | # timeout | # survived | # no cov | # error |
----------|---------|----------|-----------|------------|----------|---------|
All files |   75.00 |        6 |         0 |          0 |        2 |       1 |
 basic.js |   75.00 |        6 |         0 |          0 |        2 |       1 |
----------|---------|----------|-----------|------------|----------|---------|
21:15:42 (8636) INFO HtmlReporter Your report can be found at: file:///C:/Users/Administrator/Music/calculator/reports/mutation/html/index.html
21:15:42 (8636) INFO MutationTestExecutor Done in 4 seconds.
PS C:\Users\Administrator\Music\calculator> 
```

It also generates a dashboard in HTML format.



All files

C:/Users/Administrator/Music/calculator/reports/mutation/html/index.html#mutant

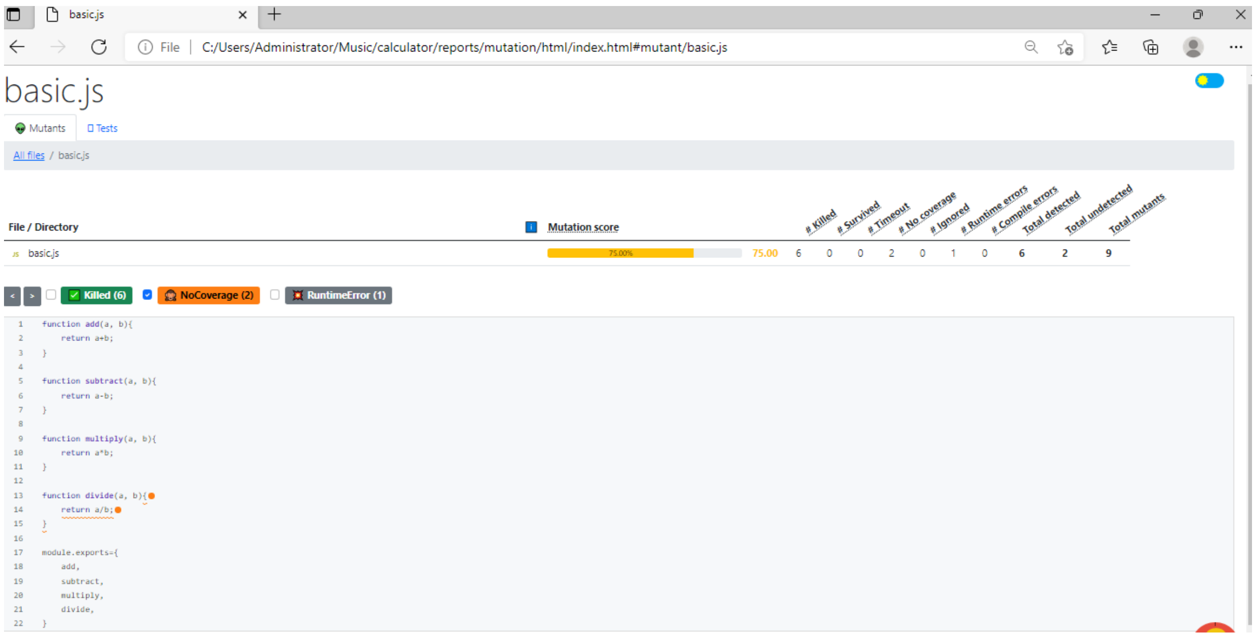
All files

Mutants | Tests

All files

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	75.00%	75.00	6	0	0	2	0	1	0	6	2	9
basic.js	75.00%	75.00	6	0	0	2	0	1	0	6	2	9

We can see specific files with mutants and their status.



basic.js

C:/Users/Administrator/Music/calculator/reports/mutation/html/index.html#mutant/basic.js

basic.js

Mutants | Tests

All files / basic.js

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
basic.js	75.00%	75.00	6	0	0	2	0	1	0	6	2	9

< > Killed (6) NoCoverage (2) RuntimeError (1)

```
function add(a, b){
    return a+b;
}

function subtract(a, b){
    return a-b;
}

function multiply(a, b){
    return a*b;
}

function divide(a, b){
    return a/b;
}

module.exports={
    add,
    subtract,
    multiply,
    divide,
}
```

In the same way, we can perform mutation testing on the advanced calculator. Just add the path to the `stryker.conf.json` file.

Lastly, if we want to perform mutation testing for the whole project, then remove the ***mutate part*** from the `stryker.conf.json` file and run the `stryker run`.

Tasks:

In the same way, perform **mutation testing** to all the unit test cases of basic and advanced calculators and **explain the dashboard**.

- Add(a, b): It takes two numbers as input and returns the summation (a+b) of these two numbers.

- Subtract(a, b): It takes two numbers as input and returns the subtraction (a-b) of these two numbers.
- Multiply(a, b): It takes two numbers as input and returns the multiplication value (a*b) of these two numbers.
- Divide(a, b): It takes two numbers, dividend and divisor as input and returns the quotient (a/b) of these two numbers.

The advanced calculator has the following functionalities-

- Pow(x, n): It takes two numbers as input and returns the powered value (x^n) of these two numbers.
- Modulo(a, b): It takes two numbers as input and returns the modulo value (a%b) of these two numbers.