**Scenarios**

1. Write Test Code Beginner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Original Code | Explanation | Mutation | Solution |
| 1 | **function** getOpposite(bool) {  **return** !bool; } | Function returns the opposite of a boolean value | Line 2 : return bool; | let bool = false;  const result = getOpposite(bool);  expect(result).toBe(true); |
| 2 | **function** getArea(side1, side2, side3){  **var** perimeter = (side1 + side2 + side3)/2;  **var** area = ***Math***.sqrt(perimeter\*((perimeter-side1)\*(perimeter-side2)\*(perimeter-side3)));  **return** area; } | Function returns the area of a triangle | Line 2: var perimeter = (side1 + side2 - side3)/2; | let side1 = 5;  let side2 = 6;  let side3 = 7;  const result = getArea(side1, side2, side3);  expect(result).toBe(14.696938456699069); |
| 3 | **function** checkSunday (year) {  **for** (**var** y = year; y <= 2050; y++)  {  **var** d = **new *Date***(y, 0, 1);  **if** ( d.getDay() === 0 ){  **return** (**"1st January is being a Sunday "**+y);  }  } } | Function logs the very first 1st January that is a Sunday between the year set to the year 2050. | Line 2: for (var y = 2014; y <= 2050; y++) | let year = 2045;  const result = checkSunday(year);  expect(result).toBe(**"1st January is being a Sunday 2045"**) |
| 4 | **function** guessNumber(gnum){  **var** num = ***Math***.ceil(***Math***.random() \* 10);  **if** (gnum == num) {  **return** (**"Matched"**);}  **else** {  **return** (**"Not matched"**);  }  } | Function lets user guess number between one and ten and matches if correct. | Line 5-7: Removed | let gnum = 11;  const result = guessNumber(gnum);  expect(result).toBe("Not matched"); |
| 5 | **function** setVariable(varName){ **var** n = 120; **this**[varName] = n; **return** (**this**[varName]) } | User chooses variable name and stores the number 120 inside the variable. | Line 2: var n = “120” | let varName = ‘egg’;  const result = setVariable(varName);  expect(result).toBe(120); |
| 6 | **function** test50(x, y) {  **if** ((x == 50 || y == 50) || (x + y == 50)){  **return** (**"Success"**);  } } | Checks two inputs If they contain 50 or if the sum of them are 50. | Line 3 : if ((x != 50 || y == 50) || (x + y == 50)){ | let x = 50;  let y=6;  const result = test50(x, y);  expect(result).toBe("Success"); |
| 7 | function start\_spec\_str(str)  {  if (str.length < 4)  {  return false;  }  front = str.substring(0, 4);  if (front == 'Java')  {  return true;  }  else  {  return false;  }  } | Function checks if a string starts with 'Java' and false otherwise. | Line 12 – 15 : Removed | let str = 'eggs';  const result = start\_spec\_str(str);  expect(result).toBe(false); |
| 8 | function capital\_letter(str)  {  str = str.split(" ");  for (var i = 0, x = str.length; i < x; i++) {  str[i] = str[i][0].toUpperCase() + str[i].substr(1);  }  return str.join(" ");  } | Function capitalizes the first letter of each word of a given string | Line 4: for (var i = 0, x = str.length; i > x; i++) { | let str = 'this is a test';  const result = capital\_letter(str);  expect(result).toBe('This Is A Test'); |
| 9 | function first\_half (str) {  if (str.length % 2 == 0) {  return str.slice(0, str.length / 2);  }  return str;  } | Function extracts the first half of a string of even length | Line 3: return str.slice(0, str.length); | let str = 'testings';  const result = first\_half(str);  expect(result).toBe('test'); |
| 10 | function sumn(val) {  var sn = 0;  var i = 0;  while (sn <= val) {  sn += i++;  }  return i - 2;  } | Function that finds the maximum integer n such that 1 + 2 + ... + n <= a given integer. | Line 5: sn = i++; | let val = 11;  const result = sumn(val);  expect(result).toBe(4); |

1. Write Test Code – Intermediate (No explanation of mutation) 1 mutation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Original Code | Explanation | Mutation | Solution |
| 1 | **function** *arraydigits*(n) {  n = [...**`**${n}**`**].map(i => *parseInt*(i));  **return** n; } | Function converts a specified number to an array of digits. | Line 2: n = [...`${n}`].map(i => *parseInt*(n)); | let n = 123;  const result = arraydigits(n);  expect(result).toEqual([1,2,3]); |
| 2 | **function** *random\_hex\_color\_code* (){  **let** n = (***Math***.random() \* 0xfffff \* 1000000).toString(16);  **return '#'** + n.slice(0, 6); } | Function generates a random hexadecimal color code. | Line 3: return '#' + n.slice(0, 5); | const result = *random\_hex\_color\_code* ();  expect(result.length).toBe(7); |
| 3 | **function** *yes\_No* (val, def = **false**) {  **return** /^(y|yes)$/i.test(val) ? **true** : /^(n|no)$/i.test(val) ? **false** : def; } | Function returns true if the string is y/yes or false if the string is n/no. | Line 2: return /^(yes)$/i.test(val) ? true : | let val = 'Y';  const result = yes\_No(val);  expect(result).toBe(true); |
| 4 | **function** *pad* (str, length, char = **' '**){  **return** str.padStart((str.**length** + length) / 2, char).padEnd(length, char); } | Function pads a string on both sides with the specified character, if it's shorter than the specified length. | Line 2 : return str.padStart((str.length + length), char).padEnd(length, char); } | let str = 'cat';  let length = 8;  const result = pad(str, length);  expect(result).toBe  (' cat '); |
| 5 | **var** *get\_Meridiem\_Suffix\_Of\_Integer* = num =>  num === 0 || num === 24  ? 12 + **'am'** : num === 12  ? 12 + **'pm'** : num < 12  ? (num % 12) + **'am'** : (num % 12) + **'pm'**; | Program converts an integer to a suffixed string, adding am or pm based on its value. | Line 5: ? 12 + 'am' | let n = 12;  let result = *get\_Meridiem\_Suffix\_Of\_Integer* (n);  expect(result).toBe('12pm'); |

1. Write Test Code – Advanced (2 mutations)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Original Code | Explanation | Mutation | Solution |
| 1 | **function** *biggerNumber*(num1, num2) {  **if** (*parseInt*(num1, 10) > *parseInt*(num2, 10)) {  **return**(**"The larger of "** + num1 + **" and "** + num2 + **" is "** + num1 + **"."**);  }  **else if** (*parseInt*(num2, 10) > *parseInt*(num1, 10)) {  **return**(**"The larger of "** + num1 + **" and "** + num2 + **" is "** + num2 + **"."**);  }  **else** {  **return**(**"The values "** + num1 + **" and "** + num2 + **" are equal."**);  } } | Function accepts two integers and display the larger. | Line 3: return("The larger of " + num2 + "  Line 4: return("The larger of " + num1 + " and " + num2 + " is " + num1 | let num1 = 5;  let num2 = 7;  const result = biggerNumber(num1,num2);  expect(result).toBe('The larger of 5 and 7 is 7.');  let num1 = 7;  let num2 = 5;  const result = biggerNumber(num1,num2);  expect(result).toBe('The larger of 7 and 5 is 7.'); |
| 2 | **var** *averageBy* = (arr, fn = 'a') =>  arr.map(**typeof** fn === **'function'** ? fn : val => val[fn]).reduce((acc, val) => acc + val, 0) /  arr.**length**; | program to compute the average of an array, after mapping each element to a value using the provided function. |  | let arr = [{ a: 4 }, { a: 2 }, { a: 8 }, { a: 6 }];  const result = averageBy(arr);  expect(result).toBe(5); |
| 3 | **function** *clone* (regExp1) {  **const** *cloneRegExp* = regExp => **new *RegExp***(regExp.**source**, regExp.flags);  **const** regExp = regExp1;  **const** regExp2 = *cloneRegExp*(regExp);  **return** regExp2; } | program to clone a given regular expression. | Line 5: const regExp = regExp;  Line 6: return regExp2; | let input = /lorem ipsum/gi;  const result = String(clone(input));  expect(result).toBe('/lorem ipsum/gi'); |

1. Restore the original code - Beginner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Original Code | Explanation | Mutation | Solution |
| 1 | **function** *fileType*(filename){  **return** filename.split(**'.'**).pop() }; | Function to get the extension of a filename. | Line 2: return filename.split('h').pop() | let filename = "system.php";  const result = fileType(filename);  expect(result).toBe('php'); |
| 2 | **function** *testhundred*(x) {  **return** ((***Math***.abs(100 - x) <= 20) ||  (***Math***.abs(400 - x) <= 20)); } | Function to check a given integer is within 20 of 100 or 400. | Line 2: return ((*Math*.abs(100 + x) <= 20) || | let x = 90  const result = testhundred(x);  expect(result).toBe(true); |
| 3 | **function** *time\_convert*(num){  **var** hours = ***Math***.floor(num / 60);  **var** minutes = num % 60;  **return** hours + **":"** + minutes; } | Function to convert a given number to hours and minutes. | Line 4: return hours + minutes; | let num = 450;  const result = time\_convert(num);  expect(result).toBe('7:30'); |
| 4 | **function** *end\_script*(str) {  **if** (str.substring(str.**length** - 6, str.**length**) == **'Script'**)  {  **return true**;  }  **else** {  **return false**;  } } | Function to test if a string end with "Script". The string length must be greater or equal to 6. | Line 4: return false; | let str = "JavaScript"  const result = end\_script(str);  expect(result).toBe(true); |
| 5 | **function** *city\_name*(str) {  **if** (str.**length** >= 3 && ((str.substring(0, 3) == **'Los'**)  || (str.substring(0, 3) == **'New'**)))   {  **return** str;  }   **return ''**; } | Function  to display the city name if the string begins with "Los" or "New" otherwise return blank. | Line 2: if (str.length >= 4 && ((str.substring(0, 3) == 'Los') | let str = "New"  const result = city\_name(str);  expect(result).toBe("New"); |
| 6 | **function** *sum\_three*(nums){  **return** nums[0] + nums[1] + nums[2]; } | Function to compute the sum of three elements of a given array of integers of length 3. | Line 2: return nums[1] + nums[1] + nums[2]; | let nums = [10, 32, 20];  const result = sum\_three(nums);  expect(result).toBe(62); |
| 7 | **function** *contains13*(nums) {  **if** (nums.indexOf(1) != -1 || nums.indexOf(3) != -1){  **return true** }  **else** {  **return false** } } | Function to test if an array of integers of length 2 contains 1 or a 3. | Line 2: if (nums.indexOf(1) == -1 || | let nums = [1, 5];  const result = contains13(nums);  expect(result).toBe(true); |
| 8 | **function** *twice3040*(arra1) {  **let** a = arra1[0],  b = arra1[1];  **return** (a === 30 && b === 30) || (a === 40 && b === 40); } | Function to test if a given array of integers contains 30 or 40 twice. The array length should be 0, 1, or 2. | Line 3: return (a === 30 && b === 30) && | let arra1 = [30, 30];  const result = twice3040(arra1);  expect(result).toBe(true); |
| 9 | **function** *alphabet\_char\_Shift*(str) {  **var** all\_chars = str.split(**""**);  **for**(**var** i = 0; i < all\_chars.length; i++) {  **var** n = all\_chars[i].charCodeAt() - **'a'**.charCodeAt();  n = (n + 1) % 26;  all\_chars[i] = ***String***.fromCharCode(n + **'a'**.charCodeAt());  }  **return** all\_chars.join(**""**); } | Function to replace each character of a given string by the next one in the English alphabet. | Line 3: for(var i = 1; i < all\_chars.length; i++) { | let str = "abcdxyz";  const result = alphabet\_char\_Shift(str);  expect(result).toBe('bcdeyza'); |
| 10 | **function** *rearrangement\_characters*(str1, str2) {  **var** first\_set = str1.split(**''**),  second\_set = str2.split(**''**),  result = **true**;   first\_set.sort();  second\_set.sort();   **for** (**var** i = 0; i < ***Math***.max(first\_set.length, second\_set.length); i++) {  **if** (first\_set[i] !== second\_set[i]) {  result = **false**;  }  }  **return** result; } | Function  to check whether it is possible to rearrange characters of a given string in such way that it will become equal to another given string. | Line 4 : result = false; | let str1 = "xyz";  let str2 = "zyx";  const result = *rearrangement\_characters*(str1, str2);  expect(result).toBe(true); |

1. Restore the original code - Intermediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Original Code | Explanation | Mutation | Solution |
| 1 | **function** *checkMultiples*(num1, num2) {  **var** sum = 0;  **for** (**var** x = 0; x < 1000; x++) {  **if** (x % num1 === 0 || x % num2 === 0) {  sum += x;  }  }  **return** sum; } | Function to sum the multiples of two numbers under 1000. | Line 4: if (x % num1 === 0 && x % num2 === 0) {  Line 5: sum = x; | let num1 = 3;  let num2 = 5;  const result = checkMultiples(num1, num2);  expect(result).toBe(233168); |
| 2 | **function** *test\_prime*(n) {   **if** (n===1)  {  **return false**;  }  **else if**(n === 2)  {  **return true**;  }**else** {  **for**(**var** x = 2; x < n; x++)  {  **if**(n % x === 0)  {  **return false**;  }  }  **return true**;  } } | Function to check if the number is prime or not. | Line 3: if (n!=1)  Line 12: for(var x = 0; x < n; x++) | let n = 37;  const result = test\_prime(n);  expect(result).toBe(true); |
| 3 | **function** *unique\_char*(str1) {  **var** str=str1;  **var** uniql=**""**;  **for** (**var** x=0;x < str.**length**;x++)  {   **if**(uniql.indexOf(str.charAt(x))==-1)  {  uniql += str[x];   }  }  **return** uniql; } | Function to extract unique characters from a string. | Line 3 : var str=str;  Line 7: uniql = str[x]; | let str1 = "thequickbrownfoxjumpsoverthelazydog";  const result = unique\_char(str1);  expect(result).toBe( 'thequickbrownfxjmpsvlazydg'); |
| 4 | **var** *array\_sum* = **function**(my\_array) {  **if** (my\_array.**length** === 1) {  **return** my\_array[0];  }  **else** {  **return** my\_array.pop() + *array\_sum*(my\_array);  } }; | Function to compute the sum of an array of integers. | Line 3: return [1,2,3].pop() + *10;* | let my\_array = [1,2,3,4,5,6];  const result = array\_sum(my\_array);  expect(result).toBe(21); |
| 5 | **function** *current\_oddeven* (num){  **var** evenodd = **''**;  **for** (**var** x = num; x <= 15; x++) {  **if** (x === 0) {  evenodd += (x + **" is even. "**);  }  **else if** (x % 2 === 0) {  evenodd += (x + **" is even. "**);  }  **else** {  evenodd += (x + **" is odd. "**);  }  }  **return** evenodd; } | Function uses a for loop that will iterate from x to 15. For each iteration, it will check if the current number is odd or even, and display a message to the screen. | Line 3: for (var x = 0;  Line 5, 8, 11: return(x + | let x = 0;  const result = current\_oddeven(x);  expect(result).toBe('0 is even. 1 is odd. 2 is even. 3 is odd. 4 is even. 5 is odd. 6 is even. 7 is odd. 8 is even. 9 is odd. 10 is even. 11 is odd. 12 is even. 13 is odd. 14 is even. 15 is odd. '); |