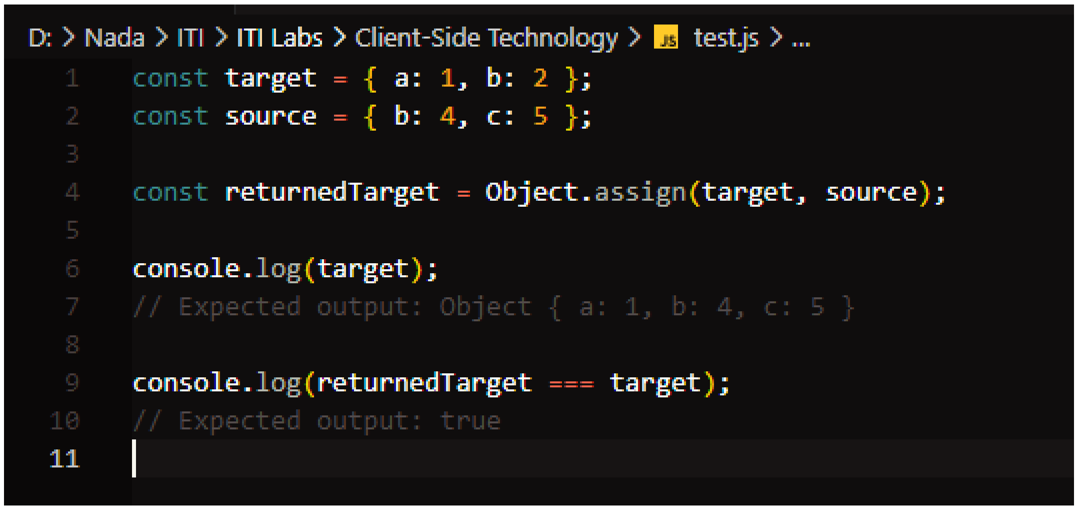
Table of Contents

Object Methods

1. Object.assign()

The Object.assign() static method copies all enumerable own properties from one or more source objects to a target object. It returns the modified target object.

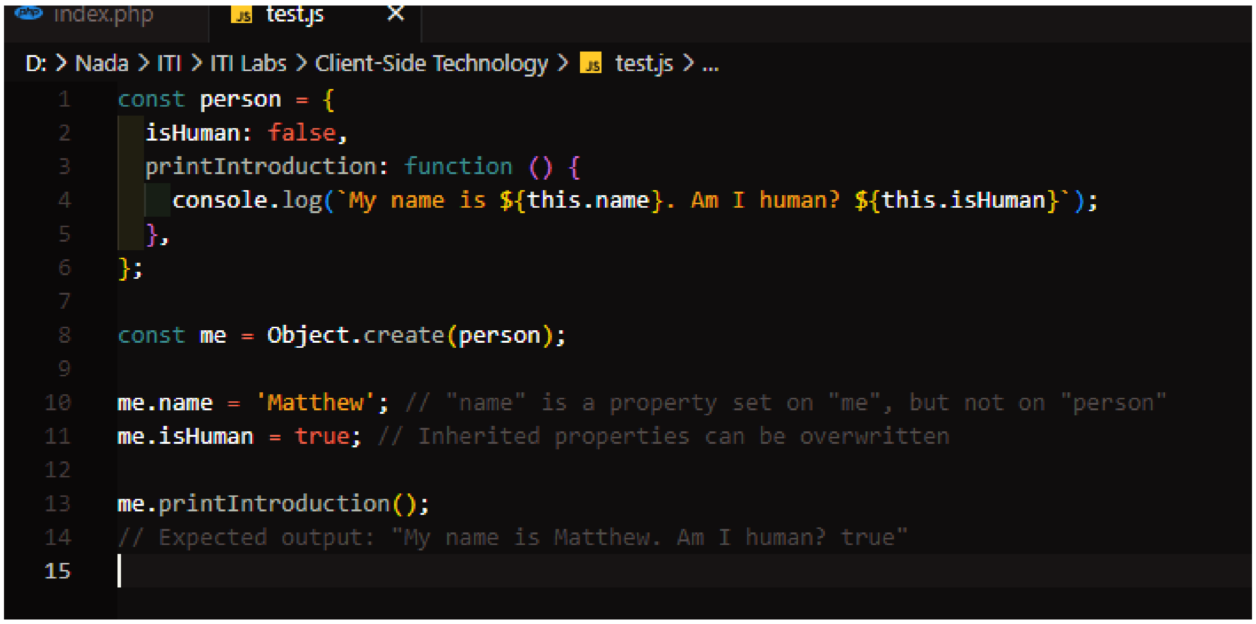


```
D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...
```

```javascript
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// Expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget === target);
// Expected output: true

```

2. Object.create()

The Object.create() static method creates a new object, using an existing object as the prototype of the newly created object.

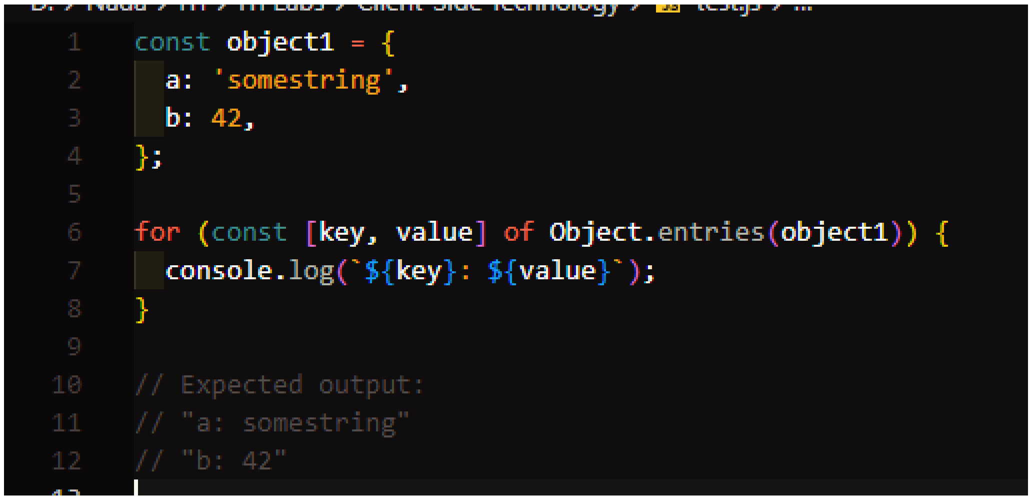


```
index.php   test.js ×
D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...
```

```javascript
const person = {
  isHuman: false,
  printIntroduction: function () {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  },
};

const me = Object.create(person);

me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // Inherited properties can be overwritten

me.printIntroduction();
// Expected output: "My name is Matthew. Am I human? true"

```

3. Object.entries()

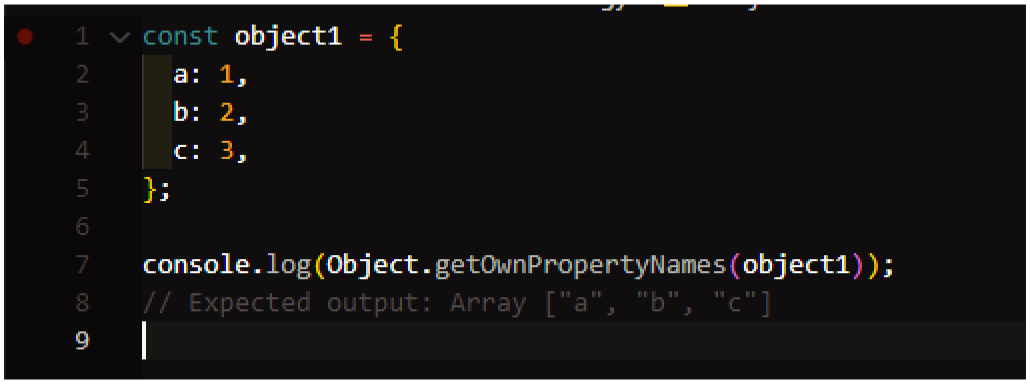
The Object.entries() static method returns an array of a given object's own enumerable string-keyed property key-value pairs.



```
const object1 = {
  a: 'somestring',
  b: 42,
};

for (const [key, value] of Object.entries(object1)) {
  console.log(`${key}: ${value}`);
}

// Expected output:
// "a: somestring"
// "b: 42"
```

4. Object. getOwnPropertyNames ()

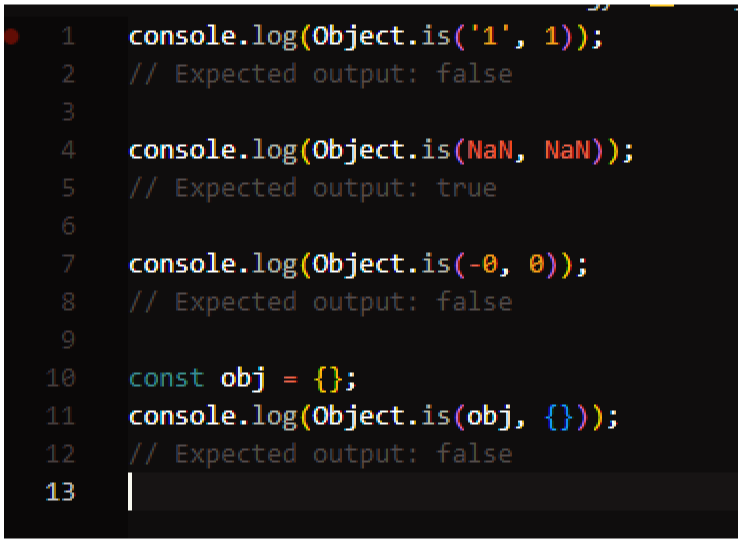
The Object.getOwnPropertyNames() static method returns an array of all properties (including non-enumerable properties except for those which use Symbol) found directly in a given object.



```
const object1 = {
  a: 1,
  b: 2,
  c: 3,
};

console.log(Object.getOwnPropertyNames(object1));
// Expected output: Array ["a", "b", "c"]
```

5. Object.is()

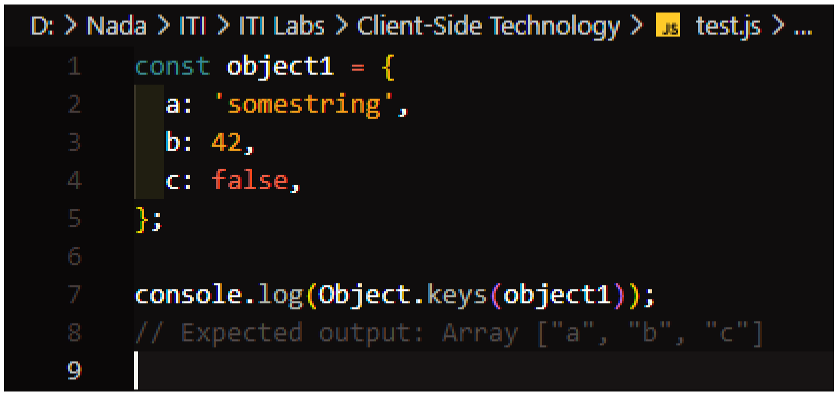
The Object.is() static method determines whether two values are the same value.



```
console.log(Object.is('1', 1));
// Expected output: false

console.log(Object.is(NaN, NaN));
// Expected output: true

console.log(Object.is(-0, 0));
// Expected output: false

const obj = {};
console.log(Object.is(obj, {}));
// Expected output: false

```

6. Object.keys()

The Object.keys() static method returns an array of a given object's own enumerable string-keyed property names.

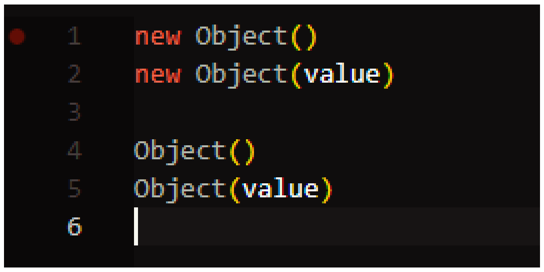


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false,
};

console.log(Object.keys(object1));
// Expected output: Array ["a", "b", "c"]

```

7. Object.object()

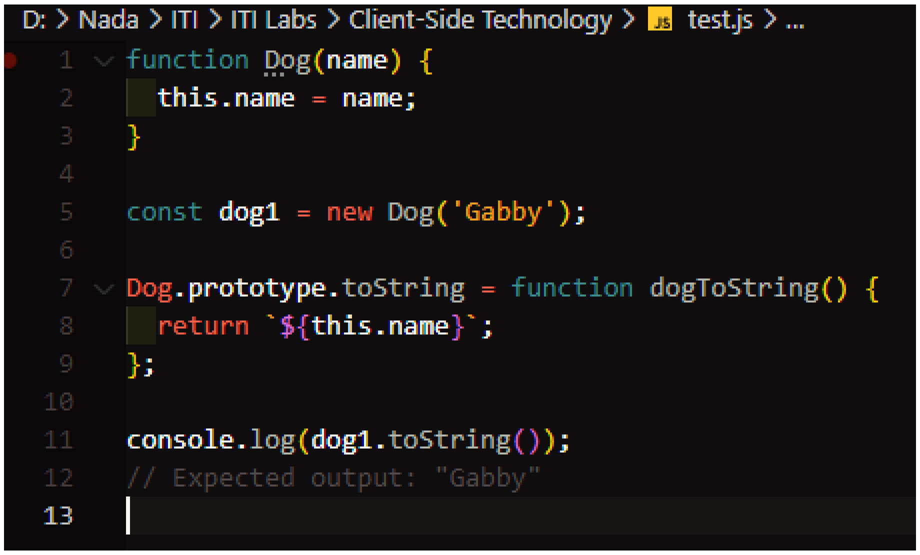
The Object() constructor turns the input into an object. Its behavior depends on the input's type.



```
new Object()
new Object(value)

Object()
Object(value)

```

8. Object.tostring()

The toString() method of Object instances returns a string representing this object. This method is meant to be overridden by derived objects for custom type coercion logic.

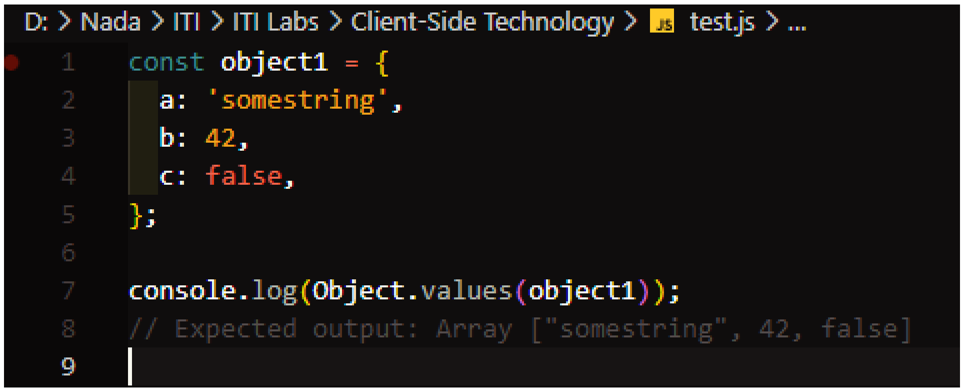


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
function Dog(name) {
  this.name = name;
}

const dog1 = new Dog('Gabby');

Dog.prototype.toString = function dogToString() {
  return `${this.name}`;
};

console.log(dog1.toString());
// Expected output: "Gabby"

```

9. Object.values()

The Object.values() static method returns an array of a given object's own enumerable string-keyed property values.

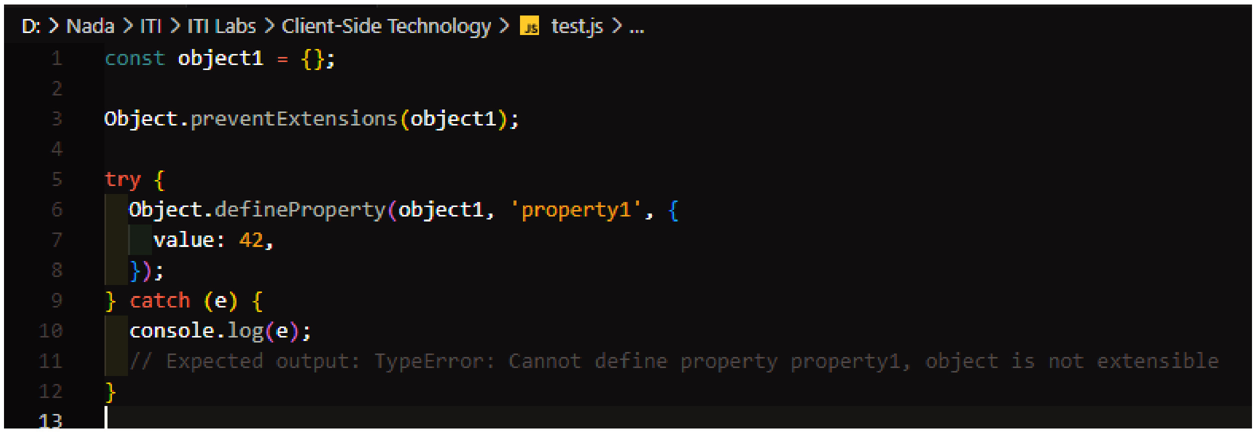


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const object1 = {
  a: 'somestring',
  b: 42,
  c: false,
};

console.log(Object.values(object1));
// Expected output: Array ["somestring", 42, false]

```

10.Object.preventExtensions()

The Object.preventExtensions() static method prevents new properties from ever being added to an object (i.e. prevents future extensions to the object). It also prevents the object's prototype from being re-assigned.

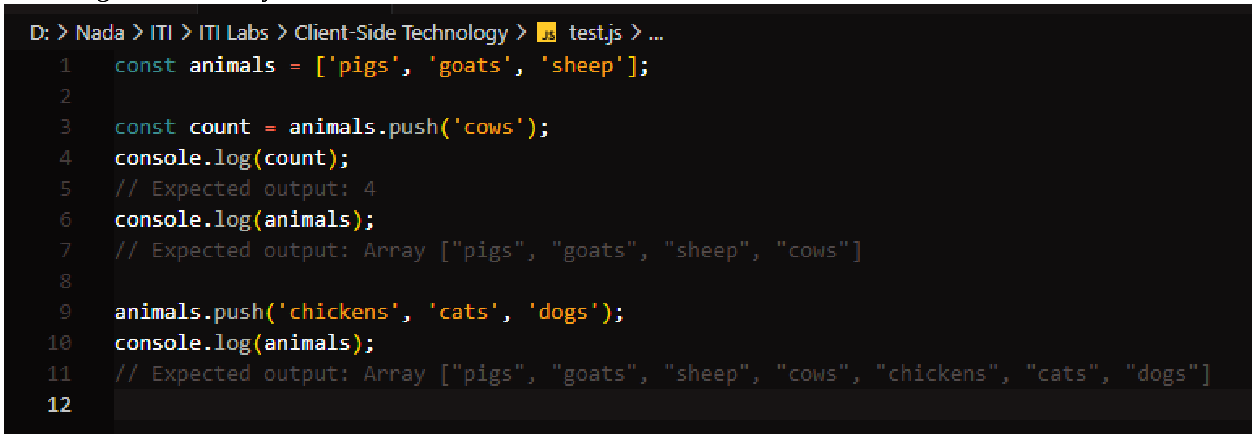

D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const object1 = {};

Object.preventExtensions(object1);

try {
  Object.defineProperty(object1, 'property1', {
    value: 42,
  });
} catch (e) {
  console.log(e);
  // Expected output: TypeError: Cannot define property property1, object is not extensible
}

```

Array Methods

1. Push()

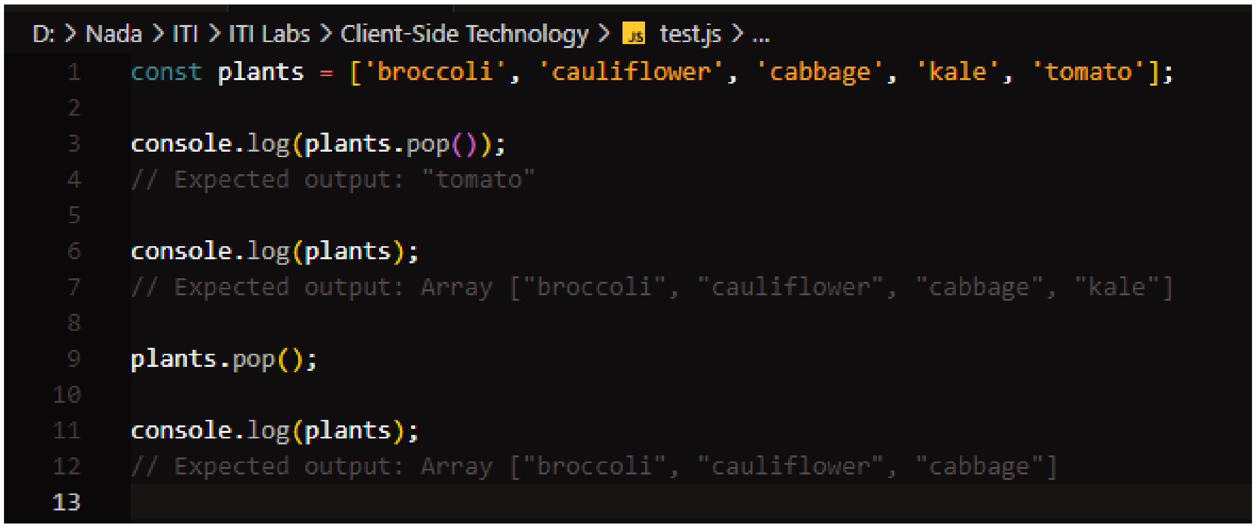
The push() method of Array instances adds the specified elements to the end of an array and returns the new length of the array.


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const animals = ['pigs', 'goats', 'sheep'];

const count = animals.push('cows');
console.log(count);
// Expected output: 4
console.log(animals);
// Expected output: Array ["pigs", "goats", "sheep", "cows"]

animals.push('chickens', 'cats', 'dogs');
console.log(animals);
// Expected output: Array ["pigs", "goats", "sheep", "cows", "chickens", "cats", "dogs"]

```

2. pop()

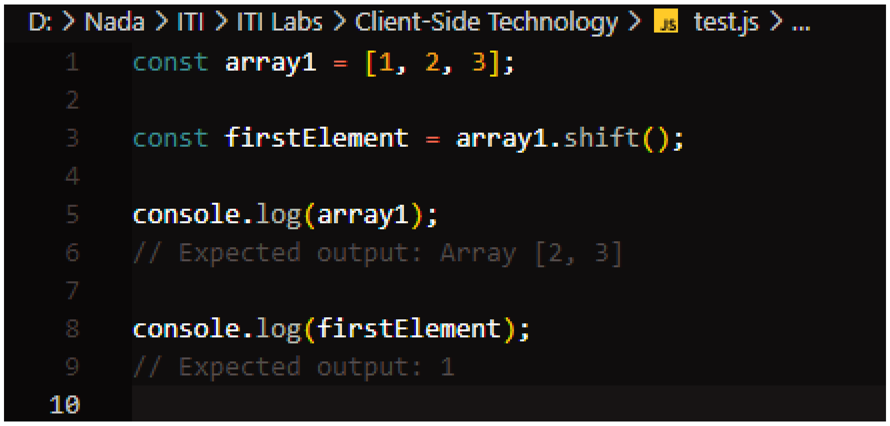
The pop() method of Array instances removes the last element from an array and returns that element. This method changes the length of the array.



```
D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...
const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());
// Expected output: "tomato"

console.log(plants);
// Expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]

plants.pop();

console.log(plants);
// Expected output: Array ["broccoli", "cauliflower", "cabbage"]

```

3. shift()

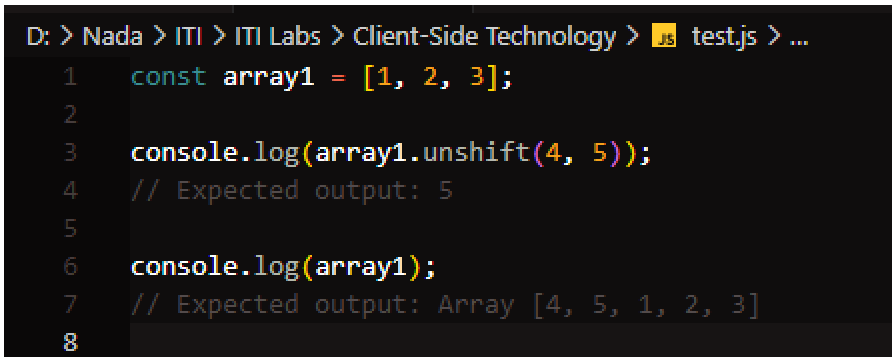
The shift() method of Array instances removes the first element from an array and returns that removed element. This method changes the length of the array.



```
D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...
const array1 = [1, 2, 3];

const firstElement = array1.shift();

console.log(array1);
// Expected output: Array [2, 3]

console.log(firstElement);
// Expected output: 1

```

4. unshift()

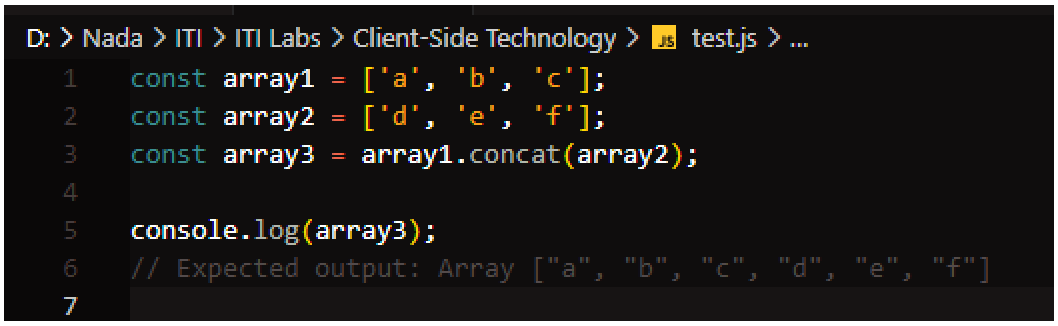
The unshift() method of Array instances adds the specified elements to the beginning of an array and returns the new length of the array.



```
D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...
const array1 = [1, 2, 3];

console.log(array1.unshift(4, 5));
// Expected output: 5

console.log(array1);
// Expected output: Array [4, 5, 1, 2, 3]

```

5. concat()

The concat() method of Array instances is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

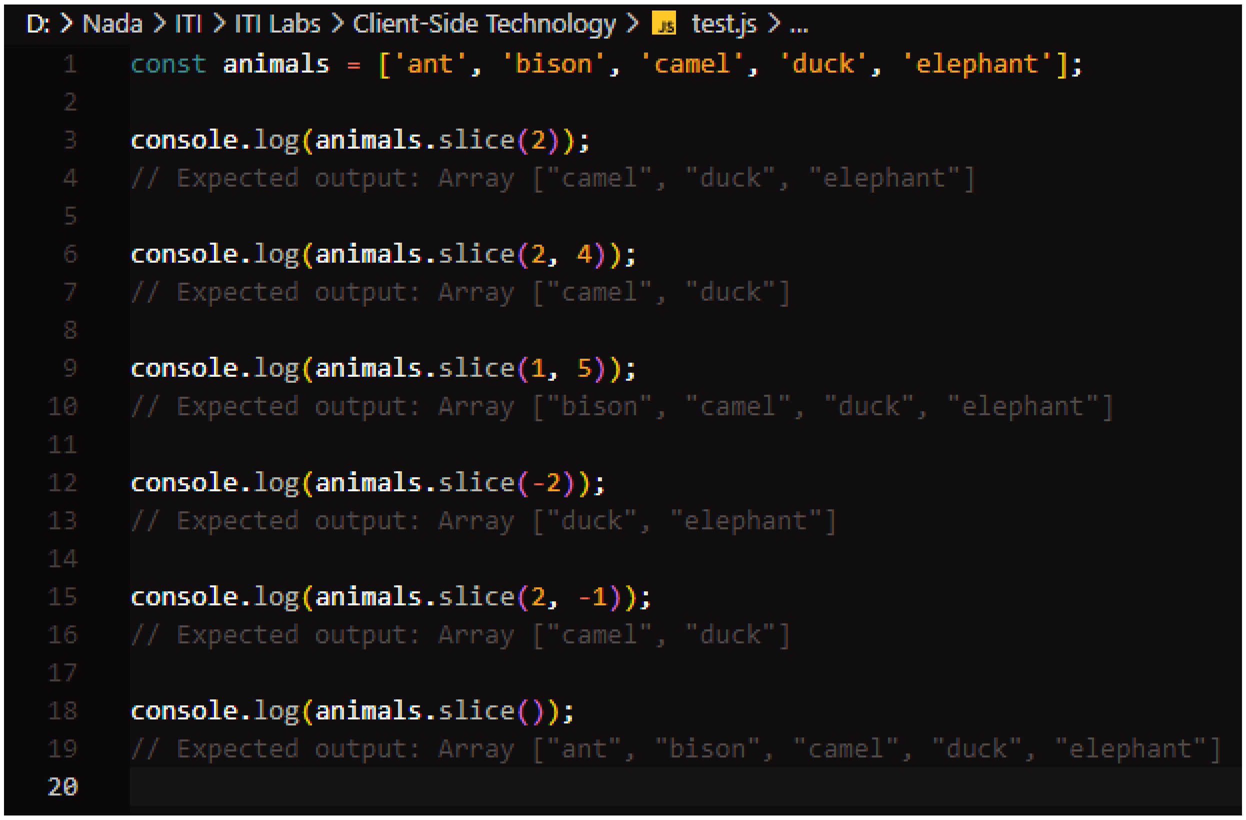


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);

console.log(array3);
// Expected output: Array ["a", "b", "c", "d", "e", "f"]

```

6. slice()

The slice() method of Array instances returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.

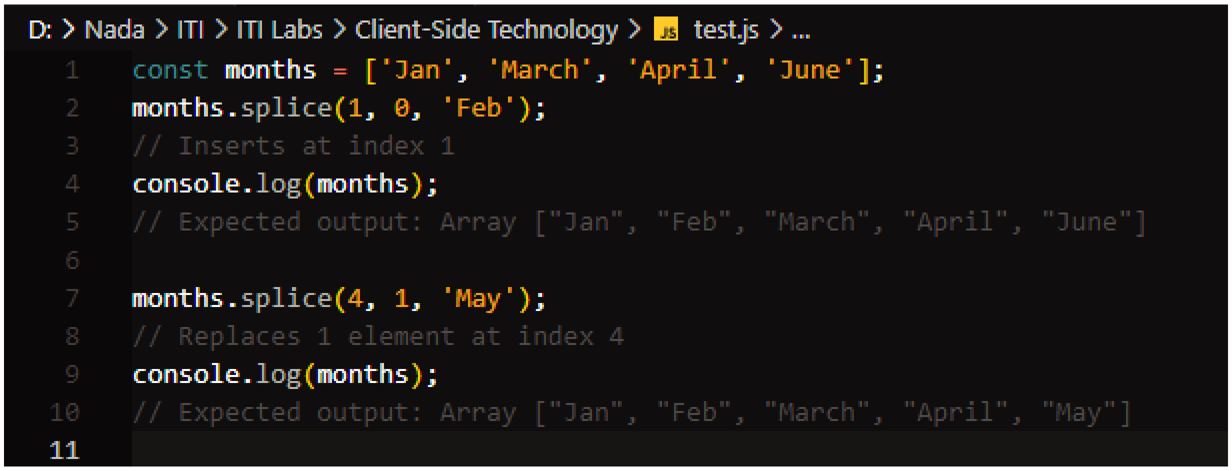


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// Expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// Expected output: Array ["bison", "camel", "duck", "elephant"]

console.log(animals.slice(-2));
// Expected output: Array ["duck", "elephant"]

console.log(animals.slice(2, -1));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice());
// Expected output: Array ["ant", "bison", "camel", "duck", "elephant"]

```

7. splice()

The splice() method of Array instances changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

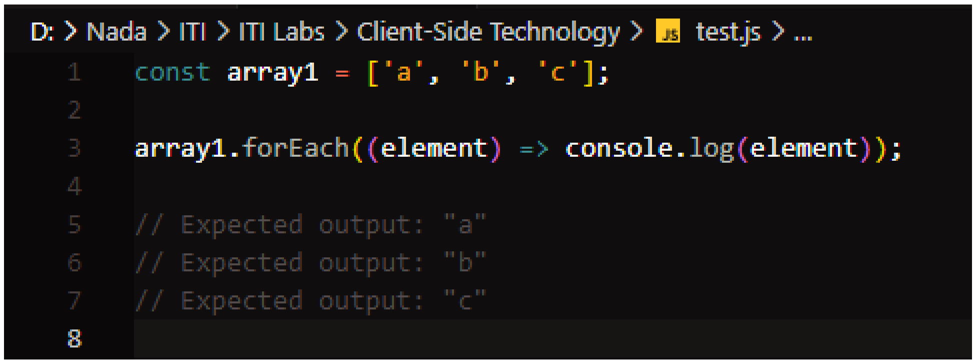


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// Inserts at index 1
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// Replaces 1 element at index 4
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "May"]

```

8. forEach()

The forEach() method of Array instances executes a provided function once for each array element.

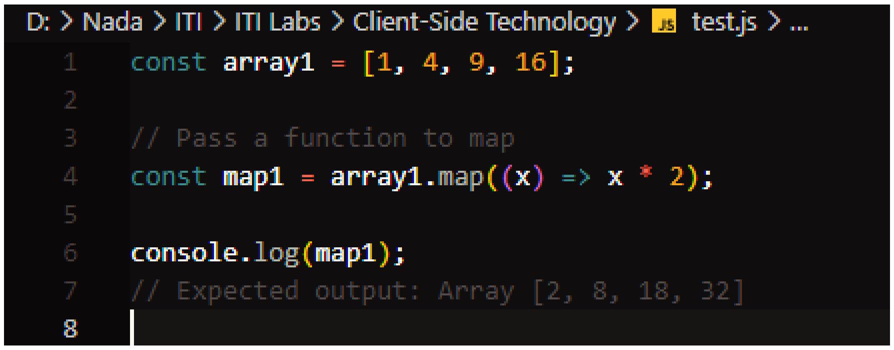


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const array1 = ['a', 'b', 'c'];

array1.forEach((element) => console.log(element));

// Expected output: "a"
// Expected output: "b"
// Expected output: "c"

```

9. map()

The map() method of Array instances creates a new array populated with the results of calling a provided function on every element in the calling array.

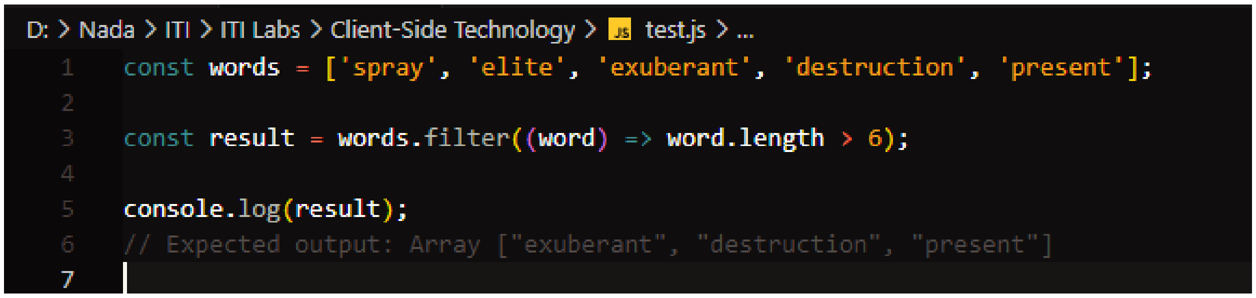


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map((x) => x * 2);

console.log(map1);
// Expected output: Array [2, 8, 18, 32]

```

10. filter()

The filter() method of Array instances creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

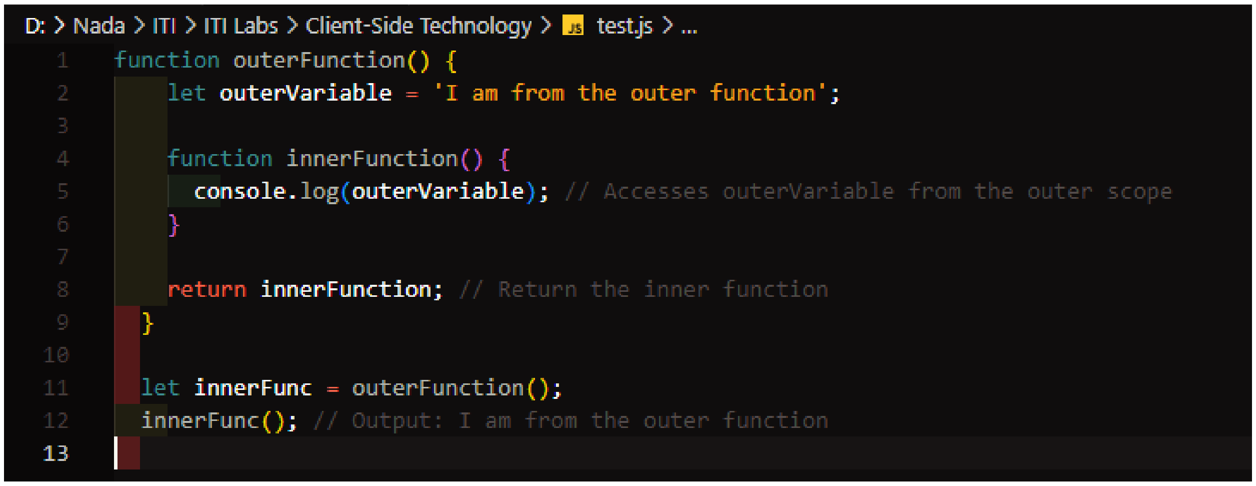


D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
const words = ['spray', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter((word) => word.length > 6);

console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]

```

Closure

In JavaScript, a closure is a combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). This means that a closure allows a function to access variables from its outer scope even after the outer function has finished executing.



D: > Nada > ITI > ITI Labs > Client-Side Technology > test.js > ...

```javascript
function outerFunction() {
    let outerVariable = 'I am from the outer function';

    function innerFunction() {
      console.log(outerVariable); // Accesses outerVariable from the outer scope
    }

    return innerFunction; // Return the inner function
  }

  let innerFunc = outerFunction();
  innerFunc(); // Output: I am from the outer function

```

In this example, innerFunction is a closure because it can access the outerVariable from its enclosing outerFunction even after outerFunction has finished executing. The innerFunc variable, which holds the reference to innerFunction, can be called later to access outerVariable.