Java Script

Introduction to JS

```
var name = "hello" ;                    #variables
var age = 19 ;
var field = "pentrest" ;

console.log(age);
console.log(name) ;
console.log(field) ;                  #print in console
alert(1)
confirm(1)
prompt(1)
```

Comments

```
console.log("Hello from JavaScript"); //in-line comment

/* this is
amulti-line comment */
```

Data types and Variables

```
/*Data Types :
undefined,variables,,null,bool,string,symbol,numberand object */
var name ="M8SZT8";
name = 8;
let ourname = "group";
const pi = 3.14;
```

▼ storing variables with the assignment operator

```
var a ;
var b = 3 ;
console.log(a)
a=7;
b=a;
console.log(a)
CONSOLE
›undefined
›7
```

▼ double quote

```
//as any language
var mystr = "I am a \"double quoted \"string inside\"quoted";
var mystr = 'I am a "double quoted " string inside " quoted';
console.log(mystre)
```

▼ scape sequences

```
/*
code output :
/'   single quote
/"   double quote
//   back slash
/n   new line
/r   carriage run
/t   tab
/b   backspace
/f   formfeed
*/
```

▼ Bracket Notation

```
var name ="M8SZT8" ;
var frstletterofname = "" ;
firstletterofname = name[0] ;
console.log(firstletterofname);
```

▼ string immutability

```
var mystr = "jello world " ;
mystr[0] = "h" //error
mystr = "hello world " //work
console.log(mystr)
```

▼ find Nth character

```
var name="mszt" ;
var lastletterofname = name[name.length-1] ;
console.log(lastletterofname)
var lastletterofname = name[name.length-3] ;
console.log(lastletterofname)
```

▼ Word Blanks

```
function wordBlanks (myNoun, myAdjective, myVerb, myAdverb) {

// Your code below this line
var result = "";
result += "The " + myAdjective + " " + myNoun + " " + myVerb +" " + "to the store"
// Your code above this line
return result;
}
// Change the words here to test your function
console.log(wordBlanks ("dog", "big", "ran", "quickly"));
```

▼ Arrays

```
// Example for Arrays
var ourArray = [["the universe", 42], ["everything", 101010]];
// Only change code below this line.
var myArray = [["Bulls", 23], ["White Sox", 45]];
```

```
-----------------------------------------
// Example for Editting Arrays
var ourArray = [18,64,99];
ourArray[1] = 45; // ourArray now equals [18,45,99].
// Setup
var myArray = [18,64,99];
// Only change code below this line.
myArray[0] = 45;
console.log(myArray)
-------------------------------------------------
// Access Multi-Dimensional Arrays
var myArray [ [1,2,3], [4,5,6], [7,8,9], [[10,11,12], 13, 14] ] ;
// Only change code below this line.
var myData = myArray[2][1] ;
console.log(myData) ;
```

▼ Push()

```
  // Example Push array to other array
var ourArray = ["Stimpson", "J", "cat"];
ourArray.push(["happy", "joy"]);
// ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]
// Setup
var myArray = [["John", 23], ["cat", 2]];
// Only change code below this line. myArray.push(["dog", 3])]
myArray.push(["dog",3])
```

▼ pop()

```
// Example Remove last element in the array

var ourArray = [1,2,3];

var removedFromOurArray = ourArray.pop();

// removedFromOurArray now equals 3, and ourArray now equals [1,2]

// Setup

var myArray = [["John", 23], ["cat", 2]];

// Only change code below this line.

var removedFromMyArray = myArray.pop();

console.log(myArray)
```

shift() && unshift()

```
// Example remove first element from array

var ourArray = ["Stimpson", "J", ["cat"]];

var removedFromOurArray = ourArray.shift();

// removedFromOurArray now equals "Stimpson" and ourArray now equals ["J", ["cat"]]

// Setup

var myArray = [["John", 23], ["dog", 3]];

// Only change code below this line.

var removedFromMyArray = myArray.shift();
--------------------------------------------------
//unshift add as first element
removedFromMyArray.unshift("m8szt8")
```

Functions

```
// Example

function ourReusableFunction() {
    console.log("Heyya, World");
}

ourReusableFunction();

// Only change code below this line

function reusableFunction() {
    console.log("Hi World");
}

reusableFunction();
```

globalscope

```
// Declare your variable here

var myGlobal = 10;

function fun1() {
```

```
// Assign 5 to oopsGlobal Here

oopsGlobal = 5;

}

// Only change code above this line

function fun2() {

var output = "" ;

if (typeof myGlobal != "undefined") {

output += "myGlobal: " + myGlobal;

}

if (typeof oopsGlobal != "undefined") {

output += " oopsGlobal: " + oopsGlobal;

}

console.log(output);

}

fun1();

fun2();
```

▼ global vs local scope in functions

```
var outerwear = "T-Shirt";

function myOutfit() {

var outerwear = "sweater";

return outerwear;

}

console.log(myOutfit());
console.log(outerWear);
```

▼ *IF statement*

```
function testEqual(val) {
  if (val == 12) {
    return "Equal";
  }
  return "Not Equal";
}
console.log(testEqual(10));
------------------------------------------------
// strong equale
function testStrict(val) {
  if (val === 7) { // Change this line
    return "Equal";
  }
  return "Not Equal";
}
testStrict (10);

/*
3 == 3    equale
3 === '3' not equal
!=    not equal ?
||  or
&& and
*/
-----------------------------------------------------
function orderMyLogic (val) {
if (val < 5) {
  return "Less than 5";
  } else if (val < ) {
  return "Less than 5";
  } else {

  return "Greater than or equal to 10";
  }
}
console.log(orderMyLogic(3))
```

▼ *SwitchStatement*

```
function caseInSwitch (val) {
var answer = "";
switch(val) {
case 1:
answer = "alpha";
break;
case 2:
answer = "beta";
break;
case 2: ▶
```

```
answer = "beta";
break;
case 2:
answer = "beta";
break;
}
return answer;
}
---------------------------------
function switchOfStuff (val) {
var answer = "";
  switch (val) {
    case "a":
       answer = "apple";
       break;
    case "b":
       answer = "bird";
    break;
    case "c":
      answer = "cat";
      break;
    default:
      answer = "stuff";
      break;
  }
  return answer;
}
  console.log(switchOfStuff (3))
```

▼ *Objects*

```
var ourDog = {
"name": "Camper",
"legs": 4,
"tails": 1,
"friends": ["everything!"]
};
var myDog = {
"name": "Quincy",
"legs": 3,
"tails": 2,
"friends": []
};
-----------------------------------------------------
```

▼ Dot Notation

```
// Setup
var testObj = {
"hat": "ballcap",
"shirt": "jersey",
"shoes": "cleats"
};
// Only change code below this line
var hatValue = testObj.hat;
var shirtValue = testObj.shirt;
```

▼ Bracket Notation

```
// Setup
var testObj = {
"an entree": "hamburger",
"my side": "veggies",
"the drink": "water"
};
// Only change code below this line
var entreeValue = testObj["an entree"];
var drinkValue = testObj ["the drink"];
```

▼ Variables in objects

```
// Setup
var testObj = {
12: "Namath",
16: "Montana",
19: "Unitas"
};
// Only change code below this line;
var player Number = 16;
// Change this Line
var player = testObj [playerNumber];
// Change this Lin
```

▼ Testing objects

```
// Setup
var myObj = {
gift: "pony",
pet: "kitten",
bed: "sleigh"
};
```

```
function checkObj (checkProp) {
// Your Code Here
if (myObj.hasOwnProperty (checkProp)) {
  return myObj [checkProp];
  } else {
        return "Not Found"
    }
}
// Test your code by modifying these values
console.log(checkObi("gift"));
```

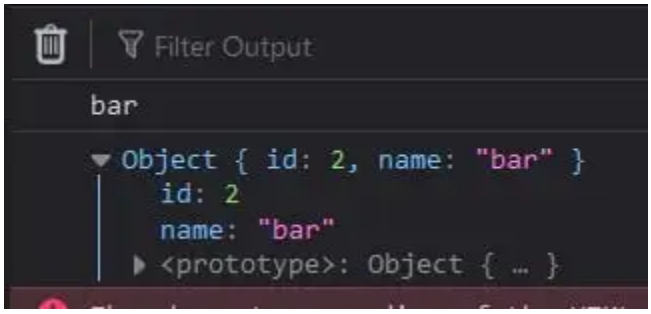
▼ Nested objects

```
// nested objects is objects in objects
const employeeInfo = {
    employeeName: "John Doe",
    employeeId: 27,
    salary: {
        2018-19: "400000INR",
        2019-20: "500000INR",
        2020-21: "650000INR"
    },
    address: {
        locality: {
            address1: "1600 pebble road",
            address2: "Nearby XYZ Bank",
        },
        city: "Mumbai",
        state: "Maharashtra",
        country: "India"
    }
}
```

▼ Nested arrays in objects

```
<!DOCTYPE html>
<html>
<body>

  <script>
   const data = {
    code: 42,
    items: [{
      id: 1,
      name: 'foo'
    }, {
      id: 2,
      name: 'bar'
```

```
    }]
  };

  const item_name = data.items[1].name;

  console.log(item_name)
  console.log(data.items[1])
</script>

</body>
</html>
// output
```



▼ Record Collection

```
// Setup
const recordCollection = {
  2548: {
    albumTitle: "Slippery When Wet",
    artist: "Bon Jovi",
    tracks: ["Let It Rock", "You Give Love a Bad Name"],
  },
  2468: {
    albumTitle: "1999",
    artist: "Prince",
    tracks: ["1999", "Little Red Corvette"],
  },
  1245: {
    artist: "Robert Palmer",
    tracks: [],
  },
  5439: {
    albumTitle: "ABBA Gold",
  },
};

// Only change code below this line
function updateRecords(records, id, prop, value) {
  if (value === "") {
```

```
    delete records[id][prop];
  } else if (prop !== "tracks") {
    records[id][prop] = value;
  } else {
    if (records[id].hasOwnProperty("tracks")) {
      records[id].tracks.push(value);
    } else {
      records[id].tracks = [];
      records[id].tracks.push(value);
    }
  }
  return records;
}

updateRecords(recordCollection, 5439, "artist", "ABBA");
```

▼ *Loops*

```
// Example
var ourArray = [];
for (var i = 10; i > 0; i -= 2) {
ourArray.push(i);
}
console.log(ourArray);
---------------------------------
// Example
var ourArr = [9, 10, 11, 12];
var ourTotal = 0;
for (var i = 0; i < ourArr.length; i++)
{
ourTotal += ourArr[i];
}
console.log(ourTotal);
-------------------------------------------------
do {
myArray.push(i);
i++;
} while (i < 5)
console.log(i, myArray);
```

▼ *LookupProfile*

```
// Setup
var contacts = [
    {
        "firstName": "Akira",
        "lastName": "Laine",
        "number": "0543236543",
```

```
        "likes": ["Pizza", "Coding", "Brownie Points"]
    },
    {
        "firstName": "Harry",
        "lastName": "Potter",
        "number": "0994372684",
        "likes": ["Hogwarts", "Magic", "Hagrid"]
    },
    {
        "firstName": "Sherlock",
        "lastName": "Holmes",
        "number": "0487345643",
        "likes": ["Intriguing Cases", "Violin"]
    },
    {
        "firstName": "Kristian",
        "lastName": "Vos",
        "number": "unknown",
        "likes": ["JavaScript", "Gaming", "Foxes"]
    }
];


function lookUpProfile(name, prop){
// Only change code below this line
for (var i=0; i<contacts.length; i++){

for (var j=0; j<contacts[i].length; j++){

if(contacts[i][0]===name&&contacts[i][j]===prop){ return contacts[i][j]    ;}

else if (contacts[i]===name) {return "No such property";}

else return "No such contact";
}
  }

// Only change code above this line
}

lookUpProfile("Akira", "likes");
```

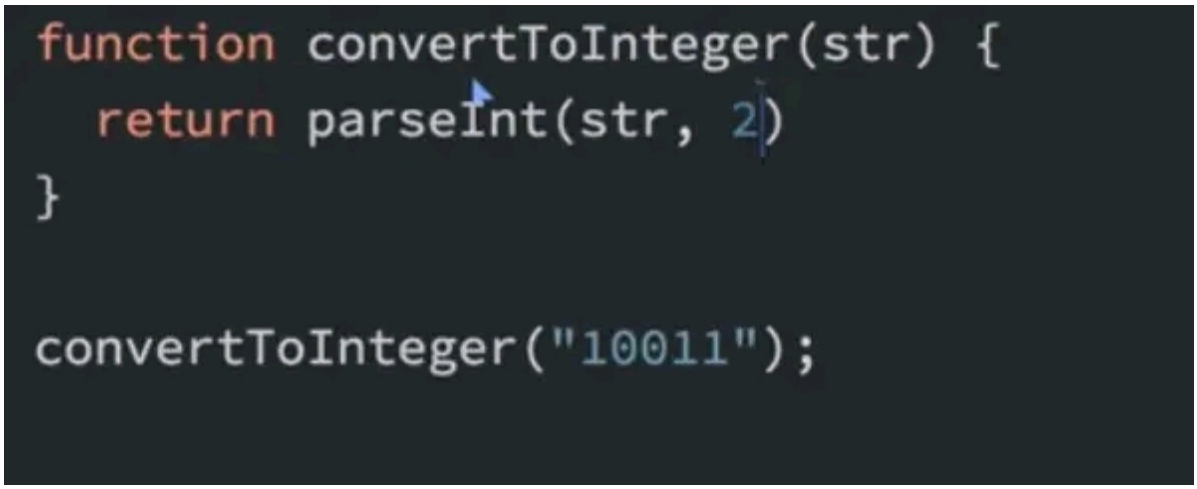
▼ Random number functions

```
// input from the user
const min = parseInt(prompt("Enter a min value: "));
const max = parseInt(prompt("Enter a max value: "));

// generating a random number
const a = Math.floor(Math.random() * (max - min + 1)) + min;
```

```
// display a random number
console.log(`Random value between ${min} and ${max} is ${a}`);
-----------------------------------------------------------
// math.random() function

// generating a random number
const a = Math.random() * (10-1) + 1
console.log(`Random value between 1 and 10 is ${a}`);
```

▼ parseint function

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Global Methods</h1>
<h2>The parseInt() Method</h2>
<p>parseInt() parses a string and returns the first integer:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
parseInt("10", 10)+ "<br>" +
parseInt("010")+ "<br>" +
parseInt("10", 8)+ "<br>" +
parseInt("0x10")+ "<br>" +
parseInt("10", 16);
</script>

</body>
</html>
```



▼ Ternary operator

```
// Syntax
condition ? exprIfTrue : exprIfFalse
//setup
function getFee(isMember) {
  return (isMember ? '$2.00' : '$10.00');
}

console.log(getFee(true));
// expected output: "$2.00"

console.log(getFee(false));
// expected output: "$10.00"

console.log(getFee(null));
// expected output: "$10.00
```

▼ let vs var vs const

```
//var is function scoped when it is declared within a function. This means that it is
available and can be accessed only within that function.

//To understand further, look at the example below.

    var greeter = "hey hi";

    function newFunction() {
        var hello = "hello";
    }
-----------------------------------------------------
var greeter = "hey hi";
    var times = 4;

    if (times > 3) {
        var greeter = "say Hello instead";
    }

    console.log(greeter) // "say Hello instead"
-----------------------------------------------------
let is block scoped

// A block is a chunk of code bounded by {}. A block lives in curly braces. Anything
 within curly braces is a block.
// So a variable declared in a block with let  is only available for use within that
 block. Let me explain this with an example:
   let greeting = "say Hi";
   let times = 4;
```

```
   if (times > 3) {
        let hello = "say Hello instead";
        console.log(hello);// "say Hello instead"
    }
   console.log(hello) // hello is not defined
-------------------------------------------------------
let can be updated but not re-declared.

// Just like var,  a variable declared with let can be updated within its scope. Unli
ke var, a let variable cannot be re-declared within its scope.

//So while this will work:

    let greeting = "say Hi";
    greeting = "say Hello instead";

//this will return an error:

    let greeting = "say Hi";
    let greeting = "say Hello instead"; // error: Identifier 'greeting' has already b
een declared
-------------------------------------------------
const declarations are block scoped

//Like let declarations, const declarations can only be accessed within the block the
y were declared.

const cannot be updated or re-declared

    const greeting = "say Hi";
    greeting = "say Hello instead";// error: Assignment to constant variable.
--------------------------------------
// ubdate const objects
const greeting = {
        message: "say Hi",
        times: 4
    }

// while we cannot do this:

    greeting = {
        words: "Hello",
        number: "five"
    } // error:  Assignment to constant variable.

//we can do this:

    greeting.message = "say Hello instead";
```

▼ Object.freeze()function

```
const obj = {
  prop: 42
};

Object.freeze(obj);

obj.prop = 33;
// Throws an error in strict mode

console.log(obj.prop);
// expected output: 42
```

▼ Arrow Function

```
const materials = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];

console.log(materials.map(material => material.length));
// expected output: Array [8, 6, 7, 9]
---------------------------------------
// Traditional Anonymous Function
(function (a) {
  return a + 100;
});

// Arrow Function Break Down

// 1. Remove the word "function" and place arrow between the argument and opening bod
y bracket
(a) => {
  return a + 100;
};

// 2. Remove the body braces and word "return" — the return is implied.
(a) => a + 100;

// 3. Remove the argument parentheses
a => a + 100;
// Traditional Anonymous Function (no arguments)
(function() {
  return a + b + 100;
});

// Arrow Function (no arguments)
() => a + b + 100;
```

```
------------------------------------------------
  (params) => ({ foo: "a" }) // returning the object { foo: "a" }
```

▼ Default parametars

```
function multiply(a, b = 1) {
  return a * b;
}

console.log(multiply(5, 2));
// expected output: 10

console.log(multiply(5));
// expected output: 5
```

▼ Rest parameters

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}

console.log(sum(1, 2, 3));
// expected output: 6

console.log(sum(1, 2, 3, 4));
// expected output: 10
```

▼ Spreed Operator

▼ Destructuring assignment

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]
```

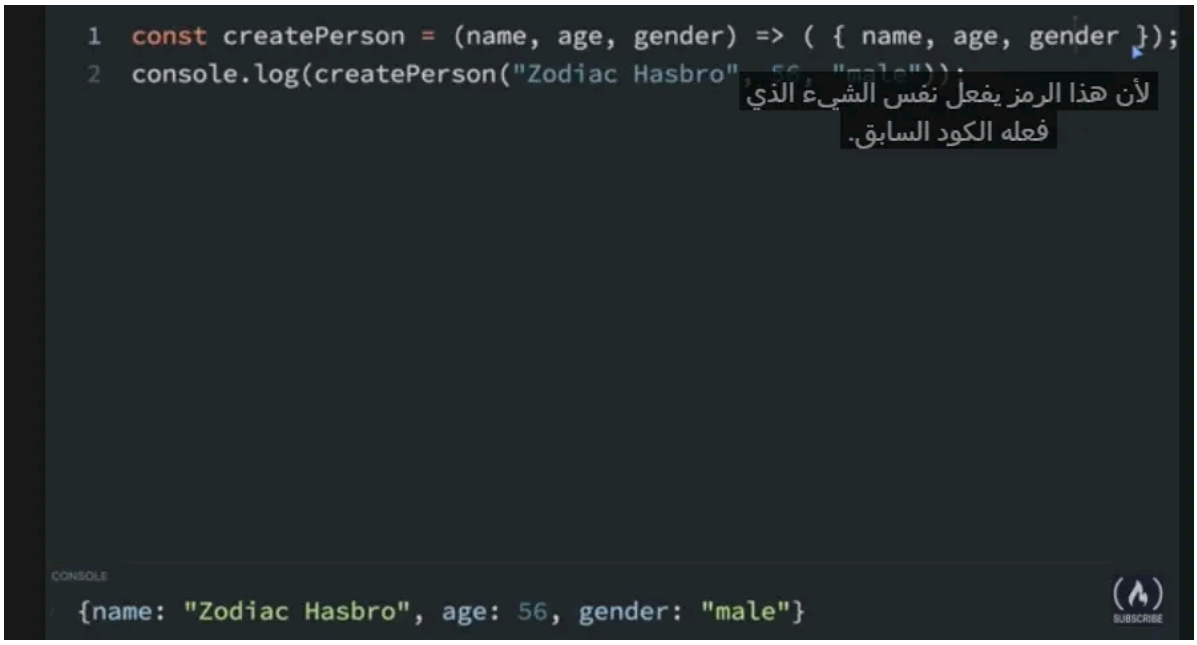
▼ Template literals (Template strings)

```
//Dollar signs can be escaped as well to prevent interpolation.
`\${1}` === '${1}' // --> true
------------------------------
console.log('string text line 1\n' +
'string text line 2');
// "string text line 1
// string text line 2"
------------------------------
//String interpolation
```

```
//Without template literals, when you want to combine output from expressions with st
rings, you'd concatenate them using the addition operator +:

const a = 5;
const b = 10;
console.log('Fifteen is ' + (a + b) + ' and\nnot ' + (2 * a + b) + '.');
// "Fifteen is 15 and
// not 20."
-----------------------
const t = Temporal.Now.instant();
"" + t; // Throws TypeError
`${t}`; // '2022-07-31T04:48:56.113918308Z'
const person = 'Mike';
const age = 28;

function myTag(strings, personExp, ageExp) {
  const str0 = strings[0]; // "That "
  const str1 = strings[1]; // " is a "
  const str2 = strings[2]; // "."

  const ageStr = ageExp > 99 ? 'centenarian' : 'youngster';

  // We can even return a string built using a template literal
  return `${str0}${personExp}${str1}${ageStr}${str2}`;
}

const output = myTag`That ${person} is a ${age}.`;

console.log(output);
// That Mike is a youngster.
-------------------------------------------------------------------
Tag functions don't even need to return a string!

function template(strings, ...keys) {
  return (...values) => {
    const dict = values[values.length - 1] || {};
    const result = [strings[0]];
    keys.forEach((key, i) => {
      const value = Number.isInteger(key) ? values[key] : dict[key];
      result.push(value, strings[i + 1]);
    });
    return result.join('');
  };
}

const t1Closure = template`${0}${1}${0}!`;
// const t1Closure = template(["","","","!"],0,1,0);
t1Closure('Y', 'A');                      // "YAY!"

const t2Closure = template`${0} ${'foo'}!`;
// const t2Closure = template(["","," ","!"],0,"foo");
t2Closure('Hello', { foo: 'World' }); // "Hello World!"

const t3Closure = template`I'm ${'name'}. I'm almost ${'age'} years old.`;
```

```
// const t3Closure = template(["I'm ", ". I'm almost ", " years old."], "name", "ag
e");
t3Closure('foo', { name: 'MDN', age: 30 }); // "I'm MDN. I'm almost 30 years old."
t3Closure({ name: 'MDN', age: 30 }); // "I'm MDN. I'm almost 30 years old."
----------------------------------------------------------------
Tag functions don't even need to return a string!

function template(strings, ...keys) {
  return (...values) => {
    const dict = values[values.length - 1] || {};
    const result = [strings[0]];
    keys.forEach((key, i) => {
      const value = Number.isInteger(key) ? values[key] : dict[key];
      result.push(value, strings[i + 1]);
    });
    return result.join('');
  };
}

const t1Closure = template`${0}${1}${0}!`;
// const t1Closure = template(["","","","!"],0,1,0);
t1Closure('Y', 'A');                      // "YAY!"

const t2Closure = template`${0} ${'foo'}!`;
// const t2Closure = template(["","," ","!"],0,"foo");
t2Closure('Hello', { foo: 'World' }); // "Hello World!"

const t3Closure = template`I'm ${'name'}. I'm almost ${'age'} years old.`;
// const t3Closure = template(["I'm ", ". I'm almost ", " years old."], "name", "ag
e");
t3Closure('foo', { name: 'MDN', age: 30 }); // "I'm MDN. I'm almost 30 years old."
t3Closure({ name: 'MDN', age: 30 }); // "I'm MDN. I'm almost 30 years old."
```

▼ Simple fields



▼ **Declarative functions**

▼ Class Syntax

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
---------------------------------
// unnamed
let Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
console.log(Rectangle.name);
// output: "Rectangle"

// named
Rectangle = class Rectangle2 {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
console.log(Rectangle.name);
```

```
// output: "Rectangle2"
----------------------------------------------------------------------
```

```
class SpaceShuttle {
  constructor(targetPlanet){
    this.targetPlanet = targetPlanet;
  }
}
var zeus = new SpaceShuttle('Jupiter');

console.log(zeus.targetPlanet)
```

```
function makeClass() {
  class Vegetable {
    constructor(name){
      this.name = name;
    }
  }
  return Vegetable;
}
const Vegetable = makeClass();
const carrot = new Vegetable('carrot');
console.log(carrot.name);
```

```
CONSOLE
Jupiter
```

▼ Getter and Setter

```
const language = {
  set current(name) {
    this.log.push(name);
  },
  log: []
};

language.current = 'EN';
```

```
language.current = 'FA';

console.log(language.log);
// expected output: Array ["EN", "FA"]
-------------------
//Syntax
{ set prop(val) { /* … */ } }
{ set [expression](val) { /* … */ } }
```

```
// Getter
const obj = {
  log: ['a', 'b', 'c'],
  get latest() {
    return this.log[this.log.length - 1];
  }
};

console.log(obj.latest);
// expected output: "c"

------------------
const obj = {
  log: ['example','test'],
  get latest() {
    if (this.log.length === 0) return undefined;
    return this.log[this.log.length - 1];
  }
}
console.log(obj.latest); // "test"
--------------------------------
//syntax
{ get prop() { /* … */ } }
{ get [expression]() { /* … */ } }
```

▼ import and export

```
//import
//syntax
import defaultExport from "module-name";
import * as name from "module-name";
import { export1 } from "module-name";
import { export1 as alias1 } from "module-name";
import { default as alias } from "module-name";
import { export1, export2 } from "module-name";
import { export1, export2 as alias2, /* … */ } from "module-name";
import { "string name" as alias } from "module-name";
import defaultExport, { export1, /* … */ } from "module-name";
import defaultExport, * as name from "module-name";
```

```
import "module-name";
---------------------------------------
//export
//syntax
// Exporting declarations
export let name1, name2/*, … */; // also var
export const name1 = 1, name2 = 2/*, … */; // also var, let
export function functionName() { /* … */ }
export class ClassName { /* … */ }
export function* generatorFunctionName() { /* … */ }
export const { name1, name2: bar } = o;
export const [ name1, name2 ] = array;

// Export list
export { name1, /* …, */ nameN };
export { variable1 as name1, variable2 as name2, /* …, */ nameN };
export { variable1 as "string name" };
export { name1 as default /*, … */ };

// Default exports
export default expression;
export default function functionName() { /* … */ }
export default class ClassName { /* … */ }
export default function* generatorFunctionName() { /* … */ }
export default function () { /* … */ }
export default class { /* … */ }
export default function* () { /* … */ }

// Aggregating modules
export * from "module-name";
export * as name1 from "module-name";
export { name1, /* …, */ nameN } from "module-name";
export { import1 as name1, import2 as name2, /* …, */ nameN } from "module-name";
export { default, /* …, */ } from "module-name";
```

index.js

```
1  const capitalizeString = (string) => {
2    return string.charAt(0).toUpperCase() + string.slice(1);
3  }
4
5  export { capitalizeString };
6
7  export const foo = "bar";
8  export const bar = "foo";
```

الآن في هذا الملف ، نقوم بتصدير هذه الدالة وهذين المتغيرين.

```
1  import * as capitalizeStrings from "capitalize_strings";
```

```
1  import subtract from "math_functions";
2
3  subtract(7,4);
```

حسنًا ، هذه هي الطريقة التي ستستورد بها تصديرًا افتراضيًا.

▼ *JSON*

```
// JSON is text, and text can be transported anywhere, and read by any programming la
nguage.
//JavaScript Objects can be converted into JSON, and JSON can be converted back into
 JavaScript Objects.
//This way we can work with the data as JavaScript objects, with no complicated parsi
ng or translations.
//JSON is a format for storing and transporting data.
<!DOCTYPE html>
<html>
<body>

<h2>Store and retrieve data from local storage.</h2>
```

```
<p id="demo"></p>

<script>
var myObj, myJSON, text, obj;

//Storing data:
myObj = { "name":"John", "age":31, "city":"New York" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

//Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>

</body>
</html>
```